Hoofdstuk 1

Handleiding voor de U-Touch HMI middels de Software van EASY xLogic

Dit Hoofdstuk beschrijft de basis functies en de karaterestiek van de Easy HMI software, the industrial control configuration software (referred to as EASY below).

It describes in details the system architecture and the functions of each component, which helps the user better understand the general picture of EASY. Besides, it describes the hardware and software requirements, installation process, and operational environment for the operation of EASY.

1.1 Overzicht

EASY is a cofiguration software developed on the supervisory computer of the the EASY Human-Machine Interface (HMI). It collects real-time data, and processes it using multiple technics, such as animation display, alarming, process control, real-time curve, historical curve, and report output, with the help of which solutions are provided for solving real project issues.

EASY combines the functions of the previous configuration software and HMI, and thus is more powerful than the ordinary touch screen, and more widely applied in the automation field.

EASY has the following basic characteristics and functions:

- Simple and flexible visual operation interface

EASY adopts the visual development interface – windows to display the system operation graphics, which makes the configuration process simple, direct, and flexible. You can use the default system architecture, or conFiguur your own graphical interfaces according to your needs. Various types and styles of graphical interfaces are available for your selection.

- Rich and vivid multimedia graphics

EASY provides real-time feedback to the operator regarding the system operation status, performance, and exception alarms in multiple forms such as graphics, symbols, reports, and curves.

Changes in the graphic size and color, flashing brightness, and graphic moving and rotating all enhance the dynamic visual effects. The animation effects can be realized by setting the properties of the graphic components and symbols.

Besides, EASY provides various WINDOWS programming controls, which makes programming easier for programmers.

- Powerful network capabilities

EASY supports multiple network architectures, such as those based on TCP/IP, Modem, and RS-485/RS-232.

- Various alarming functions

EASY provides various ways of alarming, supported by a rich variety of alarm types and flexible alarm processing functions. This not only makes the alarm setting more convenient, but also realizes the real-time system reflection and alarm information printing.

Besides, EASY keeps records of all alarm information and responses, which ensures to a great extent safe and reliable operation on the site.

- Real-time database, which makes the customized configuration much easier

The real-time database is a centralized data processing center. It is the core of the whole system, and is shared by various system parts and their functional components. Each part or component of the system inputs and exports data to and from the database separately, and each has its own error control system. During the configuration of an application system, each part can be conFiguurd and created independently; while all of the parts exchange data through the database and work interrelatedly during the whole system operation.

- Stable and reliable data management through database

EASY uses the dedicated database to keep all of the data, instead of using files for data storage.

During the configuration, the system designer needs to create a database manually. However, during the system operation, a database is generated automatically to keep and process all system data and alarm information.

The adoption of database for data storage and processing enhances the system reliability and operation efficiency. Besides, it facilitates other application software systems for directly processing the data stored in the database.

- Distributed control and management of the industrial control system

In consideration of future development tendency of the industrial control system, EASY makes full use of the currently popular DCCW (Distributed Computer Cooperator Work) technnology, to make sure that the data collecting devices and workstations distributed in different sites can cooperate with each other. This allows the various work stations to exchange real-time data through the EASY system, and thus ahieves the distributed control and management of the industrial control system.

In summary, EASY is a powerful and simple configuration software. Even ordinary engineers can easily learn and master the designing and operation of most projects after a short period of training. In addition, EASY manages to focus on solving project issues, bypassing complicated hardware and software obstacles. In a word, EASY can always come out with highly professional industrial control monitoring systems of high performance and reliability which suit the particular project needs and characteristics.

1.2 System Components

After the EASY software is installed, a program group named **EASY Industrial Control Software** will be generated in **Programs** of the **Start** menu.

This program group includes shortcuts for the following applications:

- **Project Manager**: The shortcut for EASY Industrial Control Software Project Manager. It has the functions of project management, database management, control blocks, and project uploading and downloading.
- **Ladder Diagram Editor**: The shortcut for the soft PLC ladder diagram development program.
- **Graphic Interface Editor**: The shortcut for the HMI configuration program. It has the configuration interface development system embedded.
- **Historical Data Converter**: To maximally compress the data volume contained in the HMI and store the historical data the most, the EASY system stores the historical data in a DAT file. You can download this file from the HMI, and then use this **Historical Data Converter** to convert the data into an Excel file.
- **Commissioning Output Background**: You can add printing commissioning information into the dynamic script. This printing commissioning information will be exported to the commissioning window of **Commissioning Output Background**.
- **Real-Time Data Monitoring**: This tool helps you query and modify the data stored in the real-time database of the HMI.

1.3 System Architecture

The EASY system architecture consists of three parts: configuration environment, simulated operational environment, and operational environment.

The configuration environment and simulated operational environment combined can be considered as a complete set of software tools. It can run on the PC, and you can customize the system components according to your needs. It is qualified enough to help you design and conFiguur your own projects, and accomplish function testing as well.

The operational environment is an independent system. It can only run on the HMI. It processes the project configuration according to your customization, and fulfills the goals and functions of your configuration designing. The operational environment does not mean anything by itself; its meaning is realized by configuration projects, namely, the configuration of the user application system. Once the configuration is done and the conFiguurd project is downloaded to the operational environment of the Programmable Logic Controller (PLC) via Ethernet, the project can run independently on the HMI without any interference of the configuration environments, which makes the control system reliable, real-time, accurate, and secure.

Real-Time Database – Core of EASY

The real-time database can be considered as a data processing center. Besides, it also carries out the function of public data exchange.

EASY uses the real-time database to manage all the real-time data. It takes in the real-time data collected by the external device into the real-time database, which then transfers the data to the data variables correspondent to various configuration interfaces of the system.

The real-time database automatically implements the alarm processing and saving of the real-time data. After that, it accordingly sends the related information to the other parts of the system in the form of events to trigger the related events for real-time handling.

Therefore, the data units stored in the real-time database are not only values of the variables, but also characteristics (or properties) of the variables and operating methods (such as alarm properties analyzing, alarm processing, and saving). The encapsulation of values, properties, and methods together is called a data object.

The real-time database uses exactly the object-oriented technology to provide services to the other parts of the system; for example, the data exchange between the various functional compoents of the system.

1.4 System Requirements

Hardware Requirements

- Hardware: Pentium PIII 500 or higher IBM PCs or compatible PCs.
- Memory: 64MB the minimum, and 128MB recommended.
- Monitor: VGA, SVGA, or any other graphics adapter which supports the running of

the desktop operating system. The minimum request is to support 256 colors.

- Mouse: Any mouse which can be used on a PC.
- Communication: RS-232C
- Operating system: Win2000/WinNT4.0 (patch 6)/Win XP English version

Software Requirements

EASY can run in the following operating systems:

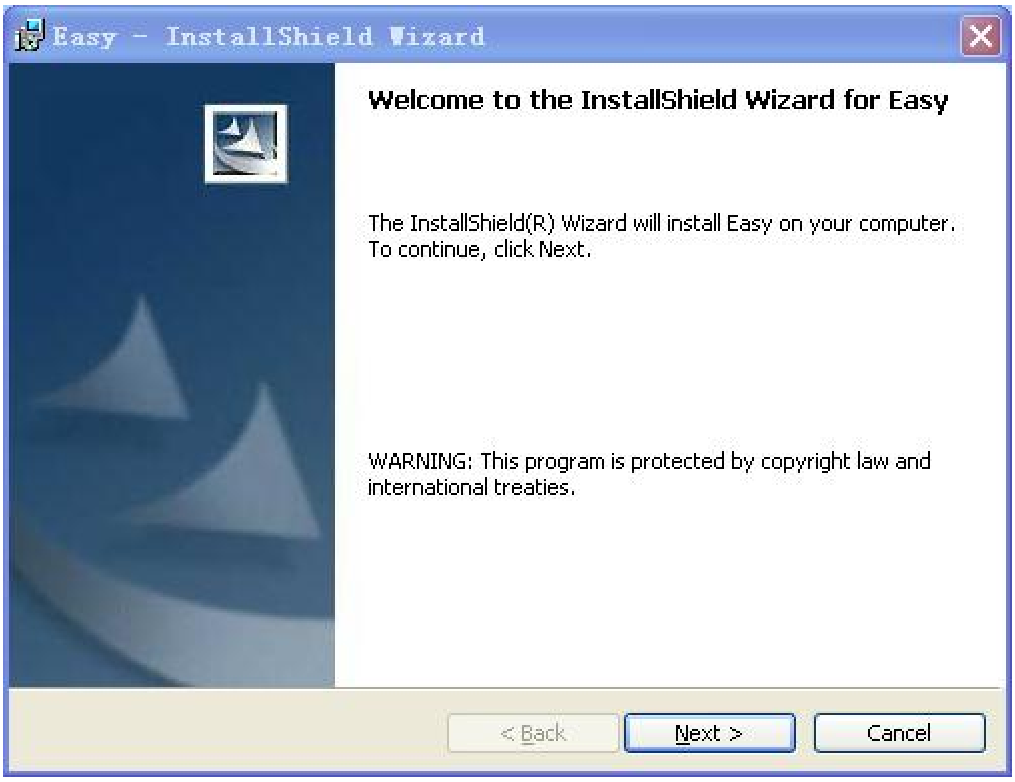
- Microsoft Windows NT Server 4.0 (SP3) or higher versions
- Microsoft Windows NT Workstation 4.0 (SP3) or higher version
- Microsoft Windows 95, 98, Me, 2000 (IE5.0 recommended for Windows 95) or higher versions

1.5 System Installation

The EASY software is stored on the disk. After you insert the disk into the CD/DVD drive of your PC, the installer **setup.exe** will start running atomatically, and initiate the EASY installation wizard.

To install EASY, please follow the steps below (Take WinXP installation for example; same installation procedure for WinNT4.0 and Win2000):

1) Start the PC.
2) Insert the EASY installation disk into the CD/DVD drive.
   The installer **EASYSoftware.exe** will start running automatically, as shown below in Figuur 1.1. (You can also double-click on the **EASYSoftware.exe** file to initiate the installation.)



Easy - InstallShield Wizard

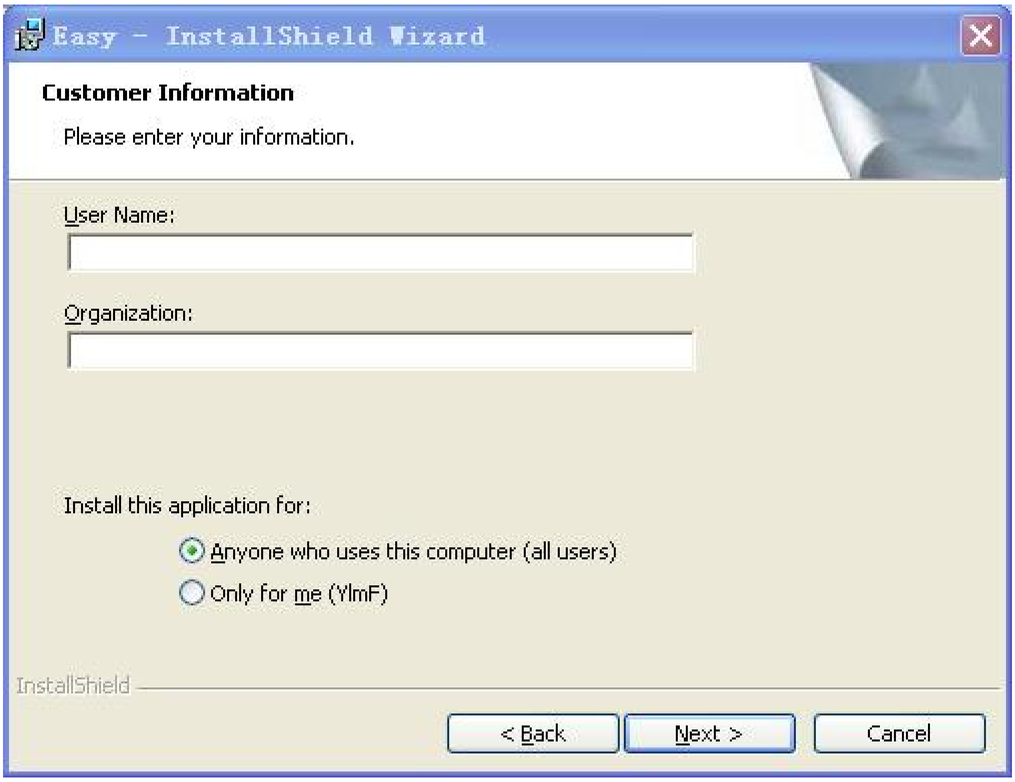
Welcome to the InstallShield Wizard for Easy

The InstallShield(R) Wizard will install Easy on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

Figuur 1.1

3) Click on the **Next** button, and you will see the user information dialog box, as shown below in Figuur 1.2.
   Type in the information for **Username** and **Company**. Click on the **Back** button to go back to the previous dialog box, and click on the **Cancel** button to exit the installation.



Easy - InstallShield Wizard

Customer Information

Please enter your information.

User Name:

Organization:

Install this application for:

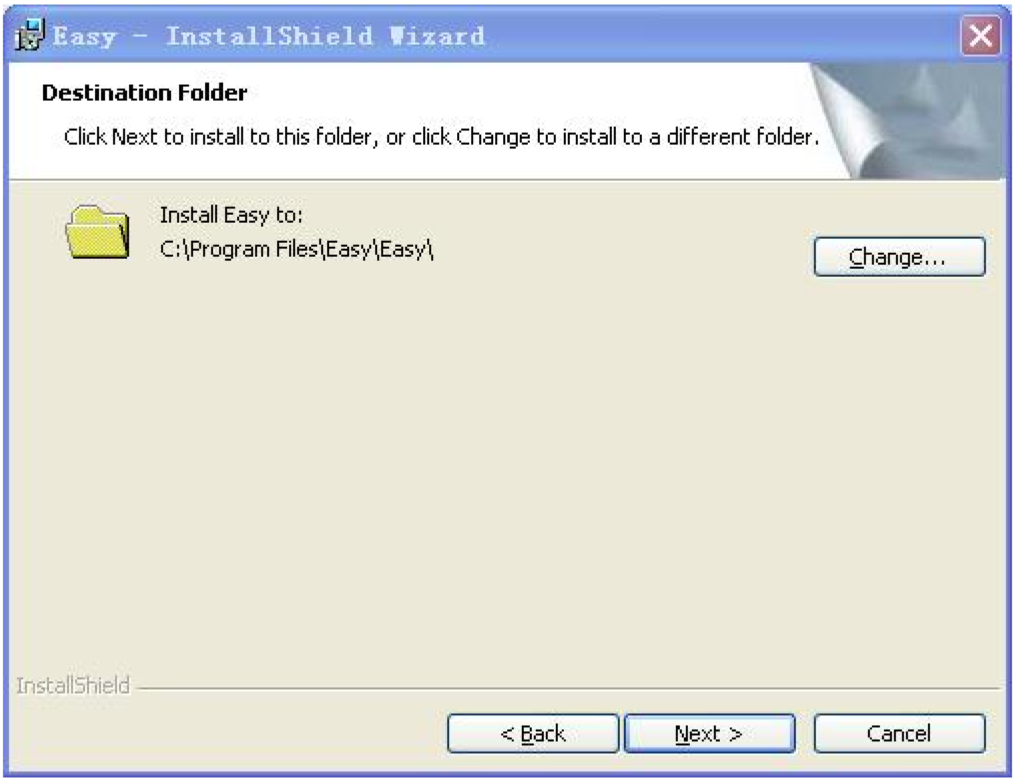
● Anyone who uses this computer (all users)

○ Only for me (YlmF)

Figuur 1.2

4) Select the installation path for EASY.

After you confirm the user registration information, the **Destination Folder** dialog box will be displayed for you to select the installation path, as shown in Figuur 1.3. This dialog box tells you in which path the EASY software will be installed. The default path is **C:\Program Files\EASY\EASY Industrial Control Software**. Click on the **Change** button to install EASY in another path.


Easy - InstallShield Wizard

Destination Folder

Click Next to install to this folder, or click Change to install to a different folder.

Install Easy to:
C:\Program Files\Easy\Easy\

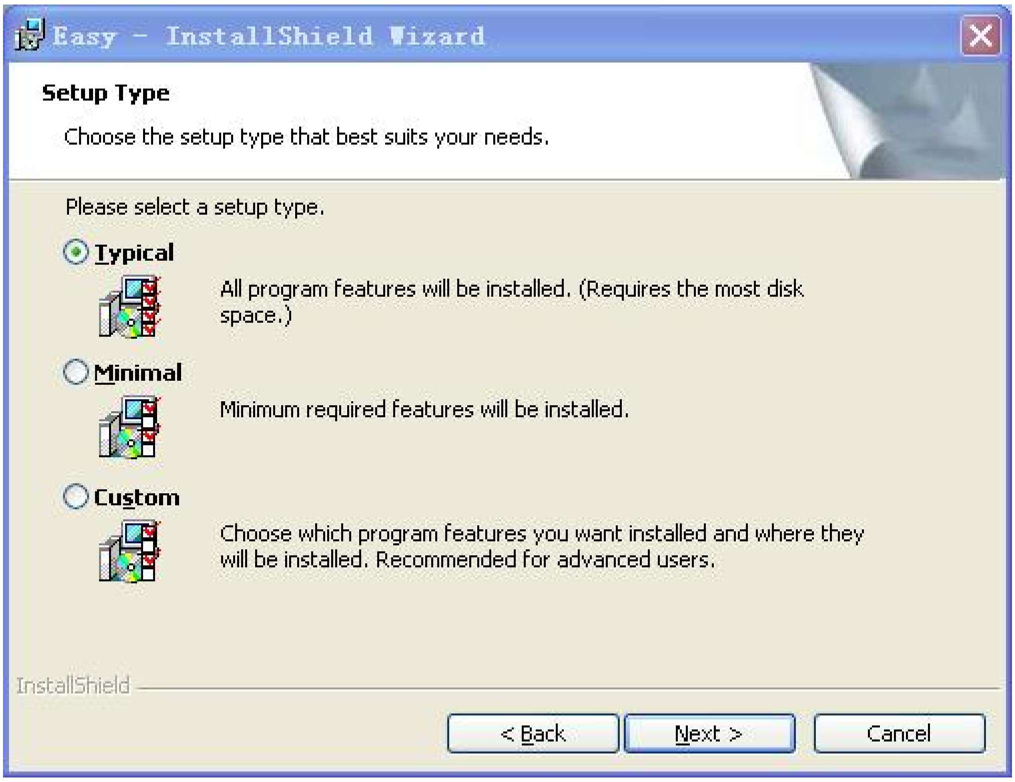
Figuur 1.3

The installer will create a destination folder in the selected path. The folder name is **EASY Industrial Control Software**.

You can always click on the **Back** button to make changes. Otherwise, click on the **Next** button to continue the installation or the **Cancel** button to exit the installation.

5) Select the installation type, as shown in Figuur 1.4.

Select the installation type that best suits your needs.



Easy - InstallShield Wizard

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

● Typical

All program features will be installed. (Requires the most disk space.)

○ Minimal

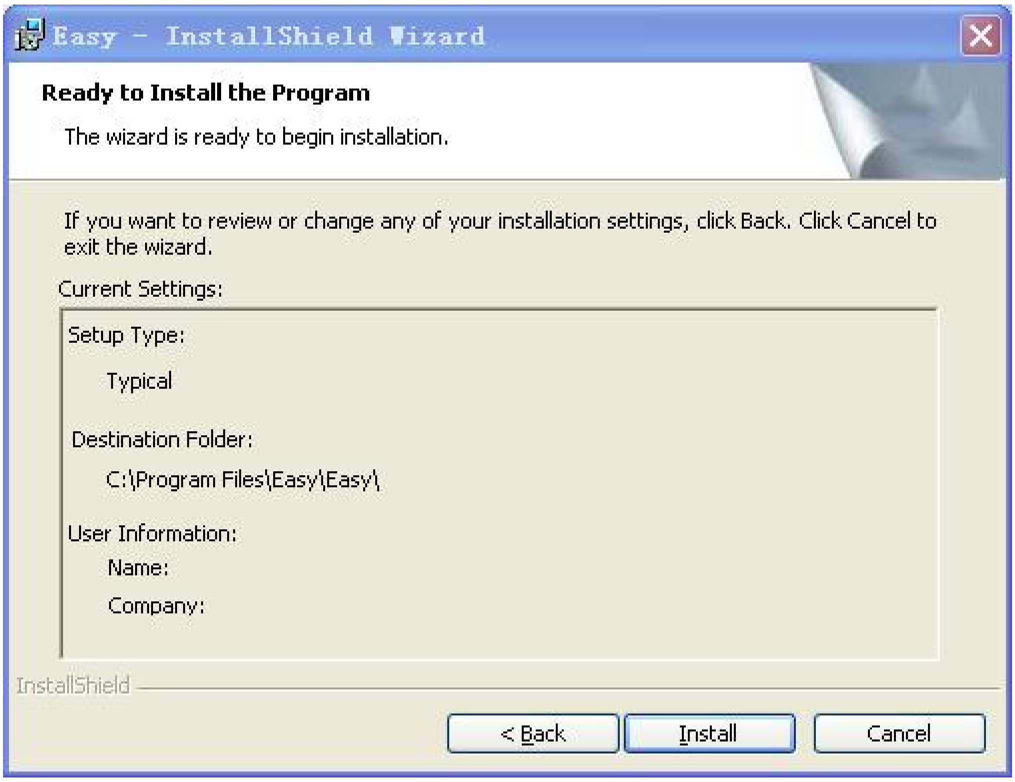
Minimum required features will be installed.

○ Custom

Choose which program features you want installed and where they will be installed. Recommended for advanced users.

Figuur 1.4

After you select the installation type, click on the **Next** button. And the following ready-for-installation window, as shown in Figuur 1.5, will be displayed. Click on the **Install** button to start the installation.



Easy - InstallShield Wizard

Ready to Install the Program

The wizard is ready to begin installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Current Settings:

Setup Type:

Typical

Destination Folder:

C:\Program Files\Easy\Easy\

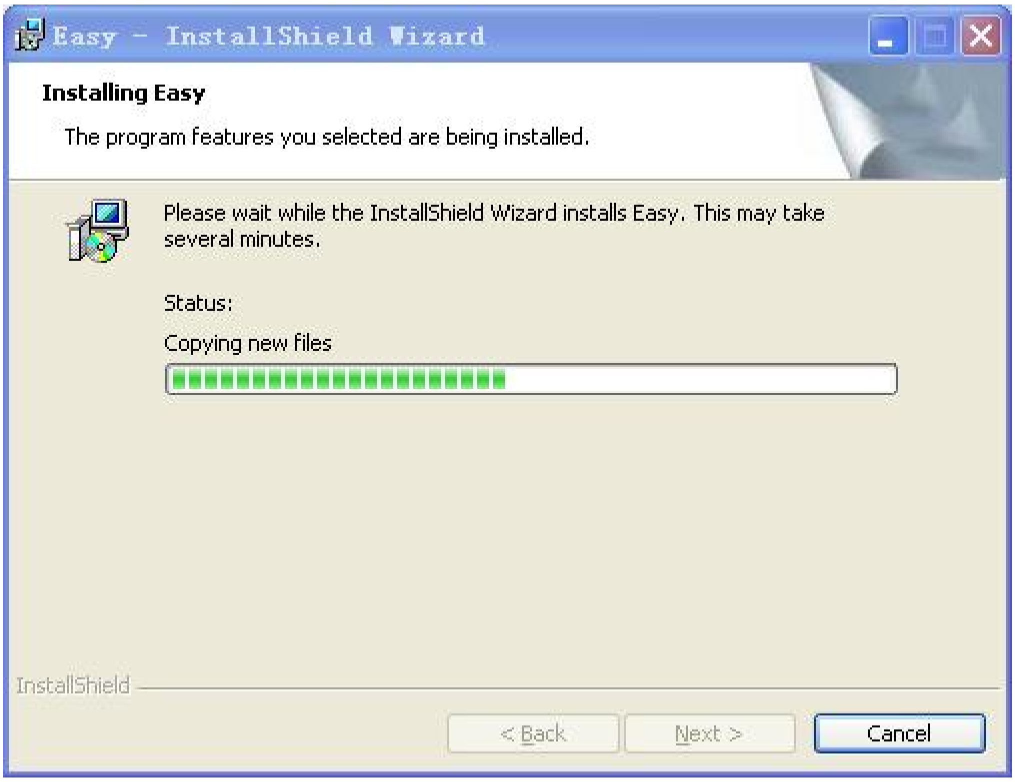
User Information:

Name:

Company:

Figuur 1.5

6) Installation started, as shown in Figuur 1.6.



Easy - InstallShield Wizard

Installing Easy

The program features you selected are being installed.

Please wait while the InstallShield Wizard installs Easy. This may take several minutes.

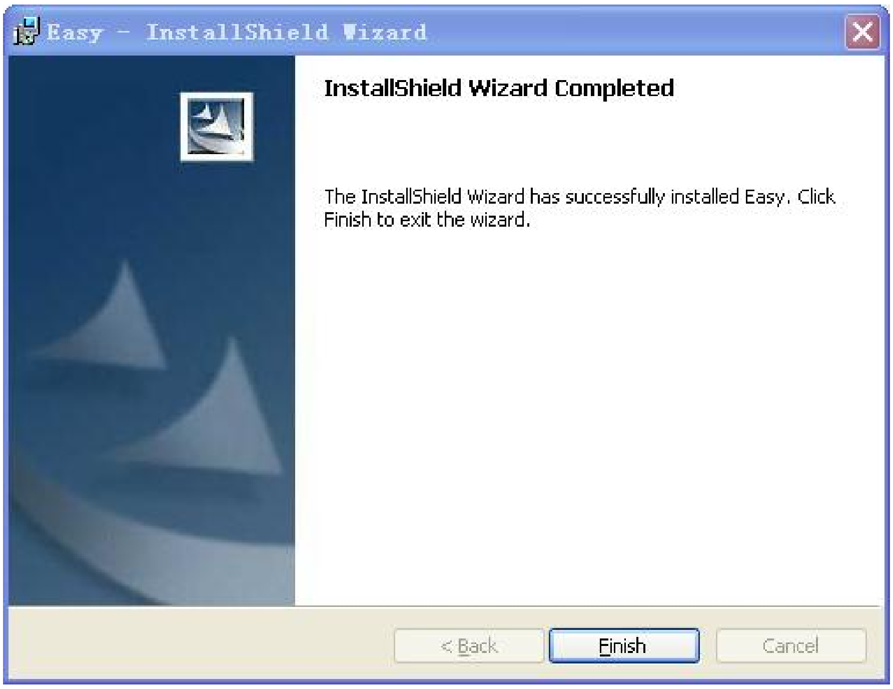
Status:

Copying new files

Figuur 1.6

7) Installation complete, as shown in Figuur 1.7. Click on the **Finish** button to exit the installation wizard.

The EASY software will be initiated for the first time.



Easy - InstallShield Wizard

InstallShield Wizard Completed

The InstallShield Wizard has successfully installed Easy. Click Finish to exit the wizard.

Figuur 1.7

Chapter 2 Getting Started with EASY

This Hoofdstuk takes a project as an example to explain comprehensively the configuration process, operation methods, and function implementation of EASY. It provides you a gneral picture regarding what EASY is, how it works, and how to use it in a short period of time.

2.1 General Procedure for Creating a Project

The general procedure for setting up a project is listed as follows:

1) Create a blank project.
2) Create a database, and add real-time data into the database.
3) ConFiguur devices on the site.
4) Design the graphic interface.
5) Set the dynamic properties of the interface to achieve rich animation effects.
6) Compile the project, and run it in the offline simulated einvironment.
7) Download the project.

One thing to be noted, the steps stated above do not follow a strict sequence. As a matter of fact, the implementation of them might often be overlapped.

Therefore, it is recommended to take the following three aspects into consideration while configuring a project using the software interface development system:

- What kind of graphic interfaces are expected by the user for whom the interfaces are developed?
  In other words, how to use abstract graphic interfaces to simulate real industrial sites and the industrial cotrol (IC) devices installed on each site?
- What data variables can be used to describe the various properties of IC objects?
  A database will be created for a project. But how to map database variables to various properties of IC objects; for example, temperature and pressure.
- How to link the data and components in graphic interfaces?
  That is to say, how to make components in graphic interfaces reflect the operation status of devices installed in real industrial sites, and how the operator types in device control commands?

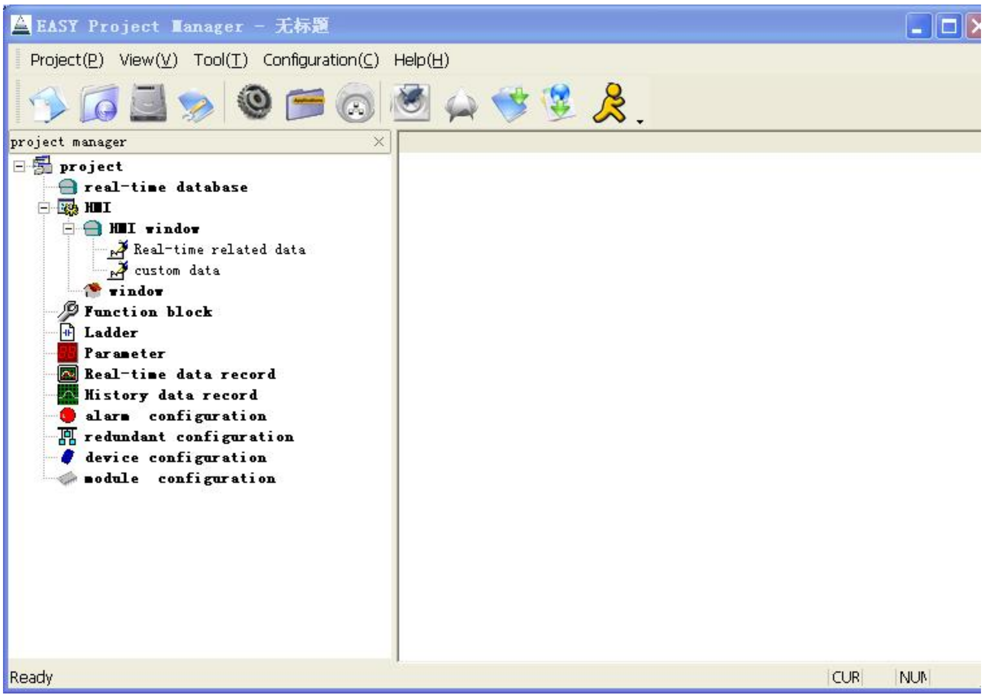
2.2 Creating a Project

This section focuses on describing the simple and easy-to-learn procedure for the beginner to create a project using the EASY software.

2.2.1 Creating a Blank Project

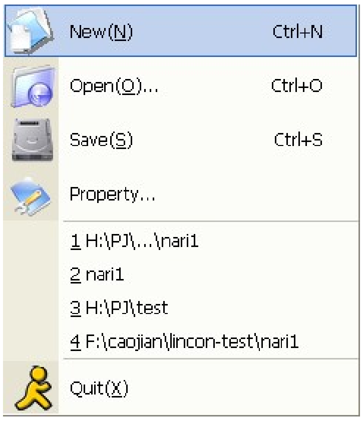
1) In the **Start** menu, go to **Programms** and select **Project Manager**.

   The **EASY Industrial Cotrol Software Development Environment** window will be displayed, as shown in Figuur 2.1.



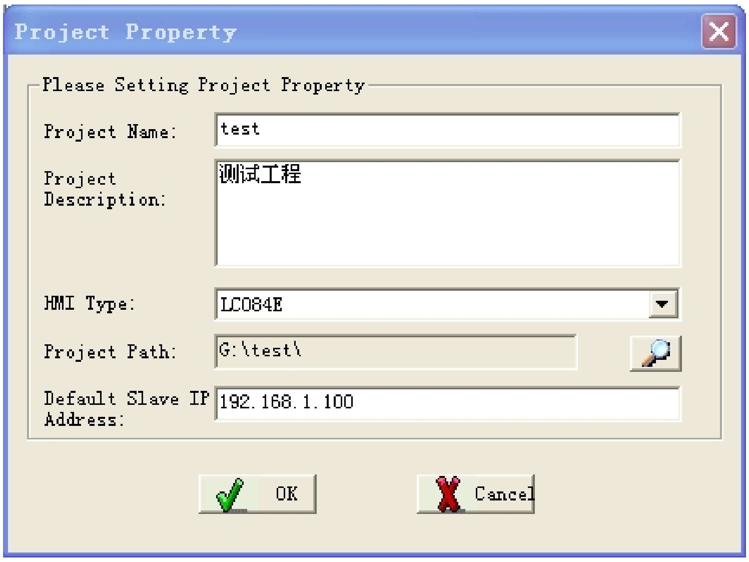
Figuur 2.1

2) Click on the **Project** menu, and then the **Create a Project** submenu, as shown in Figuur 2.2.



Figuur 2.2

The **Project Properties** window as shown in Figuur 2.3 will be displayed:



Project Property

Please Setting Project Property

Project Name: test

Project Description: 测试工程

HMI Type: LC084E

Project Path: G:\test\

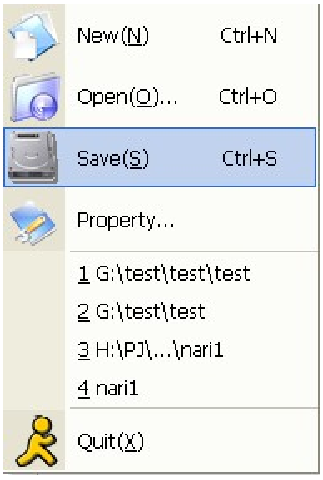
Default Slave IP Address: 192.168.1.100

Figuur 2.3

- **Project Name**: Type in the name of the project in the text box behind **Project Name**. This project name is also the path name for this project.
- **Project Description**: Type in any description information in the text box behind **Project Description**.
- **HMI Model**: Click on the arrow down button to select the HMI model. EASY provides 14 models of three series for your selection. If you are a developer, please select the HMI model accordingly.
- **Project Path**: Click on the 🔍 button, and then select a valid path in the **Select the Path** dialog box.
- **Default Slave IP Adres**: The default IP adres of the HMI provided for user interaction. As a developer, after you type in the IP adres here, you do not need to type in the IP adres again for future project downloading.

After you type in all the above project information, click on the **OK** button, and a new project is created.

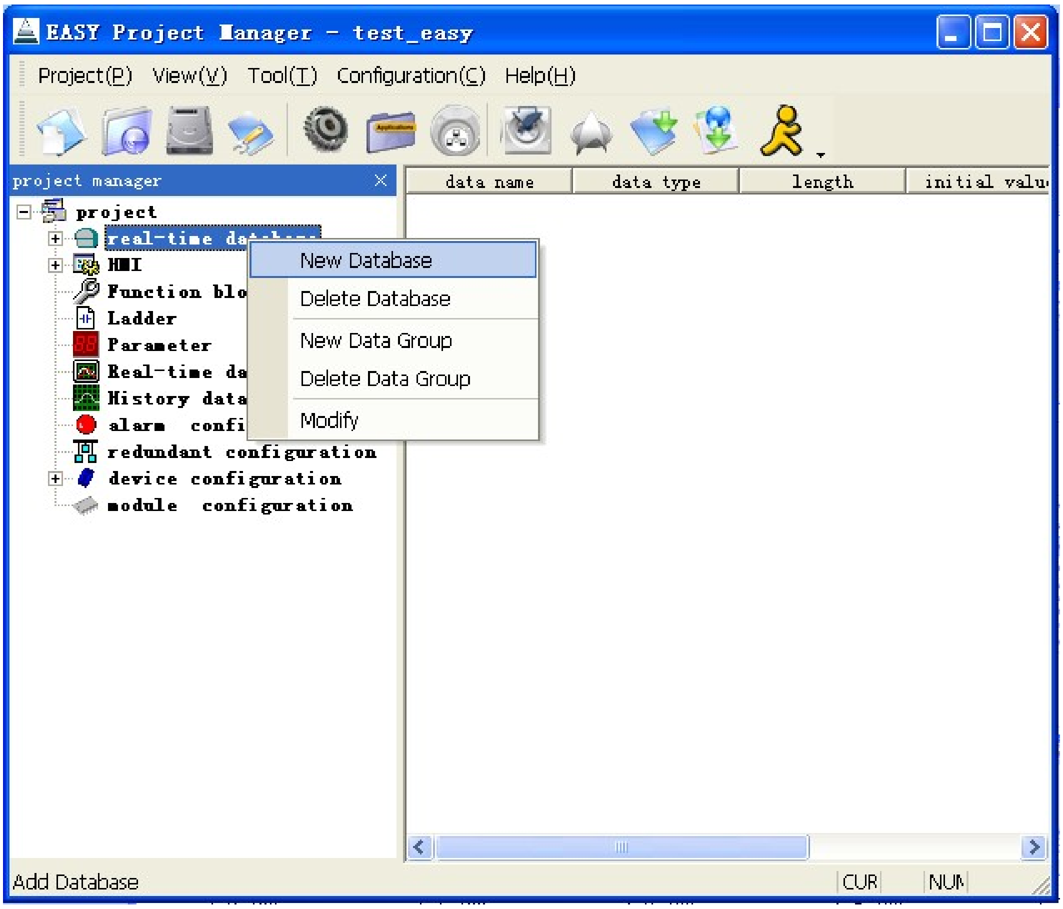
3) To save the project, click on the **Project** menu and then the **Save a Project** submenu, as shown in Figuur 2.4.



Figuur 2.4

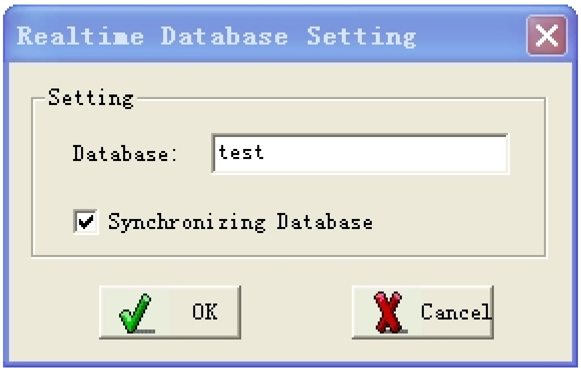
2.2.2 Adding Real-Time Data

1) In the **Project Manager** tree on the left side of the window, right-click on the **Real-Time Database** node and select **Create a Database** from the pop-out menu, as shown in Figuur 2.5.



Figuur 2.5

A dialog box as shwon in Figuur 2.6 will be displayed. Here you can set the database name as **test**. You can check or uncheck the option **Synchronize this database when multiple-device communication occurs**.

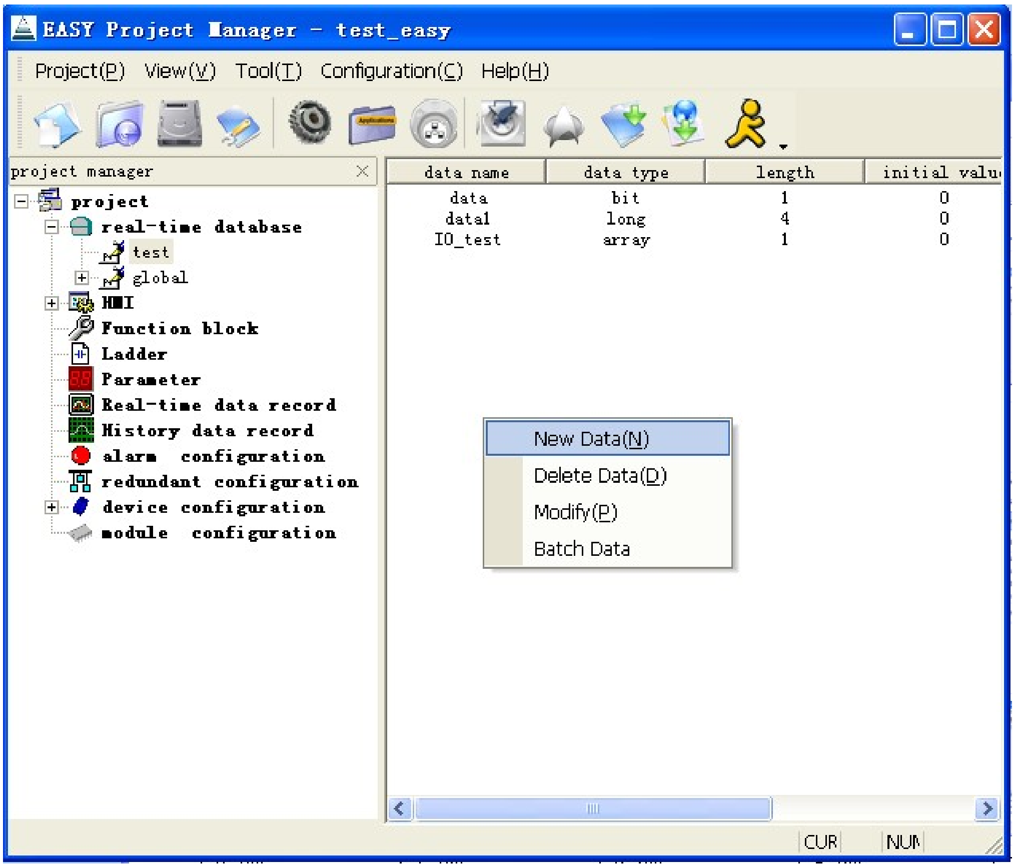


Database: test

☑ Synchronizing Database

Figuur 2.6

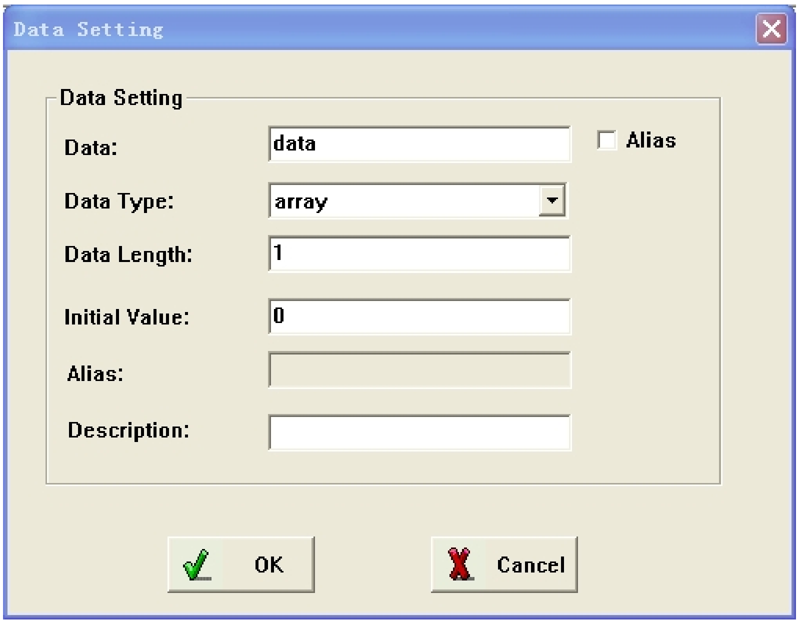
2) Select the new database **test**, right-click anywhere in the right-side pane and select **Add Data** in the pop-out menu, as shown in Figuur 2.7.



data name	data type	length	initial valu
data	bit	1	0
data1	long	4	0
IO_test	array	1	0

Figuur 2.7

A dialog box as shown in Figuur 2.8 will be displayed. Here you can set **Data Name** to **data**, **Data Type** to **Bit**, **Data Length** to **1** (For the data type Bit, the number here refers to the number of bits, and the value range 1-8 means the data is 1-bit to 8-bit binary data; for other data types, the number here refers to the number of bytes), and **Default Value** to **0**.



Data Setting

Data Setting

Data: data   □ Alias

Data Type: array

Data Length: 1

Initial Value: 0

Alias:

Description:

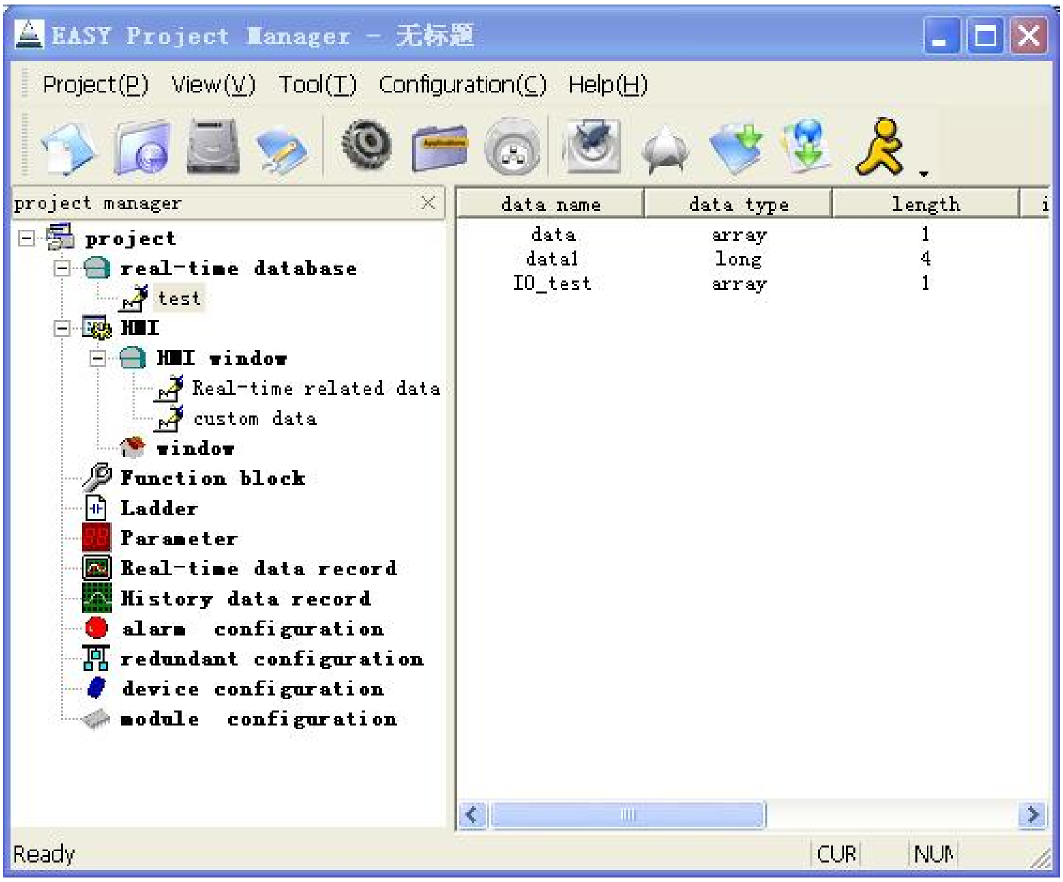
Firgure 2.8

3) Add another two data with the data information listed below:

Data Name: **data2**; **Data Type**: **Long**; **Data Length**: **4**; **Default Value**: **100**.

Data Name: **IO_data1**; **Data Type**: **Bit**; **Data Length**: **1**; **Default Value**: 0.

Figuur 2.9 lists the information of the above three new data.



data name	data type	length	i
data	array	1	
data1	long	4	
IO_test	array	1	

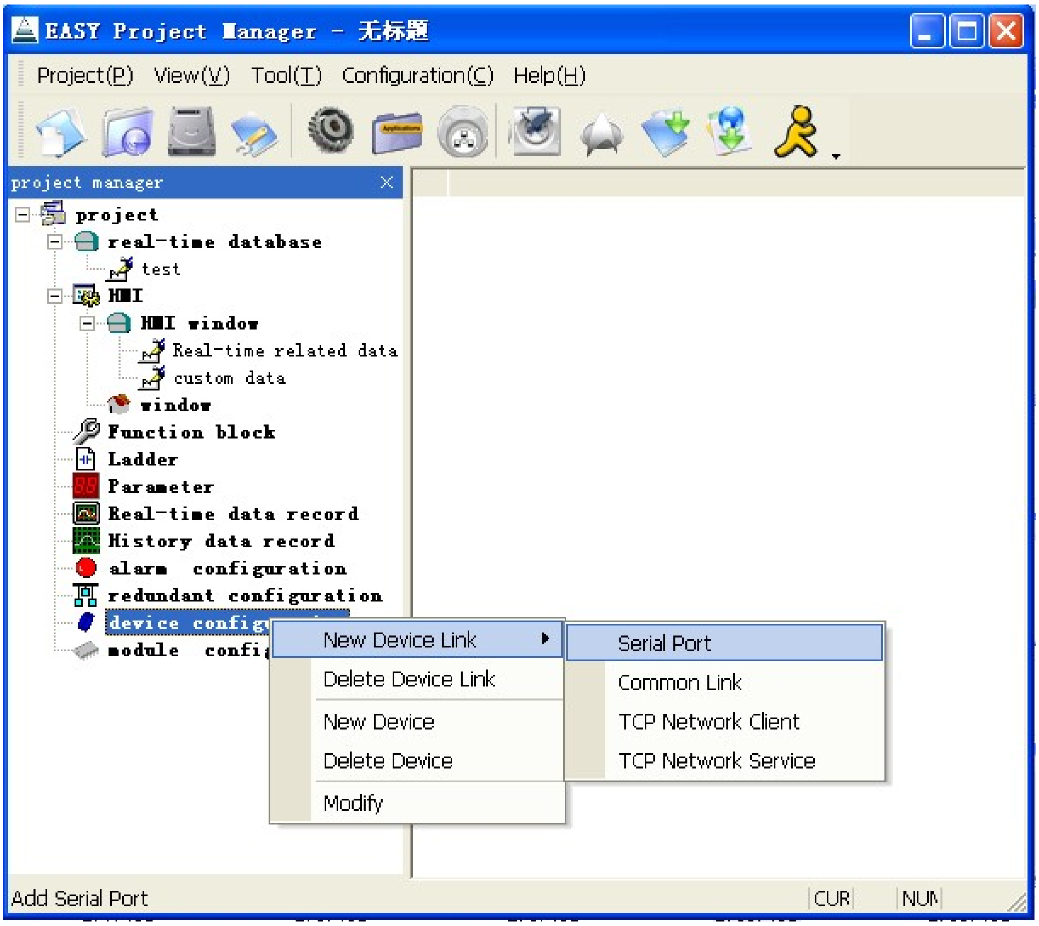
Figuur 2.9

2.2.3 Configuring Site Devices

This part takes the Siemens S7-200 Series PLC for example to explain how to conFiguur site devices.

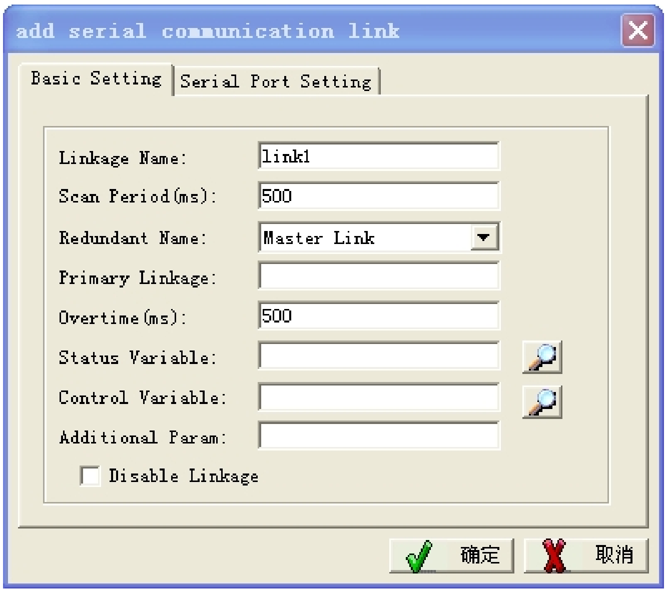
1) Add a serial port communication link.

   In the **Project Manager** tree on the left side of the window, right-click on the **Device Configuration** node, and a pop-out menu will be displayed, as shown in Figuur 2.10.



Figuur 2.10

Select **Add a Communication Link** and then **Serial Port** on the pop-out menu, a dialog box as shown in Figuur 2.11 will be displayed.



Linkage Name: link1

Scan Period(ms): 500

Redundant Name: Master Link

Primary Linkage:

Overtime(ms): 500

Status Variable:

Control Variable:

Additional Param:

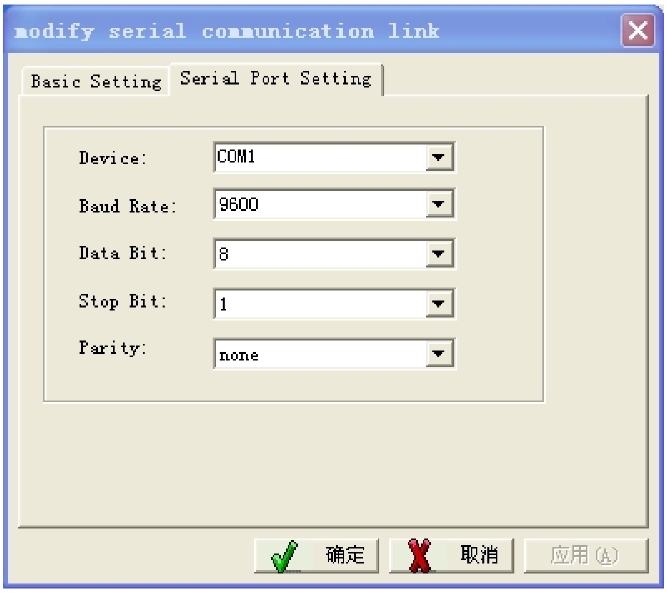
□ Disable Linkage

Figuur 2.11

In the **Basic Link Info** tab, and set the **Link Name**, **Scanning Cycle**, and **Timeout** information, as shown in Figuur 2.11.

In the **Serial Port Info** tab, set the following parameters for the serial port communication, as shown in Figuur 2.12:

Device Name: **COM1**; **Baud Rate**: **9600**; **Data Bits**: **8** bits; **Stop Bit**: **1** bit; **Parity Check**: **Even**.



Device: COM1

Baud Rate: 9600

Data Bit: 8

Stop Bit: 1

Parity: none

Figuur 2.12

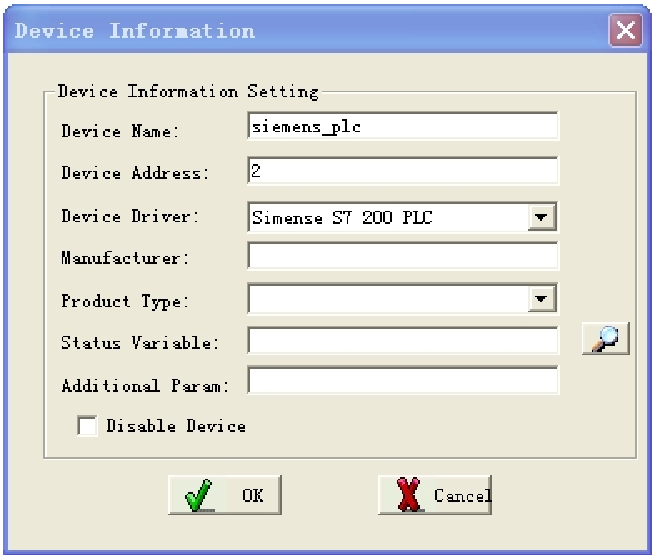
After the above settings are complete, click on **OK** to save the configuration.

2) Add an I/O device.

In the **Project Manager** tree on the left side of the window, right-click the new link

link1, and you'll see the pop-out menu as shown in Figuur 2.10.

Select the **Add a Device** menu, and you'll see the dialog box as shown in Figuur 2.13.



Device Information

Device Information Setting

Device Name: siemens_plc

Device Address: 2

Device Driver: Simense S7 200 PLC

Manufacturer:

Product Type:

Status Variable:

Additional Param:

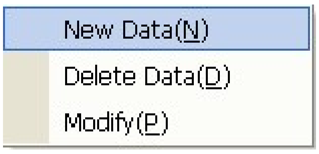
□ Disable Device

Figuur 2.13

In this dialog box, set **Device Name** to **siemens_plc**, **Device Adres** to 2, and **Device Driver** to Siemens S7 200 Series PLC.

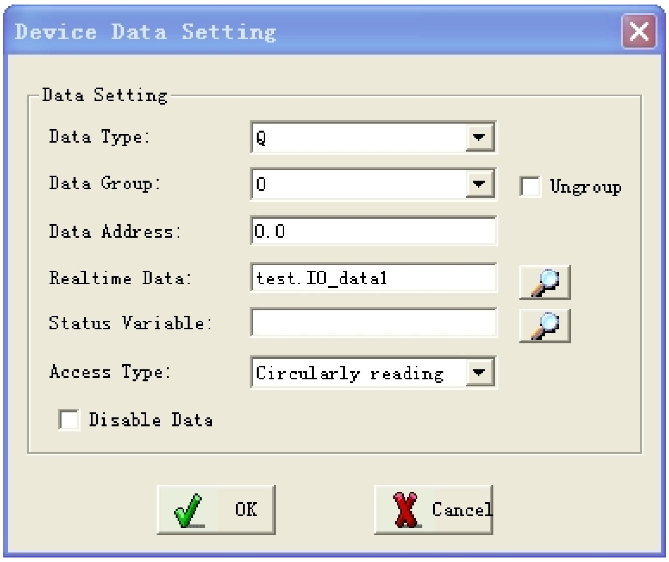
3) Add the data.

In the **Project Manager** tree on the left side of the window, select the new device **siemens_plc**. Right-click in the pane on the right side of the window, and you'll see the pop-out menu as shown in Figuur 2.14.



Figuur 2.14

Select the **Add Data** menu, and a dialog box as shown in Figuur 2.15 will be displayed.



Device Data Setting

Data Setting

Data Type: Q

Data Group: 0 □ Ungroup

Data Address: 0.0

Realtime Data: test.IO_data1

Status Variable:

Access Type: Circularly reading

□ Disable Data

Figuur 2.15

The data configuration for this new I/O device is listed below:

Data Type: **Q**; **Data Group**: **0**; **Data Adres**: **0.0**; **Real-Time Data Name**: **test.IO_data1**; **Access Mode**: **Repeat Read and Single Write**.

In this way, the digital output register **Q0.0** and the variable **test.IO_data1** in the real-time database are mapped to each other. Whatever changes on the value of **Q0.0** will be relected on the **test.IO_data1**; similarly, when the value of **test.IO_data1** is modified, the status of **Q0.0** will be affected as well.

After you set all the above device data, click on **OK** to save the configuration.

2.2.4 Creating the Configuration Interface

In the interface development system, you can create as many interfaces as you want for each project, and generate static or dynamic interrelated graphic objects for each interface. The various types of graphic objects which compose the interfaces are actually provided by the Interface Editor of the EASY system.

EASY provides two types of controls: configuration interface controls and Windows controls. The configuration interface controls are subcategorized into basic graphic objects, such as rectangles (including incremental rectangles), lines, ovals (including circles), bitmaps, or text, and complicated graphic objects, such as buttons, trend curve windows, or alarm windows. The Windows controls include checkboxes, historical lists, and drop-down lists.

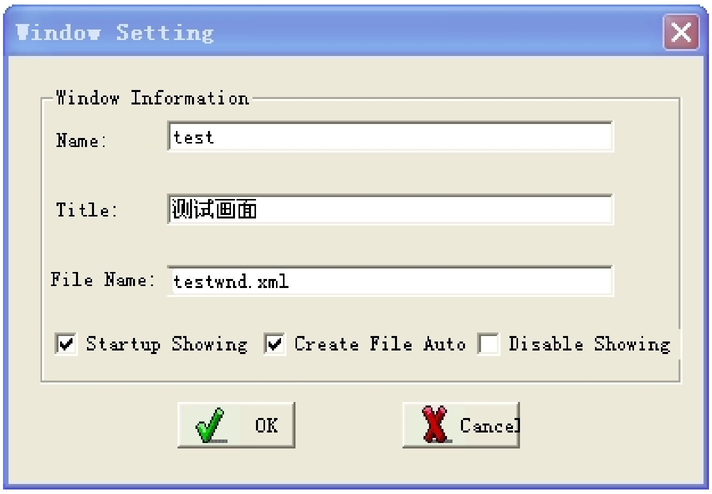
Besides, EASY provides tools for you to do the following operations on graphic objects:

- Dragging around inside the window
- Zooming in or out
- Reshaping
- Copying
- Deleting
- Aligning
- Both keyboard and mouse drawing
- Adjusting the color
- Changing the line type
- Changing the filling properties

The Interface Editor makes full use of the object-oriented technology to help you easily set up graphic interfaces. You can select the graphic objects provided by the system to generate various interfaces, just like you are building with blocks. In addition, EASY allows you to copy graphic objects among interfaces, so that you can easily use the developed results.

To create an interface, please follow the procedure below:

1) In the Project Manager window, select the **HMI** node and then the **Interface** child node. Right-click inside the list pane on the right side and select **Add an Interface**. In the **Interface Setting** dialog box as shown in Figuur 2.16, set **Interface Name** to **test**, **Title** to **Test Interface**, **File Name** to **testwnd**, and check the options **Display at the startup of the program** and **Automatically create files**.



Name: test

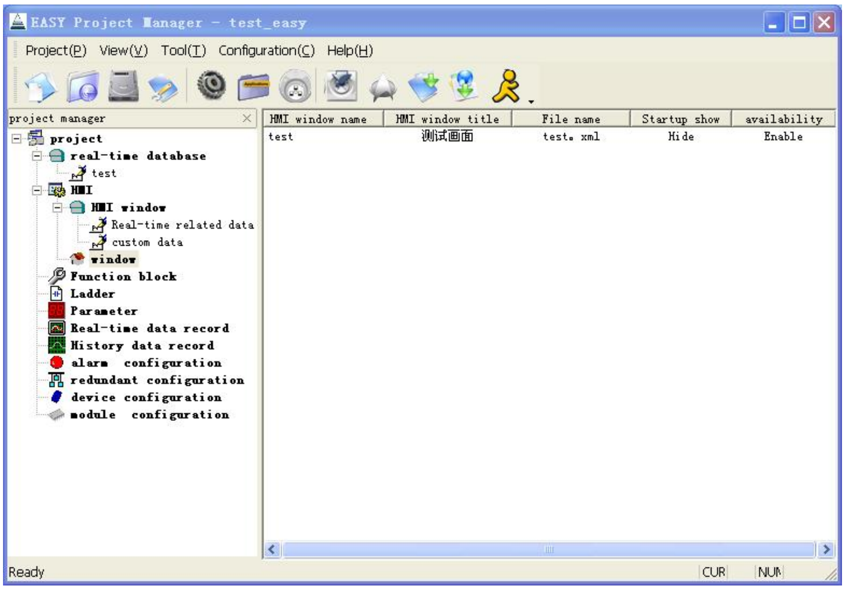
Title: 测试画面

File Name: testwnd.xml

☑ Startup Showing ☑ Create File Auto □ Disable Showing

Figuur 2.16

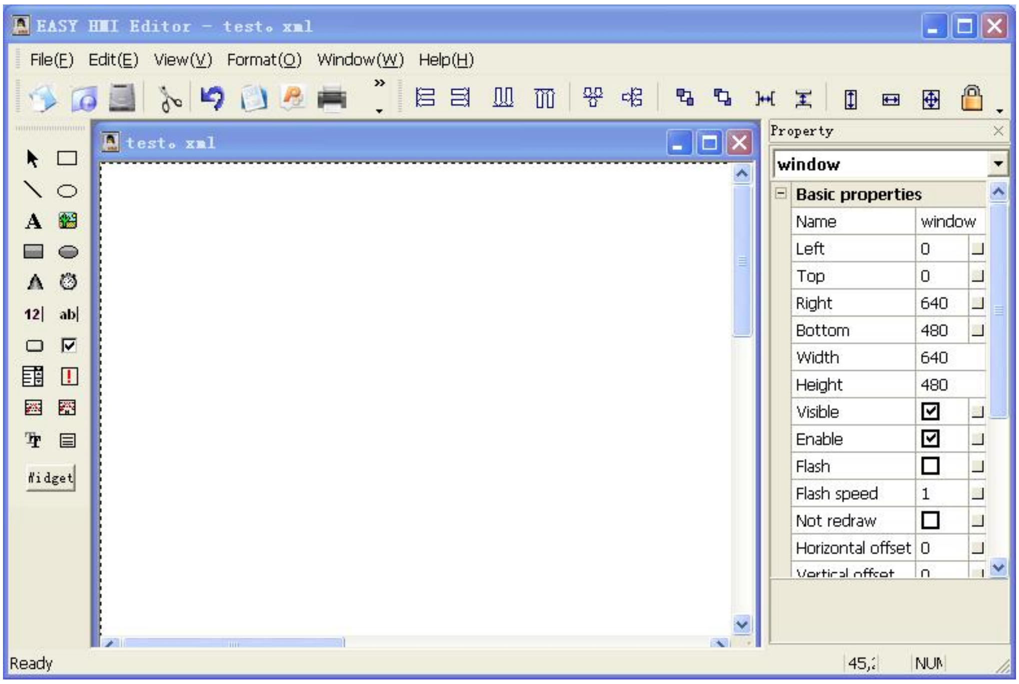
Click on **OK**, and an interface file will be generated, as shown in Figuur 2.17.



HMI window name	HMI window title	File name	Startup show	availability
test	测试画面	test. xml	Hide	Enable

Figuur 2.17

2) Double-click on the newly created **test** interface file on the right side, and the Interface Editor software window will be displayed, as shown in Figuur 2.18.


EASY HMI Editor - test. xml
File(F) Edit(E) View(V) Format(O) Window(W) Help(H)
test. xml
Widget
Property
window
Ready
45,
NUM

Basic properties

Name	window
Left	0
Top	0
Right	640
Bottom	480
Width	640
Height	480
Visible	☑
Enable	☑
Flash	□
Flash speed	1
Not redraw	□
Horizontal offset	0
Vertical offset	0

Figuur 2.18

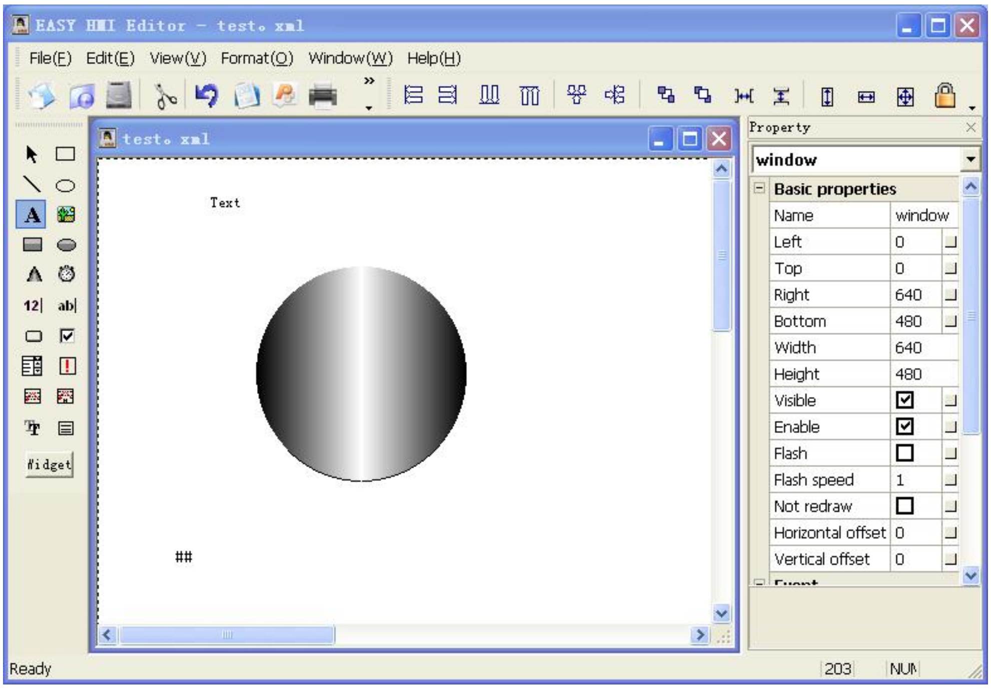
3) In the tools set on the left side of the window, click on the dynamic text tool A and drag across in the editing pane to create a text graphic component. The default text set by the system is **text**.

4) Select this text graphic component, and set the text properties in the property list on the right side of the window, such as **Text Color**, **Text Contents**, **Text Font**, etc. For examble, you can change **Text Contents** to **0**.

5) Similarly, click on the incremental oval tool from the tools set on the left to create an incremental oval.

6) Or click on the data input tool to create a data graphic component.

7) Select **File** > **Save** to save this interface, as shown in Figuur 2.19.


EASY HMI Editor - test.xml
File(F) Edit(E) View(V) Format(O) Window(W) Help(H)
test.xml
Text
##
Widget
Property
Ready
203
NUM

window

Basic properties

Name	window
Left	0
Top	0
Right	640
Bottom	480
Width	640
Height	480
Visible	☑
Enable	☑
Flash	□
Flash speed	1
Not redraw	□
Horizontal offset	0
Vertical offset	0

Figuur 2.19

2.2.5 Setting Dynamic Properties for the Interface

Defining animation links is to establishing connection between graphic ojbects in the interface and data variables in the database. On one hand, any changes of the variables will be vividly reflected by the animation effects of the graphic objects. On the other hand, the software developer can change properties of the graphic objects so as to change values of the data variables.

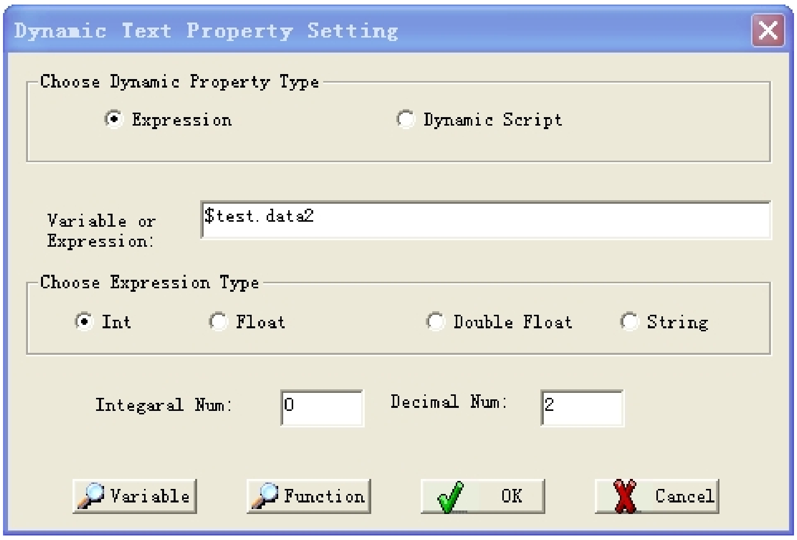
EASY Interface Editor provides 21 various types of animation links, which are categorized as follows:

Property Changes	Changes in line properties, filling properties, and text color
Location and Size Changes	Filling, zooming, and horizental and vertical positioning
Value Output	Digitals, analogs, and customized expressions
Special Characteristics	Flashing and visibility
Command Languages	For pressing down and releasing a button

Multiple animation links can be defined for each graphic object to form complicated animation effects, so as to meet various display needs during the site operation.

1) Select the **text** graphic component created in section 2.2.4. In the **Property List**

on the right side of the window, click on the small rectangle button □ on the right most side of the **Text** property, and the **Dynamic Text Property Settings** window as shown in Figuur 2.20 will be displayed. Click on the **Variable** button, and select the corresponding variable in the HMI database as shown in Figuur 2.21.



Dynamic Text Property Setting

Choose Dynamic Property Type

● Expression ○ Dynamic Script

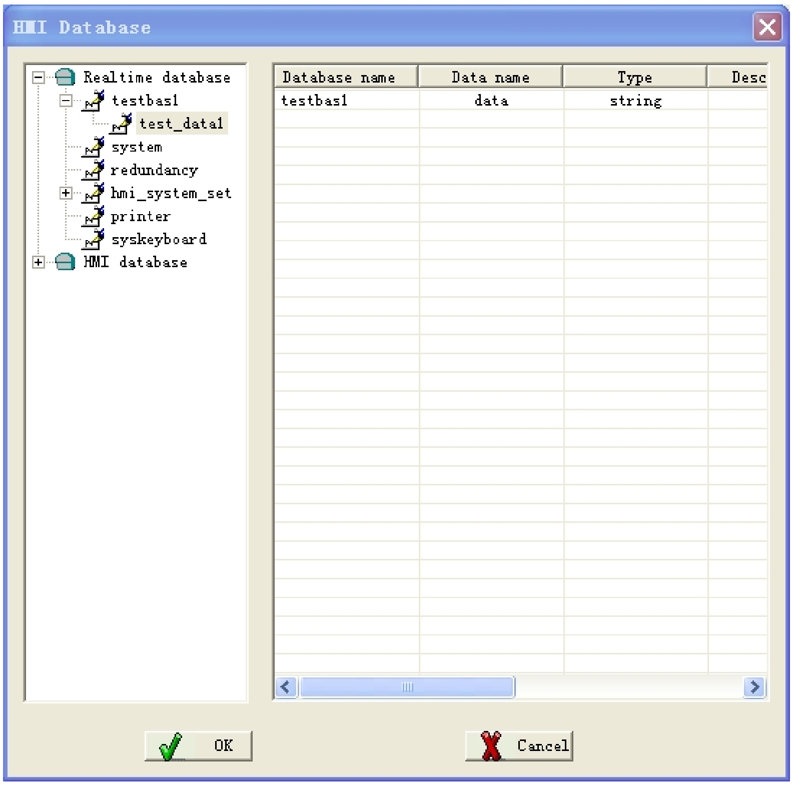
Variable or Expression: $test.data2

Choose Expression Type

● Int ○ Float ○ Double Float ○ String

Integaral Num: 0 Decimal Num: 2

Figuur 2.20



HMI Database

- Realtime database
  - testbas1
    - test_data1
  - system
  - redundancy
  - hmi_system_set
  - printer
  - syskeyboard
- HMI database

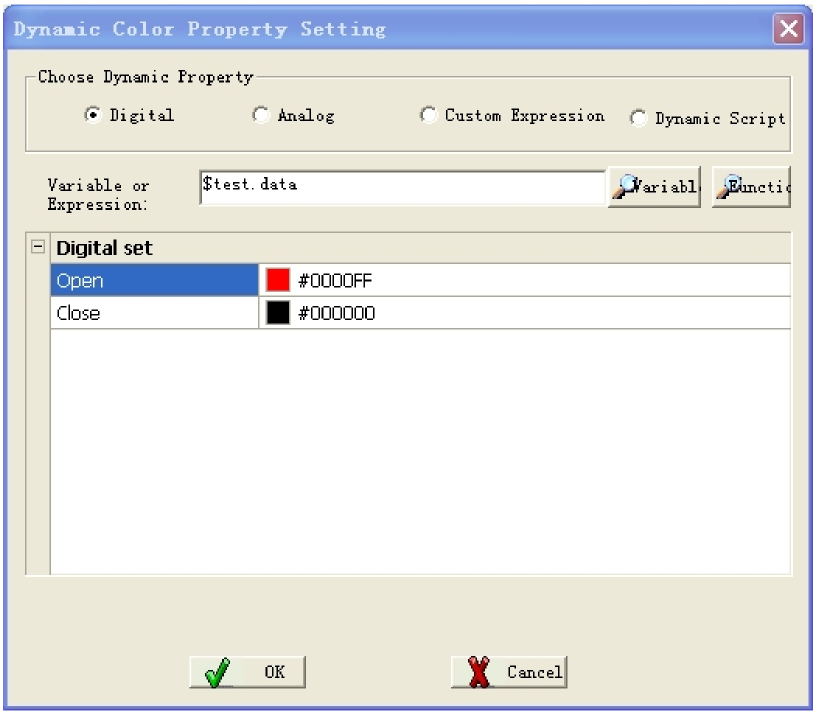
Database name	Data name	Type	Desc
testbas1	data	string	

Figuur 2.21

Click on **OK** to save the property settings.

2) Select the oval graphic component created in section 2.2.4. In the **Property List** on the right side of the window, click on the small button on the right most of the **Start**

Color property, and the **Dynamic Color Property Settings** window as shown in Figuur 2.22 will be displayed. Set **Dynamic Property Type** to **Digital**, and set the color for both **On** and **Off** under **Digital Settings**. Click on the **Variable** button and select the digital variable from the HMI database as shown in Figuur 2.23.



Dynamic Color Property Setting

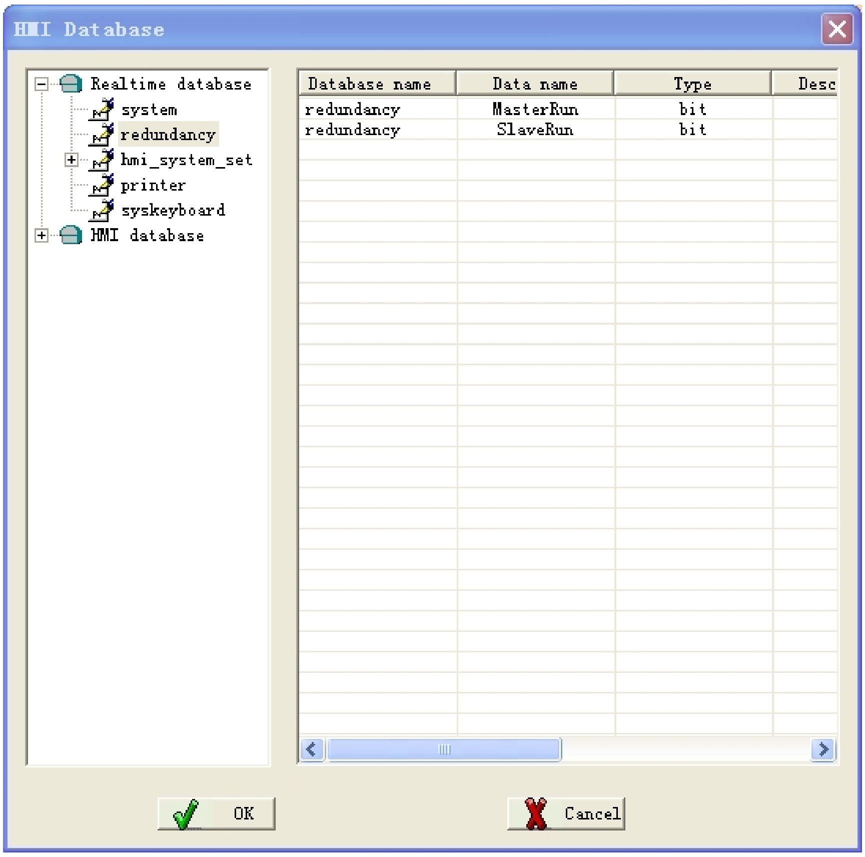
Choose Dynamic Property

● Digital ○ Analog ○ Custom Expression ○ Dynamic Script

Variable or Expression: $test.data

Digital set	
Open	🟥 #0000FF
Close	■ #000000

Figuur 2.22



HMI Database

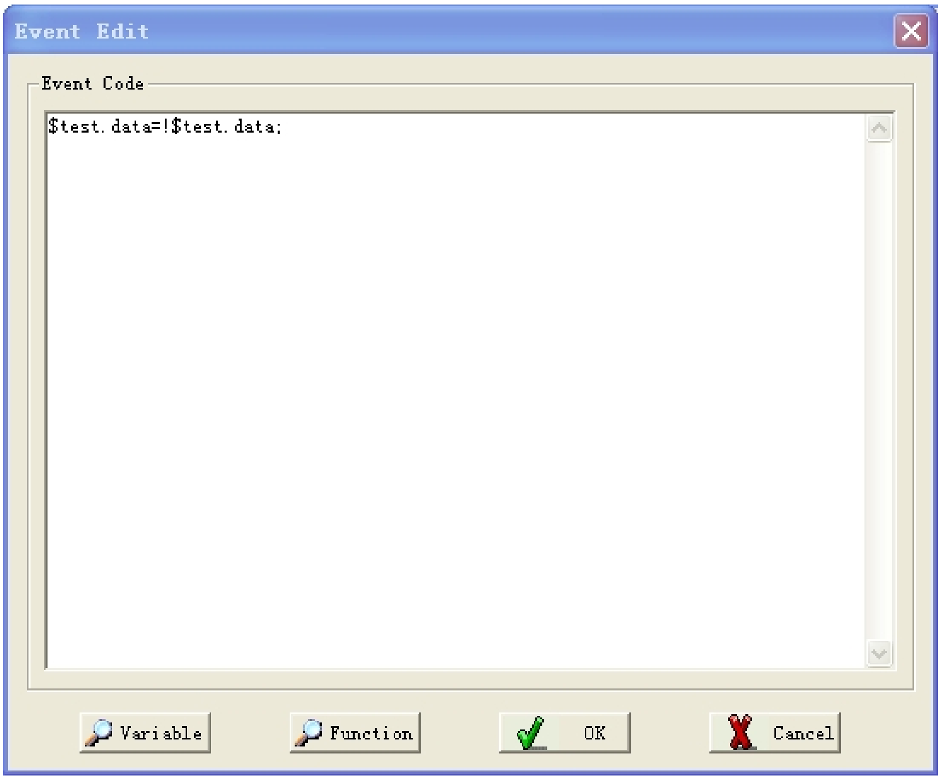
- Realtime database
  - system
  - redundancy
  - hmi_system_set
  - printer
  - syskeyboard
- HMI database

Database name	Data name	Type	Desc
redundancy	MasterRun	bit	
redundancy	SlaveRun	bit	

Figuur 2.23

Click on File and then Save to save the property settings.

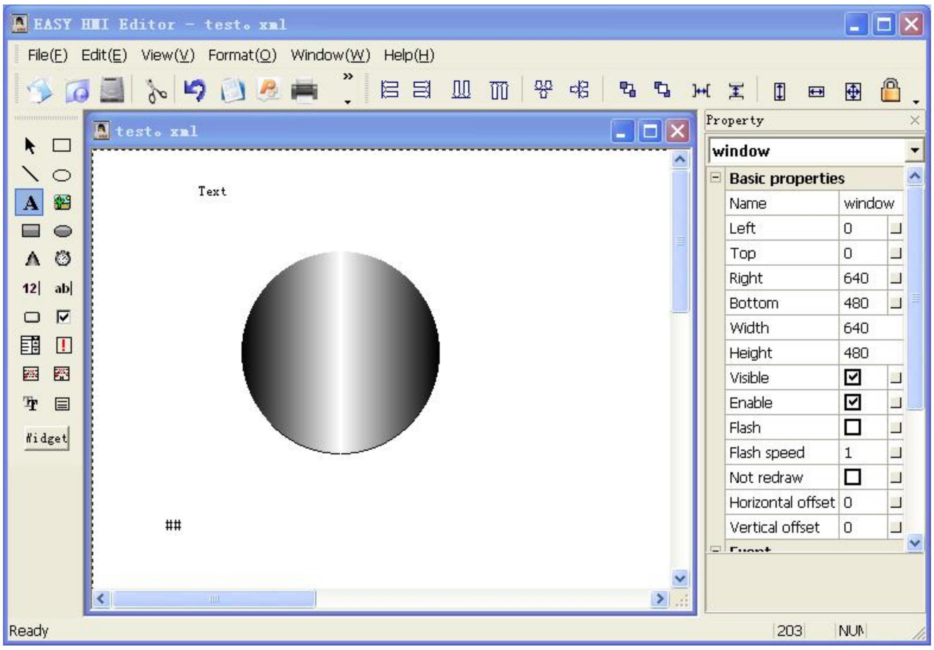
3) Select the oval graphic component created in section 2.2.4. In the Property List on the right side of the window, click on the **Press** event editing box, and the Event Editing window as shown in Figuur 2.24 will be displayed. Type in the event code **$test.data=!$test.data;** and click on **OK**. (**Caution**: According to the syntax of the C language, a semi-colon is compulsory at the end of the command line.)



```
$test.data=!$test.data;
```

Figuur 2.24

4) Select the data input graphic component created in section 2.2.4. In the **Property List** on the right side of the window, click in the **Variable Name** property editing box, and select the corresponding variable **test.IO_data1** from the HMI database, as shown in Figuur 2.25.


EASY HMI Editor - test.xml
File(F) Edit(E) View(V) Format(O) Window(W) Help(H)
test.xml
Text
##
Widget
Property
window
Basic properties
Event
Ready
203
NUM

Basic properties	
Name	window
Left	0
Top	0
Right	640
Bottom	480
Width	640
Height	480
Visible	☑
Enable	☑
Flash	□
Flash speed	1
Not redraw	□
Horizontal offset	0
Vertical offset	0

Figuur 2.25

5) Click on **File** and then **Save** to save the event code.

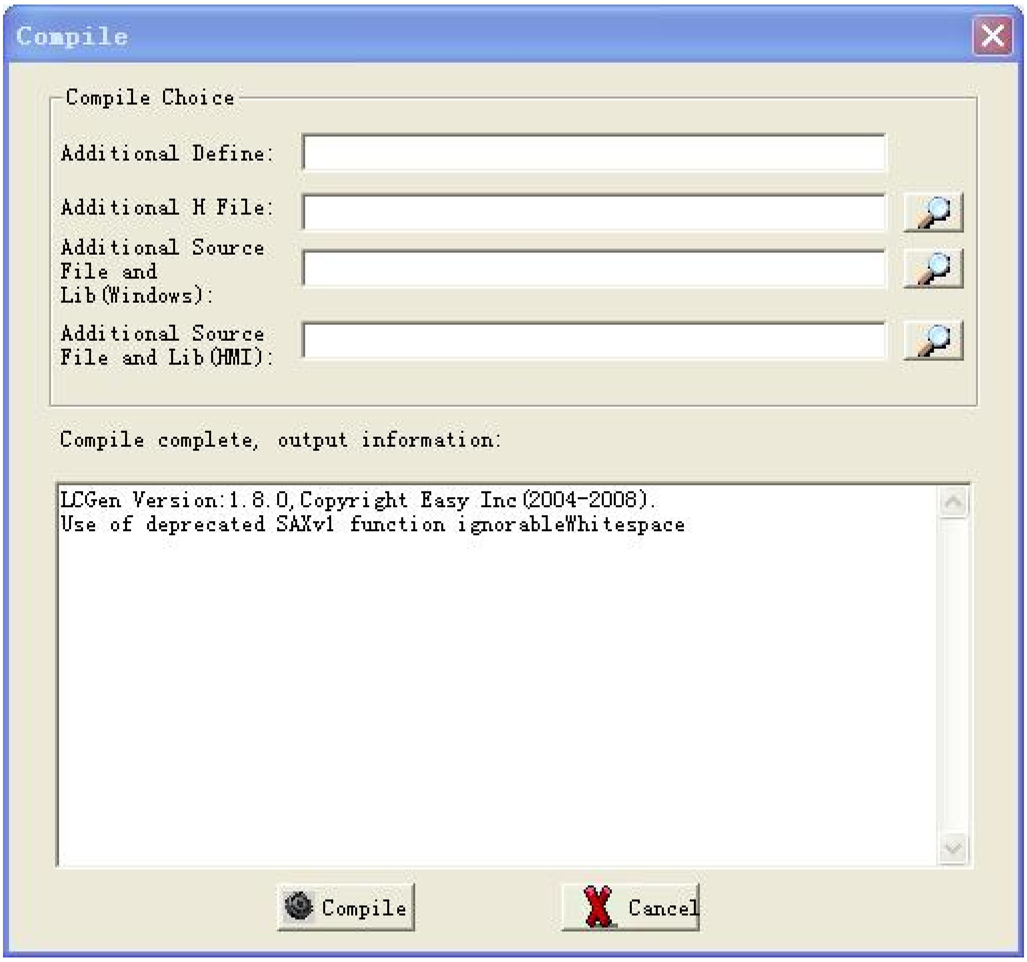
6) Exit the Interface Editor software, and it will take you back to the main **Project Manager** interface.

2.2.6 Compiling a Project and Project Simulation

1) Select **Project** and then **Save a Project** to save the conFiguurd project.

2) Select **Tools** and then **Compile a Project**, and you will see a window as showin in Figuur 2.26. Click on the **Start Compiling** button.

3) Click on the **Exit** button after the compiling is complete.

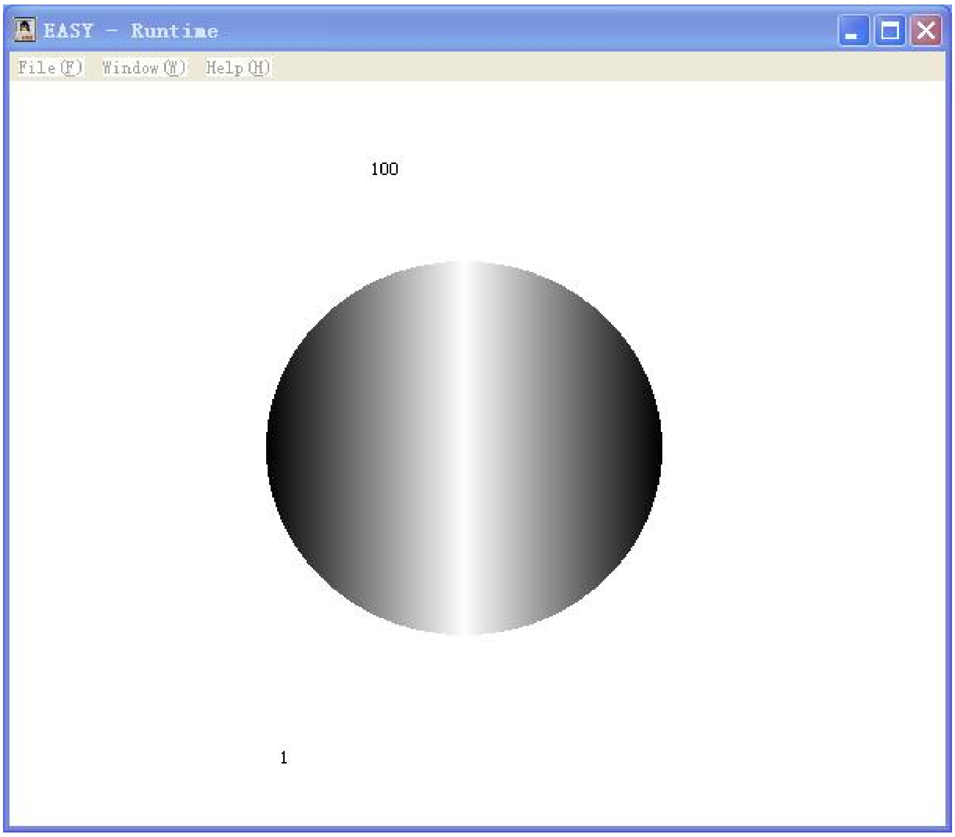


Compile
Compile Choice
Additional Define:
Additional H File:
Additional Source File and Lib(Windows):
Additional Source File and Lib(HMI):
Compile complete, output information:
Compile
Cancel

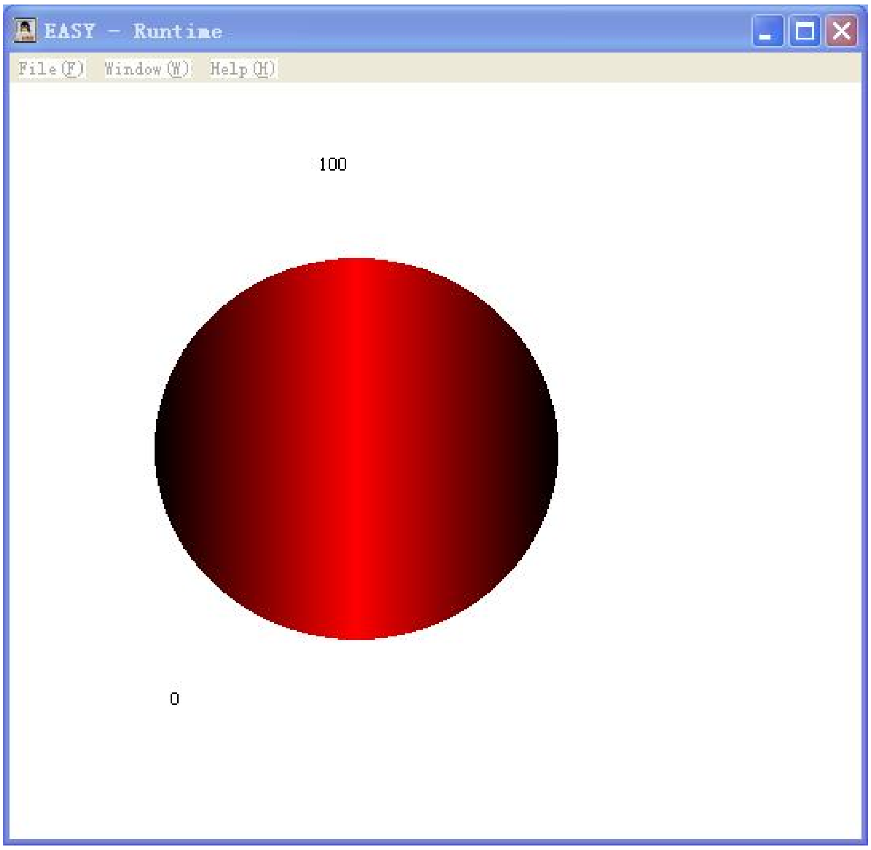
```
LCGen Version:1.8.0,Copyright Easy Inc(2004-2008).
Use of deprecated SAXv1 function ignorableWhitespace
```

Figuur 2.26

4) Select Tools and then Offline Simulation, and the configuration interface as shown in Figuur 2.27 will be displayed. In this interface, the text **100** reflects the data value of **data2**. Click on the oval graphic component, and you can see the color changes. Click on the **Data Input** box, type in the data **0** or **1**, and you can see the graphic component on the PLC closes and then opens as the value of the digital output point Q0.0 changes from 0 to 1.



Figuur 2.27



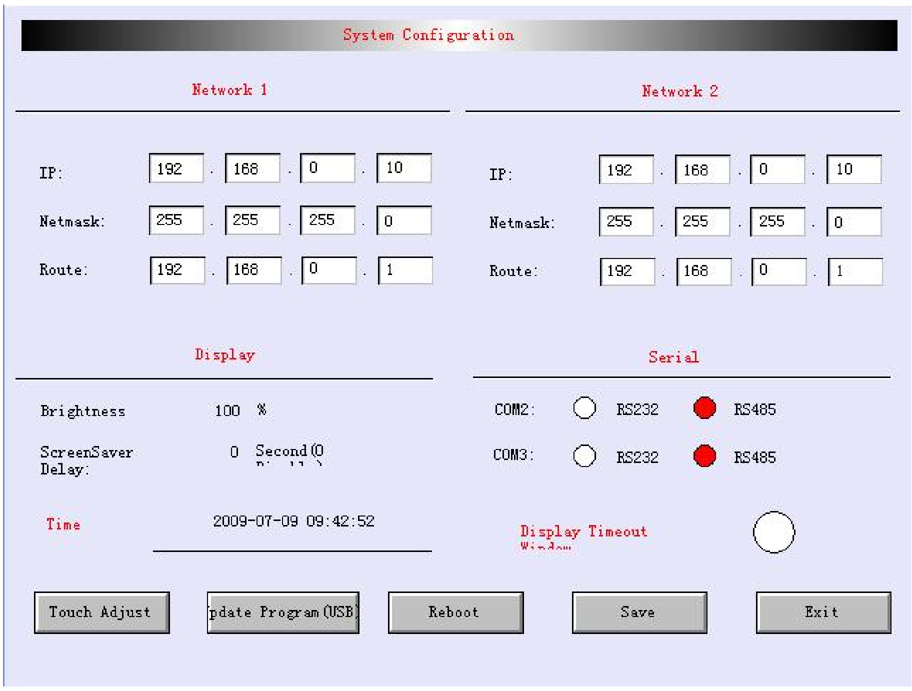
Figuur 2.28

2.2.7 Downloading een Project

EASY support project downloading via Ethernet. Voordat u begint met downloaden van een project, dient u er zeker van te zijn dat de PC en de HMI communicate over het netwerk (LAN) goed verloopt.

1) Het conFiguurren van het IP adres van de HMI.

Klik 1 maal op de linker boven hoek van de HMI vervolgens de rechterbovenhoek en gevolgt door de linker beneden hoek, en de **System Configuratie** window zoals u hier onder ziet in Figuur 2.29 word getoond. Zet het IP adres van het netwerk card based on the the network port used by the HMI. Het systeem default IP adres van de HMI is 192.168.1.10. (linkerzijde van het display onder Network 1). Network 2 word niet gebruikt!



System Configuration

Network 1

IP: 192 . 168 . 0 . 10
Netmask: 255 . 255 . 255 . 0
Route: 192 . 168 . 0 . 1

Network 2

IP: 192 . 168 . 0 . 10
Netmask: 255 . 255 . 255 . 0
Route: 192 . 168 . 0 . 1

Display

Brightness 100 %
ScreenSaver Delay: 0 Second(0
Time 2009-07-09 09:42:52

Serial

COM2: ○ RS232 ● RS485
COM3: ○ RS232 ● RS485
Display Timeout ○

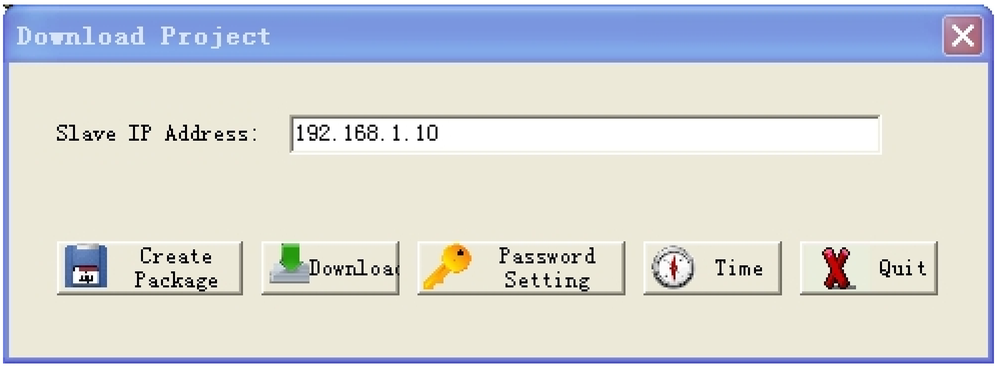
Figuur 2.29

2) De computer en de HMI dienen zich in het zelfde netwerk IP reeks te bevinden, in dit geval de IP reeks 192.168.0.xxx. Verzeker u zelf er eerst van dat u de HMI kan pingen middels het HMI ping IP adres vanaf de pc die u gebruikt. In dit geval: PING 192.168.0.10

3) Selecteer **Tools** en dan **Download a Project**, and you will see the **Download a Project** window as shown in Figuur 2.30. Type in the IP adres of the HMI in the text box behind **Slave IP Adres**.

4) Click on the **Generate Downloading Package** button to generate a downloading package.

5) Click on the **Download** button to start downloading the project.



Download Project
Slave IP Address:
192.168.1.10
Create Package
Download
Password Setting
Time
Quit

Figuur 2.30

Chapter 3 Real-Time Database

3.1 Overview

The real-time database is the core of EASY. During the system operation, the actual manufacturing status of various sites are expected to be reflected vividly on the screen through various animation effects. Meanwhile, the commands entered into the PC by the engineer are expected to be delivered to the sites speedily. The ladder diagram, the function control program, and the HMI all exchange data through the real-time database. In a word, the real-time database works as the bridge between the supervisory computer and the PLC.

Besides, the real-time database also functions for data communication with external I/O devices. Different from the historical database, all the data in the real-time database are real-time data, namely, current values. The real-time database stores current values of variables, including system variables and user-defined variables.

3.2 Basic Concepts

- Real-Time Database

The real-time database is a set of real-time data. Multiple real-time databases are allowed in EASY. The main advantage of this is that the same data can be kept in various real-time databases. In other words, the various real-time databases can function together to increase the system efficiency.

- Data Group

Various data groups can be defined in a real-time database. The data of the same category can be grouped together into one data group for better management and more efficient inquiry. Data and sub-data groups can be defined in data groups to establish hierarchical data structure.

- Real-Time Data

The real-time data defines all the data used by the HMI. The real-time data can belong to a real-time database or a data group. When referencing, the real-time data is referenced as “Name of the real-time database.Name of the real-time data”. Therefore, the real-time data of the same database cannot share the same name (even though they are of different data groups).

However, the real-time data can use alias. EASY supports data access in the real-time database by alias.

3.3 Data Types

Types of the data in the real-time database are similar to those of variables used in the programming language – the C language. The data types are defined based on the syntax of the C language, and thus meet the basic programming requirements and needs. Based on this similarity, the real-time data can be also called variables.

There are the following types of data in the real-time database:

- **bit**

  The bit data can be used for digitals. It usually has one bit, and values 0 and 1.

  EASY allows the bit data which has more than 1 digit; the data in EASY can be 1 to 7 digits.

  Note: When adding the bit data, the data length is the number of digits.

- **char**

  Similar to the signed char variable in the C language. This type of data is the signed single-byte data.

- **uchar**

  Similar to the unsigned char variable in the C language. This type of data is the unsigned single-byte data.

- **short**

  Similar to the short variable in the C language. This type of data is signed double-byte data.

- **ushort**

  Similar to the unsigned short variable in the C language. This type of data is unsigned double-byte data.

- **long**

  Similar to the long variable in the C language. This type of data is signed double-byte data.

- **ulong**

  Similar to the unsigned long variable in the C language. This type of data is unsigned double-byte data.

- **float**

  Similar to the float variables in the C language. This type of data is single-precision floating point data.

- **double**

  Similar to the double variable in the C language. This type of data is double-precision floating point data.

- **string**

  Similar to the character array in the C language. This type of data are character strings with specific meaning. You can customize the data length. However, similar to the C language, all string variables end with **\0**. Therefore, the actual data length should be the data length minus 1.

- **array**

  Similar to the array variable in the C language. The data length is the number of data bytes. At present, this type of data can only be used for the script for interface configuration and exteral C language programs.

3.4 Data Definitions

3.4.1 Naming Conventions

The naming conventions for the database, data array, and data are as follows:

1) Same naming conventions for the identifier of the C language, but with the Chinese language supported. That is, the name starts with a letter, the underscore, or a Chinese character, and then is followed by letters, numbers, the underscore or Chinese characters.

2) The database name, data array name, and data name are all case sensitive.

3) Names for data arrays and data of the same database must be different.

4) The following database names are reserved for the system internal use only, and thus are not available for user selection:

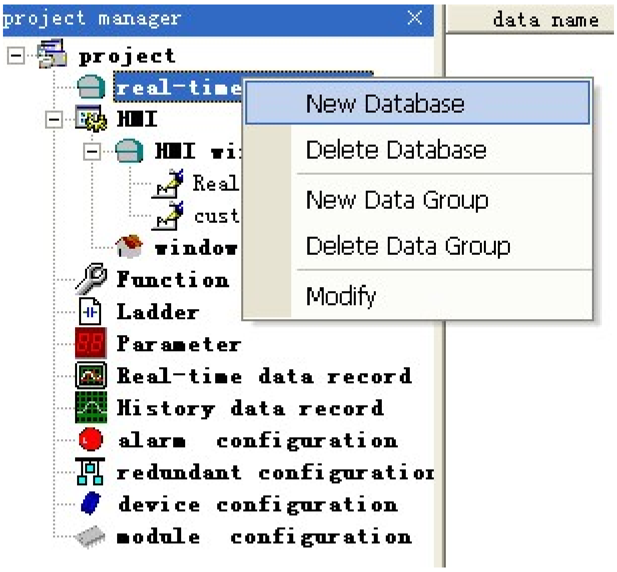
system, **redundancy**, **hmi_system_set**, **printer**, **syskeyboard**, **redund_vars**, and all database names starting with **EASY**.

3.4.2 Definition of the Database

3.4.2.1 Creating a Database

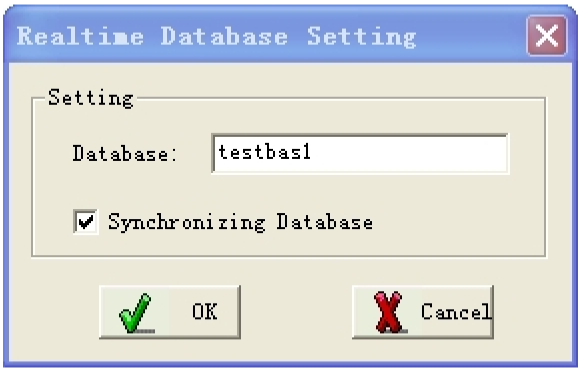
To create a database, do as follows:

1) In the **Project Manager** window, right-click on **Real-Time Database**, and you will see a right-click menu as shown in Figuur 3.1.



Figuur 3.1

2) Select **Add a Database**, and you will see a dialog box as shown in Figuur 3.2.



Database: testbas1

☑ Synchronizing Database

Figuur 3.2

3) Type in a data as the database name; for example, **testbase1**.

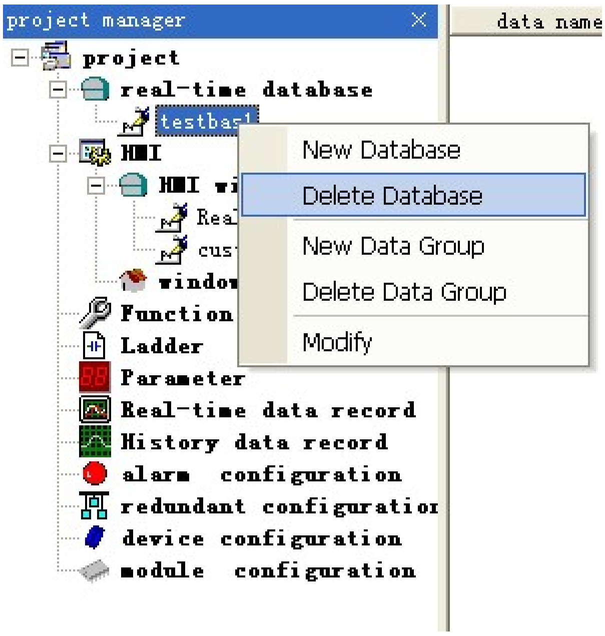
4) Click on **OK**.

   And a database named **testbase1** is created. Make sure to check the **Synchronize this database when multiple-device communication occurs** option, so that the data in the database **testbase1** will be synchronized when redundant communication occurs between multiple HMIs.

3.4.2.2 Deleting a Database

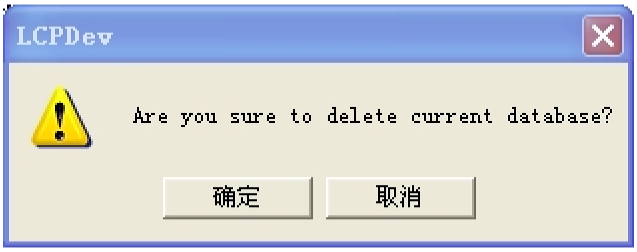
To delete a defined database, do as follows:

1) On the left side of the **Project Manager** window, right-click on a database you want to delete, as shown in Figuur 3.3.



Figuur 3.3

2) Select **Delete a Database**, and you will see a dialog box as shown in Figuur 3.4.



Are you sure to delete current database?

Figuur 3.4

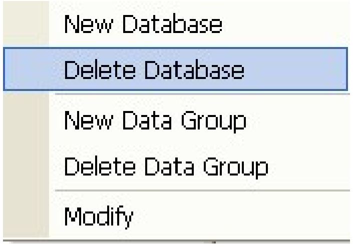
3) Click on **OK**, and the selected database will be deleted.

3.4.3 Definition of the Data Group

3.4.3.1 Creating a Data Group

After creating a database, you can categorize the data into different data groups. Do as follows:

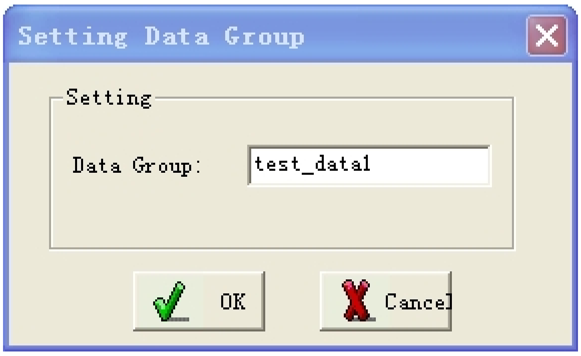
1) Right-click on the created database called **database1**, as shown in Figuur 3.5.



Figuur 3.5

2) Select **Create a Data Group**, and you will see a dialog box as shown in Figuur

3.6.


Setting Data Group

Setting

Data Group: test_data1

Figuur 3.6

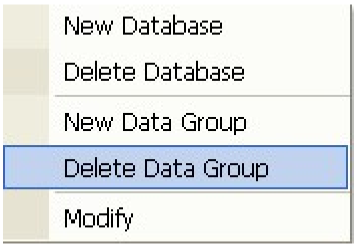
3) Type in a name for the data group **test_data1**, and click on **OK**.

A data group called **test_data1** is created.

3.4.3.2 Deleting a Data Group

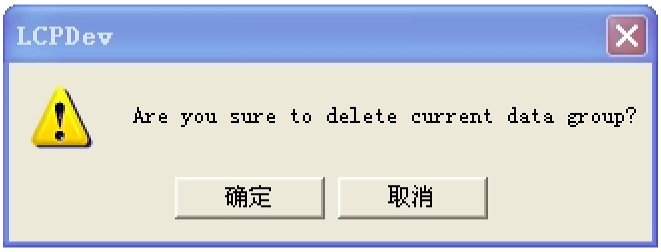
To delete a data group, do as follows:

1) Right-click on a data group, for example, **test_data1**, as shown in Figuur 3.7.



Figuur 3.7

2) Select Delete a Data Group, and you will see a dialog box as shown in Figuur 3.8.


LCPDev

Are you sure to delete current data group?

Figuur 3.8

3) Click on OK, and then the data group **test_data1** is deleted.

3.4.4 Definition of the Real-Time Data Variable

3.4.4.1 Adding Real-Time Data

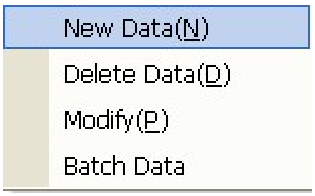
To add data into a real-time database, do as follows:

On the left side of the **Project Manager** window, select the database created earlier **database1**, and then right-click in the list pane on the right side of the window.

To add data into a data group (The data in the data group and that in the database must not share the same name.), do as follows:

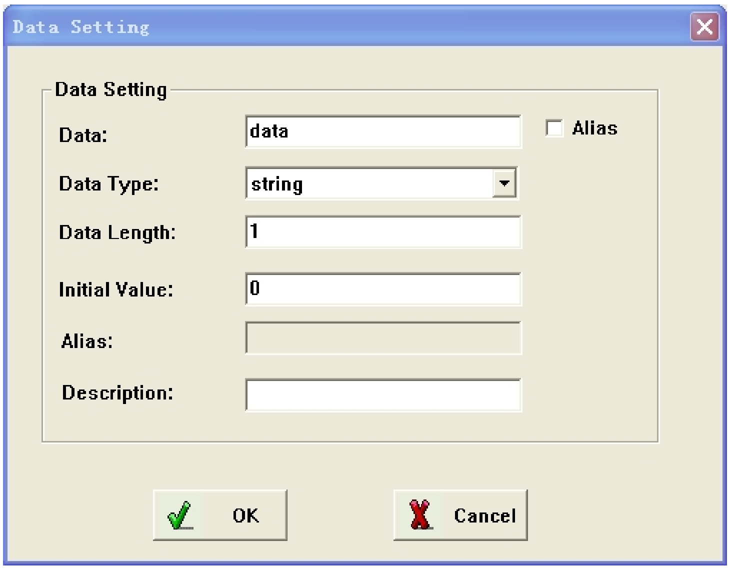
1) Select the database **test_data1** from the left side of the Project Manager window.

2) Right-click in the list pane on the right side of the window, as shown in Figuur 3.9.



Figuur 3.9

3) Select **Add Data**, and you will see a dialog box as shown in Figuur 3.10.



Data Setting

Data Setting

Data: data

□ Alias

Data Type: string

Data Length: 1

Initial Value: 0

Alias:

Description:

OK

Cancel

Figuur 3.10

4) In Figuur 3.10, set the data properties according to the definitions below:

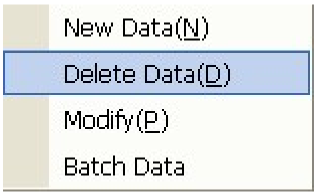
- **Data Name**: Unique name for a data variable used in an application. Data variables of the same database must not share the same names. Names of data variables are case sensitive. Click anywhere in the text box to edit the property; for example, to type in the variable name.
- **Data Type**: To define the type of the data. Click on the arrow down button to select a data type from the drop-down list.
- **Data Length**: The data length varies from variable to variable. For example, for a long data, you can define the data length to four bytes. You can define the data length for the other variables similarly.

- **Default Value**: To define the default value of variables. The default value is necessary in many situations. For example, when you design a ladder diagram, you need to define a timer; the timer timing uses the default value.
- **Alias**: To give the real-time data in the real-time database another name as identifier. When you refer to this alias, you are actually refering the original data. To use the alias, check the Alias checkbox, and type in the alias of the data (namely, the original data name). Click on **OK**, and a real-time data is created in the database.

3.4.4.2 Deleting Real-Time Data

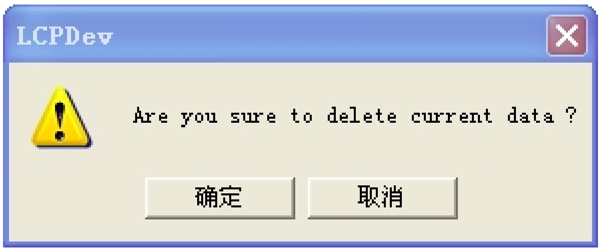
To delete real-time data, do as follows:

1) Right-click on the real-time data you want to delete, and you will see a right-click menu as shown in Figuur 3.11.



Figuur 3.11

2) Select **Delete Data**, and you will see a dialob box as shown in Figuur 3.12.


LCPDev

Are you sure to delete current data ?

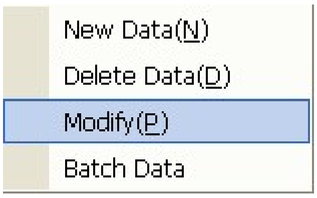
Figuur 3.12

3) Click on **OK**, and the real-time data will be deleted.

3.4.4.3 Modifying Data

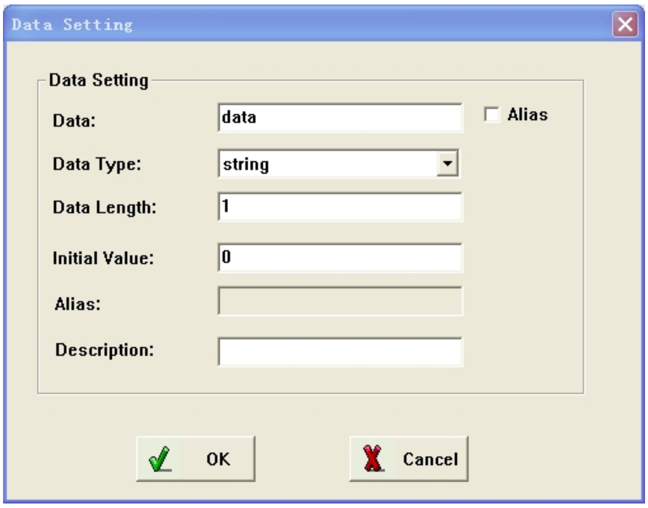
To modify real-time data, do as follows:

1) Right-click on the real-time data you want to modify, and you will see a right-click menu as shown in Figuur 3.13.



Figuur 3.13

2) Select **Modify**, and you will see a dialog box as shown in Figuur 3.14.



Data Setting

Data: data

□ Alias

Data Type: string

Data Length: 1

Initial Value: 0

Alias:

Description:

Figuur 3.14

3) Do the modifications, and click on **OK** to save them.

3.4.4.4 Adding Batch Data

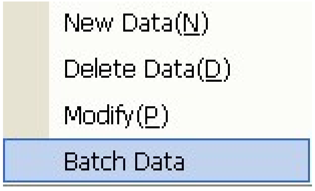
You can add a batch of data in the real-time database at a time, which saves the trouble of repeatedly defining data.

Names of the batch data must follow the following naming conventions:

The names of all the data in the batch must start with the same prefix and end with continuously incremental integers.

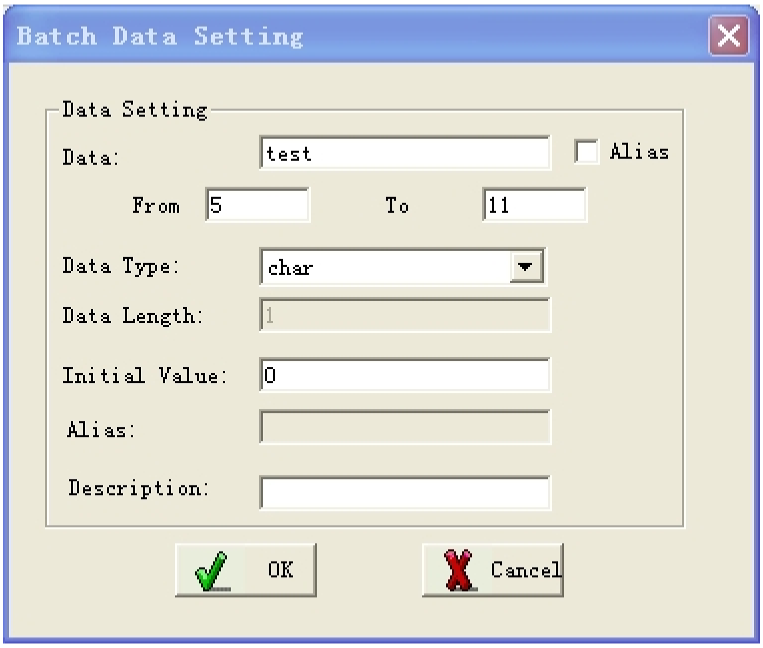
This section takes adding a batch of 7 data **test5** to **test11** into the database for example, to explain how to add batch data.

1) On the left side of the **Project Manager** window, select the real-time database into which you are going to add the batch data, and then right-click in the list pane on the right side of the window, as shown in Figuur 3.15.



Figuur 3.15

2) Select **Batch Data**, and you will see a dialog box as shown in Figuur 3.16.



Batch Data Setting

Data Setting

Data: test ☐ Alias

From 5 To 11

Data Type: char

Data Length: 1

Initial Value: 0

Alias:

Description:

Figuur 3.16

The parameters in Figuur 3.16 are described as follows:

- **Data Name**: The common prefix for the names of the batch data. It is **test** in this example.
- **From** … **To** …: Start and end values of the integer suffix for the names of the batch data. They are 5 and 11 in this example.

For descriptions about the other parameters, see section 3.4.4.1 Adding Real-Time Data.

3.5 Data Referencing

The data defined in the real-time database can be referenced directly in the interface configuration script.

Follow the rules below for data referencing:

1) If the data is not a parameter of a system function, refer to the data as **$Database Name.Data Name**. For example, you can refer to the data **data1** of the database **test** as **$test.data1**.

2) If the data is a parameter of a system function and the data is a string, refer to the data as **Database Name.Data Name**. (To be noted, no $ in the front.) For example, **data_input_window**(“**test.data1**”, “**test**”, 0, 100, 0).

3.6 System Variables of EASY

EASY has some internally defined data variables which are called system variables of the EASY configuration software. You can directly use these system variables for configuration. You can also use them for accessing or modifying the EASY internal system parameters, so as to implement some specific functions.

Chapter 4 Interface Database

4.1 Overview

Besides the real-time database, the EASY system also provides the interface database. Different from the real-time database, the data in the interface database can only be used during interface configuration; they cannot be used in the ladder diagram, real-time or historical records, alarms, or device configuration.

Data types and naming conventions for the interface database are the same as for the real-time database. For details, please see sections 3.3 and 3.4.1.

The interface database contains two types of data, real-time associated data and user-defined data, which will be described in details in the following sections.

4.2 Real-Time Associated Data

The real-time associated data must be associated with a certain data in the real-time database. There is one-to-one mapping relatsionship between the real-time associated data and the original real-time data to which it is associated. In other words, any modifications on the real-time associated data will be reflected on the original real-time data in the real-time database, and vise versa.

Defining the real-time associated data provides the following two advantages:

1) Defining a cycle time for each real-time associated data.

Usually, the system obtains all the real-time data from the real-time database and uploads them to the interface according to the default cycle time. In the EASY system, you can define the cycle time through the system variable **system.HmiDefCycleTime**, and the default cycle time is 500ms.

However, some data might have special characteristics. For example, some data might be changing fast, and thus needs to be refreshed and then uploaded to the interface more frequently. In this case, you can define a cycle time especially for these data; the system then refreshes the value of such data in the real-time database and uploads the data to the interface at the defined cycle time. Meanwhile, the refreshing and uploading for the other data still follow the default cycle time.

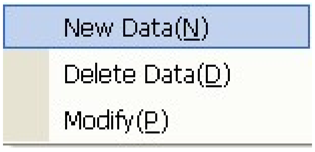
2) Accessing the data with the data name only, instead of with the database name as well.

For details, see section 4.4 Data Referencing.

4.2.1 Adding Data

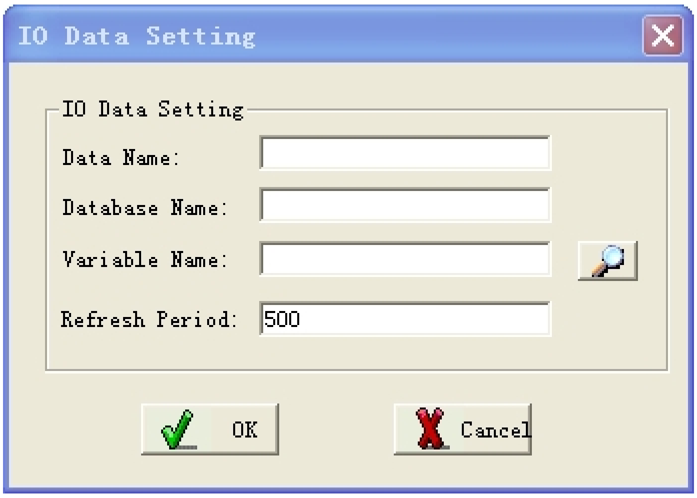
To add a real-time associated data, do as follows:

1) On the left side of the **Project Manager** window, select **Real-Time Associated Data**, and right-click in the list pane on the right of the window, as shown in Figuur 4.1.



Figuur 4.1

2) Select **Add Data**, and you will see a dialog box as shown in Figuur 4.2.



IO Data Setting

IO Data Setting

Data Name:

Database Name:

Variable Name:

Refresh Period: 500

Figuur 4.2

In Figuur 4.2, set the data properties according to the definitions below:

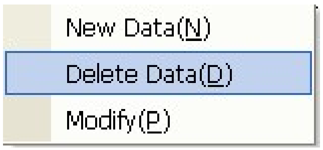
- **Data Name**: Unique name for identifying a data variable used in an application. The data variables of the same database must have different names. Names of data variables are case sensitive.
- **Database Name**: Name of the database which contains the real-time data to which the real-time associated data is associated.
- **Variable Name**: Name of the data variable in a real-time database to which the real-time associated data is associated.

  You can also click on the 🔍 button on the right side to select the database name and variable name of the data to which the real-time associated data is associated.
- **Cycle Time**: To define the cycle time for refreshing the real-time associated data.

4.2.2 Deleting Data

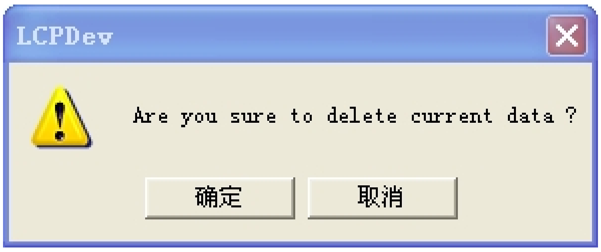
To delete a real-time associated data, do the following:

1) Right-click on the real-time data you want to delete, and you will see a right-click menu as shown in Figuur 4.3.



Figuur 4.3

2) Select **Delete Data**, and you will see a dialog box as shown in Figuur 4.4.



Are you sure to delete current data ?

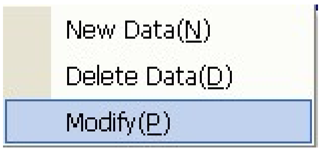
Figuur 4.4

3) Click on **OK**, and the selected real-time associated data is deleted.

4.2.3 Modifying Data

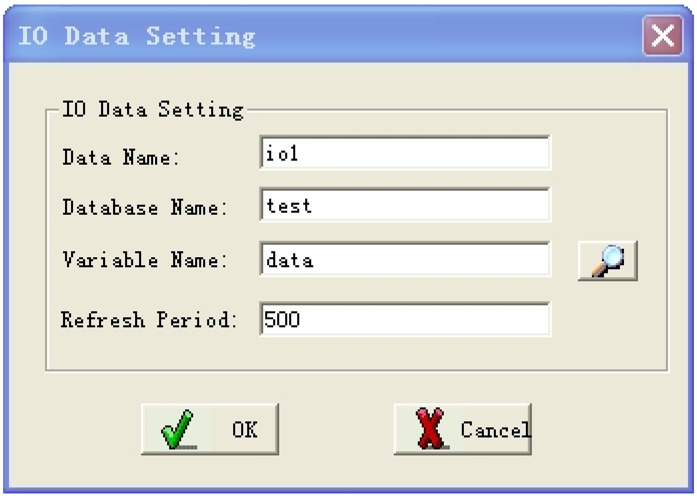
To modify a real-time associated data, do as follows:

1) Right-click on the real-time data you want to modify, and you will see a right-click menu as shown in Figuur 4.5.



Figuur 4.5

2) Select Modify, and you will see a dialog box as shown in Figuur 4.6.



Data Name: io1

Database Name: test

Variable Name: data

Refresh Period: 500

Figuur 4.6

3) After you complete modifying the data, click on **Save** to save the modifications.

4.3 User-Defined Data

The user-defined data is only used for interface configuration. It is unnecessary to associate this type of data with the data in the real-time database. Besides, accessing the user-defined data from the interface database is much faster than accesing the data from the real-time database.

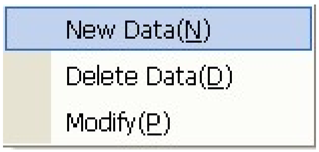
Therefore, for the data which is only used for interface control but not for the ladder diagram, real-time or historical recordings, alarms, or device configuration, it is recommended to define them as user-defined data in the interface database.

This following sections describe how to define the user-defined data in details.

4.3.1 Adding Data

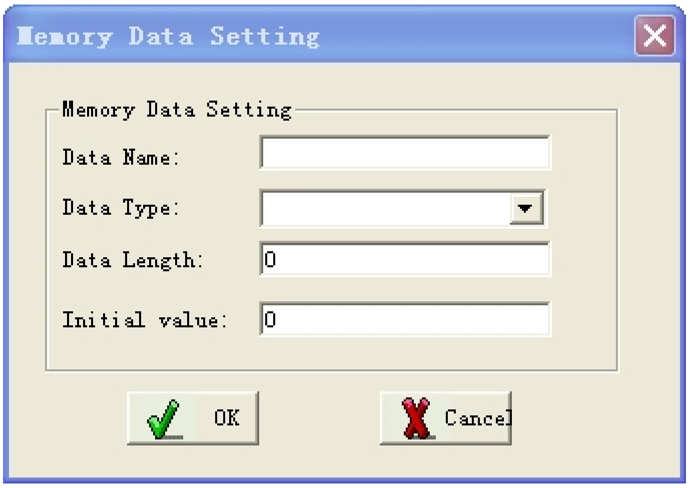
To add a user-defined data, do as follows:

1) On the left side of the **Project Manager** window, select **User-Defined Data** and right-click in the list pane on the right side of the window, as shown in Figuur 4.7.



Figuur 4.7

2) Select **Add Data**, and you will see a dialog box as shown in Figuur 4.8.



Memory Data Setting

Data Name:

Data Type:

Data Length: 0

Initial value: 0

Figuur 4.8

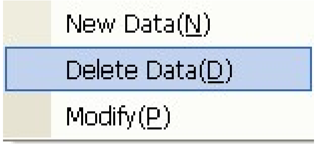
In Figuur 4.8, set the data properties according to the definitions below:

- **Data Name**: Unique name for identifying a data variable used in an application. The data variables of the same database must have different names. Names of data variables are case sensitive. Click anywhere inside the text box to start typing. If you are a project engineer, you can type in here the variable name.
- **Data Type**: Type of the data. Click on the arrow down button to select from the list of data types provided for your selection.
- **Data Length**: The length of the data varies from variable to variable. For example, the data length for a long data is 4 bytes. The data length for the other types of data follows the specific rules accordingly.
- **Initial Value**: Initial value of a variable. It is necessary to set an initial value. For example, the timer of a ladder diagram needs an initial value for the timing.

4.3.2 Deleting Data

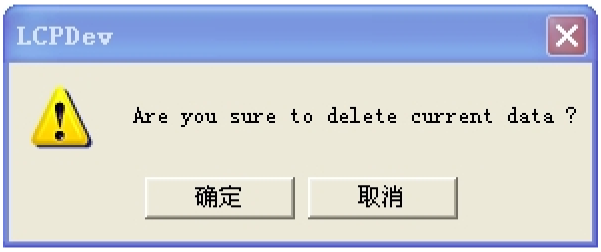
To delete a user-defined data, do as follows:

1) Right-click on a real-time data you want to delete, and you'll see a right-click menu as shown in Figuur 4.9.



Figuur 4.9

2) Select **Delete Data**, and you will see a dialog box as shown in Figuur 4.10.


LCPDev

Are you sure to delete current data ?

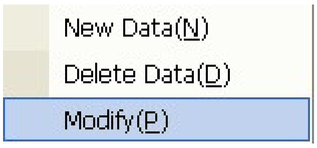
Figuur 4.10

3) Click on **OK**, and the selected user-defined data is deleted.

4.3.3 Modifying Data

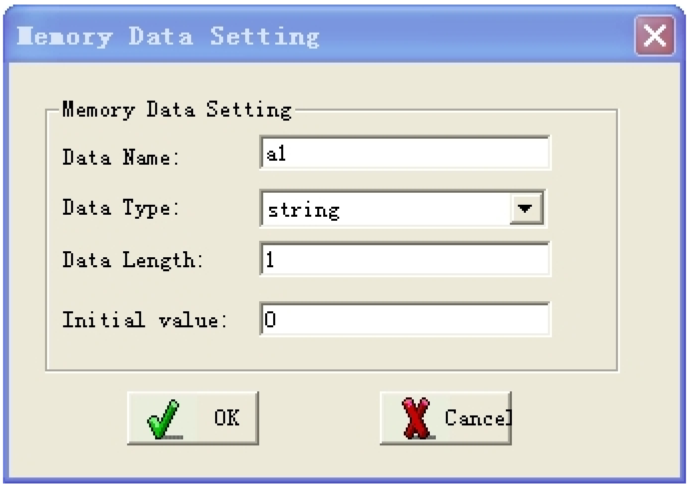
To modify a user-defined data, do as follows:

1) Right-click on a user-defined data, and you will see a right-click menu as shown in Figuur 4.11.



Figuur 4.11

2) Select **Modify**, and you will see a dialog box as shown in Figuur 4.12.


Memory Data Setting

Memory Data Setting

Data Name: a1

Data Type: string

Data Length: 1

Initial value: 0

Figuur 4.12

3) Click on **OK** to save all the modifications.

4.4 Data Referencing

Different from refering to the data of the real-time database, it is unnecessary to define the database name while refering to the data of the interface database. Please follow the two rules below:

1) When the data does not work as a system function, refer to the data as **$Data**

Name. For example, you can use **$pic_data1** to refer to the data **pic_data1** of the interface database.

2) When the data works as a system function and the data is of the string data type, refer to the data as **Data Name**. (To be noted, no $ in the front.) For example, **data_input_window**(“pic_data”, “test”, 0, 100, 0).

Chapter 5 Interface Configuration

EASY provides convenient, flexible, and powerful interface configuration functions.

EASY supports various basic graphic controls, such as:

- Rectangles (including incremental rectangles)
- Lines
- Ovals (including circles)
- Images
- Text
- Buttons
- Checkboxes
- Drop-down boxes
- Timers
- Trend curve windows
- Alarm windows, and
- Historical lists

Besides, EASY provides a graphics library, which has controls that might be used in various industries, such as pumps, pedestal actuators, and digitrons.

You can easily conFiguur the properties of these controls. Some of the properties are called dynamic properties, which means they change dynamically during the operation. These dynamic properties use the standard C language scripts. In other words, they not only follow the syntax of the C language, but also support the functions from the function library of the C language. Because of all this, EASY is highly flexible and can achieve a rich variety of functions which are impossible for the traditional configuration methods.

5.1 Interface Windows

EASY allows configuring graphic interfaces for an application based on windows. After creating a window, you can add various types of graphic objects into it and define their properties, so as to achieve nice and vivid dynamic interfaces of different styles.

5.1.1 Naming Conventions

Names of windows follow the following naming conventions:

1) Naming rules for identifiers of the C language: Chinese not supported; starting with letters which are followed by letters, digits, or underscores.

2) Window names are case sensitive.

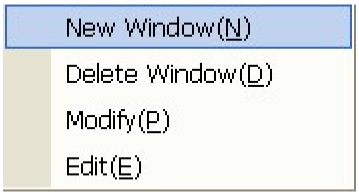
3) The following window names are reserved for the system internal use only, and thus are not available for user selection:

sys_set_time_wnd, **sys_link_timeout_wnd**, **hmi_sys_set_wnd**, and all window names starting with **EASY**.

5.1.2 Creating an Interface Window

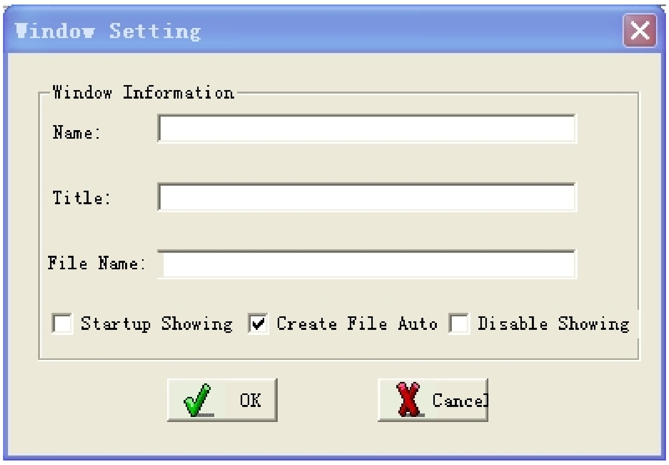
To create an interface window, do as follows:

1) On the left side of the **Project Manager** window, select **Interface** and then right-click in the list pane on the right side, as shown in Figuur 5.1.



Figuur 5.1

2) Select **Add an Interface**, and you will see a dialog box as shown in Figuur 5.2.



Name:

Title:

File Name:

□ Startup Showing ☑ Create File Auto □ Disable Showing

Figuur 5.2

In Figuur 5.2, set the interface parameters according to the definitions below:

- **Name**: Name of the new interface window; unique identifier of a window. All the windows in the EASY system have different names.
- **Title**: Title of the new interface window. After you set the title property to **Display the title**, and then this title will be displayed as the window title.
- **File Name**: The EASY system saves all the window-related property information into an **xml** file. This parameter defines the name of the **xml** file.
- The **Display at application startup** checkbox: If you check this checkbox, the window name, title, and the xml file name will be displayed at the application startup; otherwise, they will not be displayed.

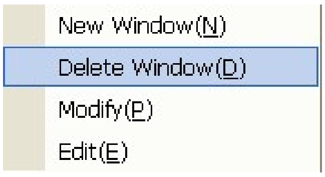
- The **Auto create file** checkbox: If you check this checkbox, the system will automatically create an **xml** file and save the window-related information to that **xml** file; otherwise, the system will allow associate this window to an existing **xml** file rather than creating a new **xml** file.
- The **Disable the interface** checkbox: If you check this checkbox, the system will not display the interface and all graphic components on this interface. In addition, none of the related dynamic properties and scripts will not be executed.

5.1.3 Deleting an Interface Window

To delete an interface window, do as follows:

1) On the left side of the **Project Manager** window, select **Interface** and then right-click on a window name you want to delete in the list pane on the right side of the window.

   You will see a right-click menu as shown in Figuur 5.3.



Figuur 5.3

2) Select **Delete an Interface**, and the selected interface window will be deleted from the system.

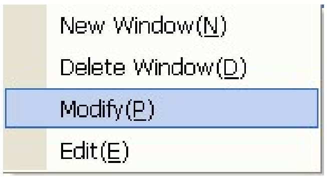
Note: After you select **Delete an Interface**, only the interface window is deleted from the project configuration; the **xml** file which stores the window-related information is not deleted from the system.

5.1.4 Modifying the Window Configuration

To modify the configuration of an interface window, do as follows:

1) On the left side of the **Project Manager** window, select **Interface** and then right-click on a window for which you want to modify the configuration in the list pane on the right side of the window.

   You will see a right-click menu as shown in Figuur 5.4.



Figuur 5.4

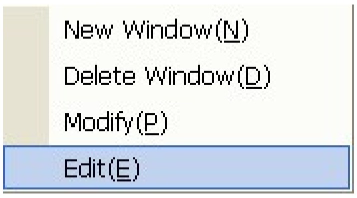
2) Select **Modify** to modify the window configuration.

5.1.5 Editing an Interface Window

To edit an interface window, do as follows:

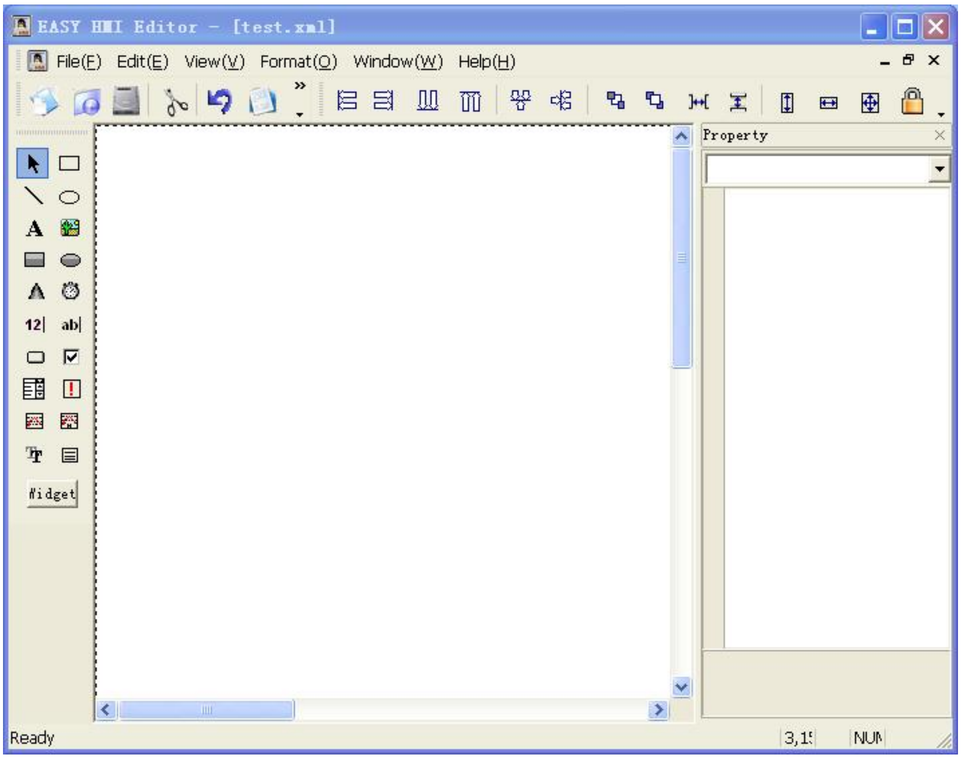
1) On the left side of the **Project Manager** window, select **Interface** and then double-click (or right-click) on a window which you want to edit in the list pane on the right side of the window.

   You will see a menu as shown in Figuur 5.4.



Figuur 5.5

2) Select **Edit**, and you will see the **Interface Editor** window as shown in Figuur 5.6.



Figuur 5.6

In this **Interface Editor** window, you can add graphic components, set static and dynamic properties for the added graphic components, or implement other interface configuration tasks.

5.2 Interface Editor

On the left side of the **Project Manager** window, select **Interface** and then double-click (or right-click) on a window which you want to edit in the list pane on the right side of the window. Select **Edit**, and you will see the **Interface Editor** window, as shown in Figuur 5.6.

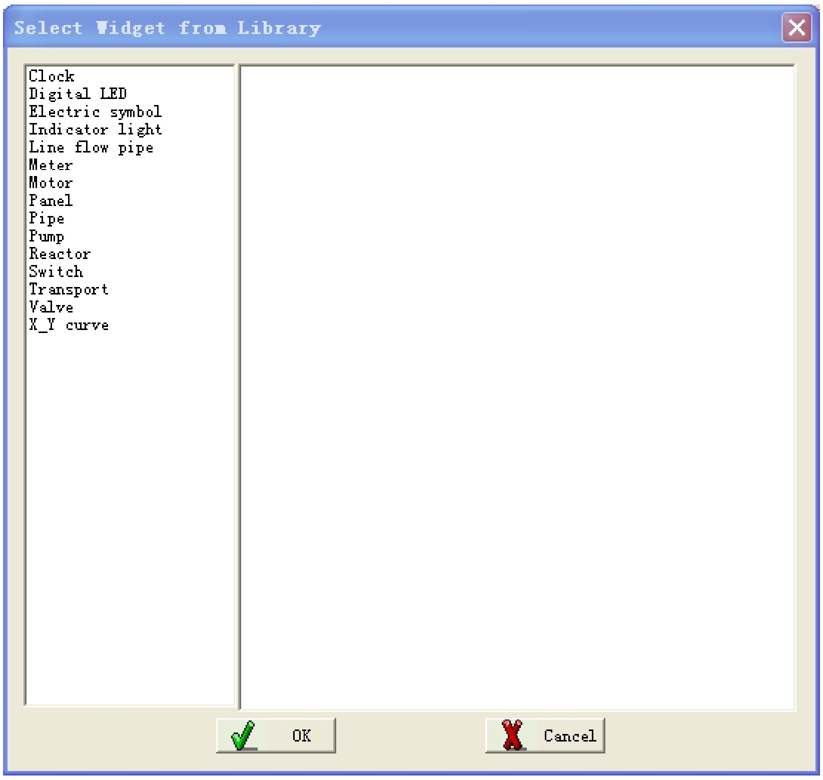
5.2.1 Adding a Graphic Component

On the left side of the **Interface Editor** window is the toolbox area, which lists the basic graphic controls supported by the system and the Graphics Library button as well.

To add a basic graphic control, you just need to select the graphic component with a left click.

To add a control from the graphics library, do as follows:

1) Click on the **Graphics Library** button, and you will see the **Select from Library** window as shown in Figuur 5.7.



Select Widget from Library

Clock
Digital LED
Electric symbol
Indicator light
Line flow pipe
Meter
Motor
Panel
Pipe
Pump
Reactor
Switch
Transport
Valve
X_Y curve

Figuur 5.7

2) Select a graphic control you want to add from the graphics library and click on **OK**.

3) Move the cursor to the editting area on the right side of the toolbox area in the **Interface Editor** window.

   And you will see the cursor becomes a cross.

4) Click anywhere in the editting area to add the graphic control.

5.2.2 Deleting a Graphic Component

To delete a graphic component, select a graphic control you want to delete, and press the **Delete** key from your keyboard. And the selected graphic component will be deleted.

5.2.3 Layout of Graphic Components

In the **Interface Editor** window, select the **Format** menu, and you can see all the function menus related to layout of graphic components.

5.2.3.1 Selecting Multiple Graphic Components

You can select multiple graphic objects in the following two ways:

- Using the **Ctrl** key. Do as follows:
  1) Select any one graphic object with a left click.
  2) Press and hold the **Ctrl** key on your keyboard while clicking on any other graphic object.
- By dragging the mouse. Do as follows:
  1) Place the cursor in a point where all the target graphic objects can be covered when you drag the mouse.
  2) Click on the top left, top right, bottom left, or bottom right rectangle of a graphic object by pressing the left button of the mouse.
  3) Drag the mouse across the target graphic objects.
     You will see a big rectangle in dot lines covering all the target graphic objects.
  4) Release the mouse.
     All the graphic objects inside the dot-lined rectangle are selected.

Each selected graphic object has 8 small rectangles on its sides. But among all the selected objects, only one object has solid rectangles, while the others are all hollow. The following formatting operations are all implemented on the graphic object with the solid rectangles.

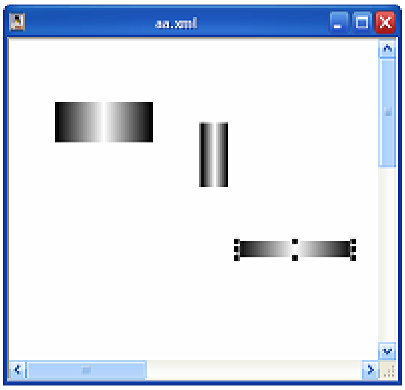
5.2.3.2 Formatting Graphic Components - Alignment

In the **Interface Editor** window, select **Format** and then **Alignment**, and you can see a list of cascading menus which are described in the table below.

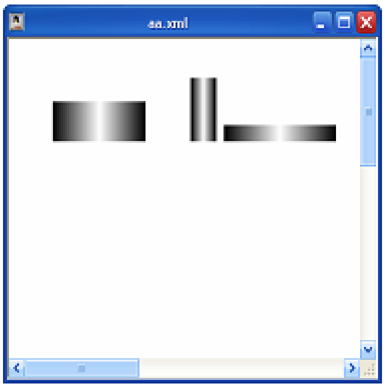
Menu	Description

Top Alignment	Aligns multiple selected objects with the top border of the topmost object. To achieve top alignment, select multiple graphic objects, and then select **Format** > **Alignment** > **Top Alignment**.
Center Alignment	Aligns two or more graphic objects by putting their centers on the same vertical line. To achieve center alignment, select multiple graphic objects, and then select **Format** > **Alignment** > **Center Alignment**.
Bottom Alignment	Aligns two or more selected objects with the bottom border of the bottom-most object. To achieve bottom alignment, select multiple graphic objects, and then select **Format** > **Alignment** > **Bottom Alignment**.
Left Alignment	Aligns two or more selected objects with the left border of the left-most object. To achieve left alignment, select multiple graphic objects, and then select **Format** > **Alignment** > **Left Alignment**.
Right Alignment	Aligns two or more selected objects with the right border of the right-most object. To achieve left alignment, select multiple graphic objects, and then select **Format** > **Alignment** > **Right Alignment**.

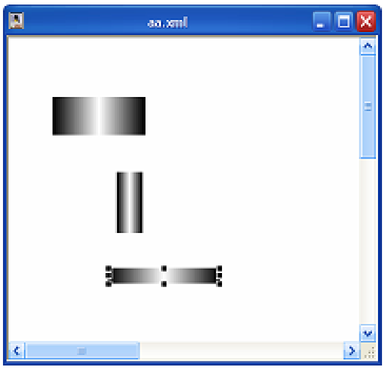
Select the alignment menus described in the table above, and you will see the different effects as shown below:



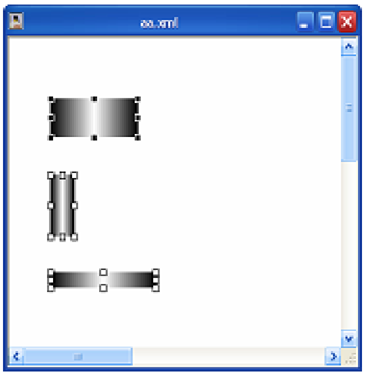
Figuur 5.8 Before executing **Bottom Alignment**



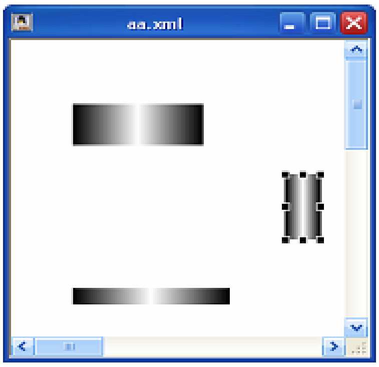
Figuur 5.9 After executing **Bottom Alignment**



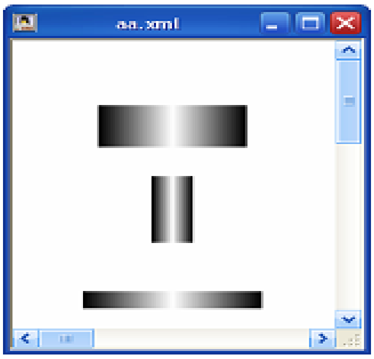
Figuur 5.10 Before executing **Left Alignment**



Figuur 5.11 After executing **Left Alignment**



Figuur 5.12 Before executing **Center Alignment**



Figuur 5.13 After executing **Center Alignment**

5.2.3.3 Formatting Graphic Components - Measurement

In the **Interface Editor** window, select **Format** and then **Measurement**, and you can see a list of cascading menus which are described in the table below.

Menu	Description
Same Width	Sets multiple selected objects to the same width. To achieve same width, select multiple graphic objects, and then select **Format** > **Measurement** > **Same Width**.
Same Height	Sets multiple selected objects to the same height. To achieve same height, select multiple graphic objects, and then select **Format** > **Measurement** > **Same Height**.
Same Width and Height	Sets multiple selected objects to the same width and height. To achieve same width and height, select multiple graphic objects, and then select **Format** > **Measurement** > **Same Width and Height**.

5.2.3.4 Formatting Graphic Components - Spacing

In the **Interface Editor** window, select **Format** and then **Spacing**, and you can see a list of cascading menus which are described in the table below.

Menu	Description
Horizontal Spacing\Same Spacing	Sets the horizontal spacing between the multiple selected objects to the same. To achieve same horizontal spacing, select multiple graphic objects, and then select **Format** > **Horizontal Spacing** > **Same Spacing**.
Vertical Spacing \Same Spacing	Sets the vertical spacing between the multiple selected objects to the same. To achieve same vertical spacing, select multiple graphic objects, and then select **Format** > **Vertical Spacing** > **Same Spacing**.

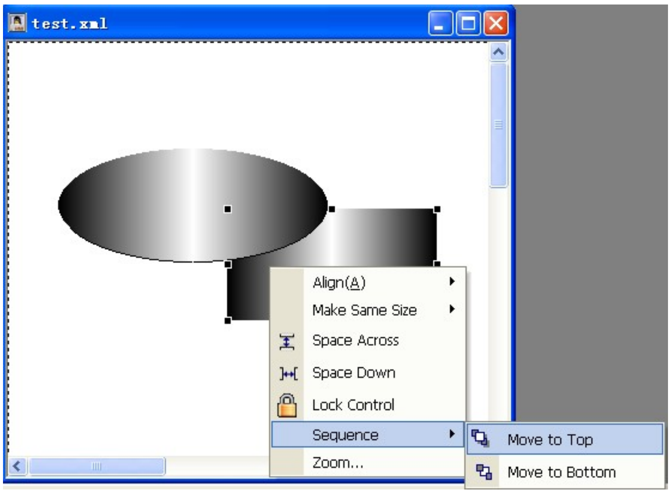
5.2.3.5 Formatting Graphic Components - Layering

In the **Interface Editor** window, select **Format** and then **Layering**, and you can see a list of cascading menus which are described in the table below.

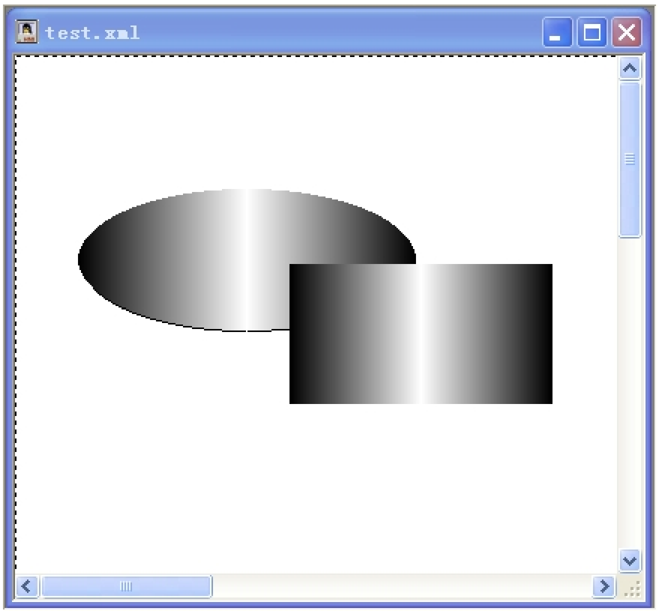
Menu	Description
Move to Top	Moves one or more selected objects to the top layer as the foreground for the overlapped graphic objects.
Move to Bottom	Moves one or more selected objects to the bottom layer as the background for the overlapped graphic objects.

The two graphic components in Figuur 5.14 are overlapped. The incremental rectangle is behind the incremental oval. To move the incremental rectangle to the front of the incremental oval, you need to right-click on the incremental rectangle and select **Move to Top**.

Figuur 5.14 shows how the objects look like before the **Move to Top** command is executed; Figuur 5.15 shows how they look like after the **Move to Top** command is executed.



Figuur 5.14

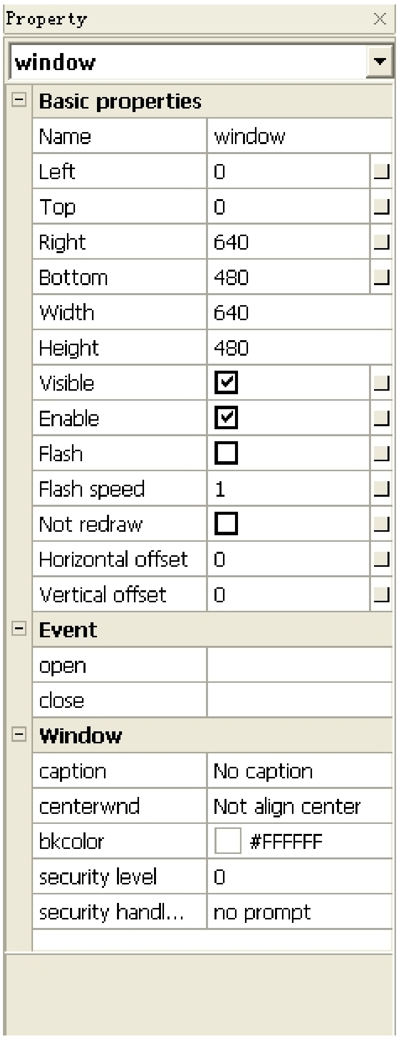


Figuur 5.15

5.3 Properties of Graphic Components

5.3.1 Overview

Select a graphic component, and the properties of this graphic component will be listed on the right side of the Interface Editor window, as shown in Figuur 5.16.



Basic properties		
Name	window	
Left	0	❏
Top	0	❏
Right	640	❏
Bottom	480	❏
Width	640	
Height	480	
Visible	☑	❏
Enable	☑	❏
Flash	□	❏
Flash speed	1	❏
Not redraw	□	❏
Horizontal offset	0	❏
Vertical offset	0	❏
Event		
open		
close		
Window		
caption	No caption	
centerwnd	Not align center	
bkcolor	□ #FFFFFF	
security level	0	
security handl...	no prompt	

Figuur 5.16

Some properties have a small rectangle ❏ at the right-most side, as shown in Figuur 5.16. Click on the small rectangle ❏, and you can set the expression or dynamic script for this property. In this case, the value of the property will change during the operation. This type of properties are thus called dynamic properties. Once a property is conFiguurd as dynamic, the small rectangle will become red 🟥.

The property list is composed of the following five parts:

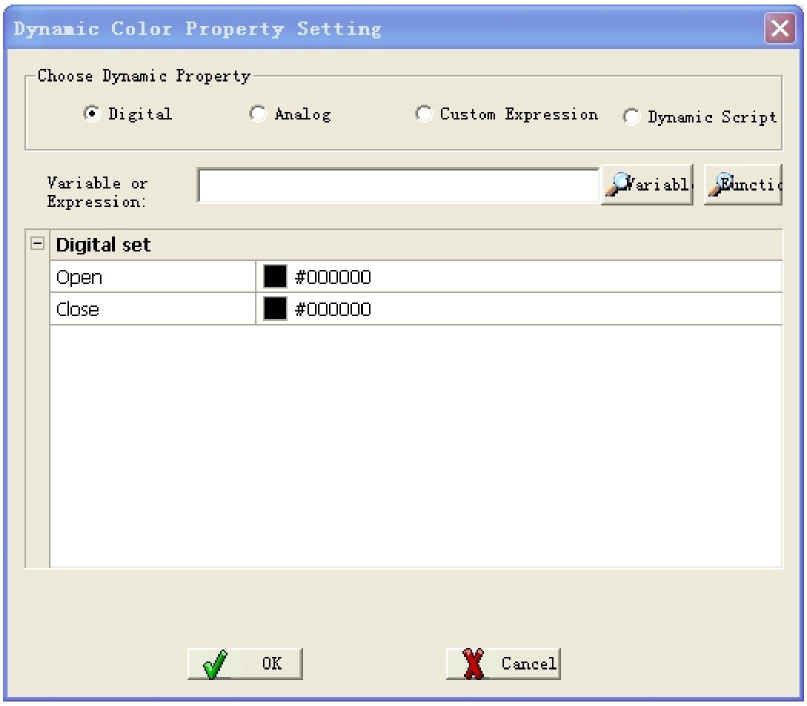
- **Graphic Components Drop-Down List Box**: Lists all the graphic components contained in the current window. You can select graphic components from this drop-down list for configuration or modification.
- **Basic Properties**: Common properties shared by all graphic components, such as **Top**, **Bottom**, **Left**, **Right**, **Width**, and **Height**.
- **Events**: Events supported by the selected graphic component. You can compile the event script, which can be executed when the corresponding event occurs.
- **Graphic-Control Specific Properties**: Each graphic control component has its own specific properties, which vary from component to component.
- **Property Description**: A brief description of a selected property is displayed at the bottom of the **Property List** pane.

Considering that basic properties are those shared by all control components, section 5.3.3 will be focusing on describing them. However, most basic properties and control-specific properties can be conFiguurd as dynamic. Therefore, dynamic properties will be introduced first.

5.3.2 Dynamic Properties

5.3.2.1 Dynamic Color Properties

ConFiguur the dynamic **Color** properties in the **Dynamic Color Properties Setting** dialog box as shown in Figuur 5.17.



Choose Dynamic Property

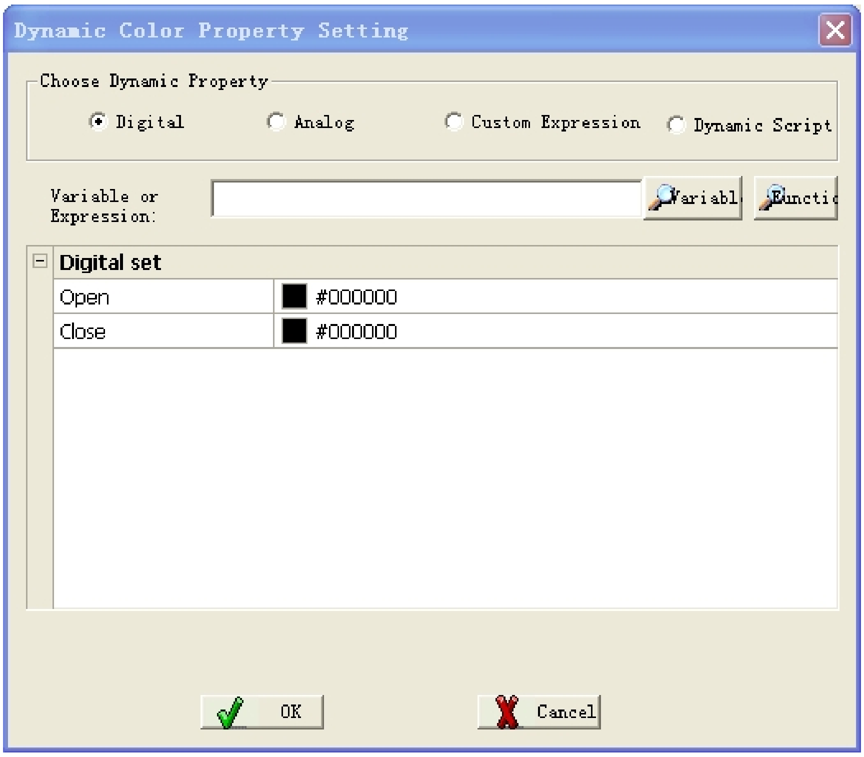
● Digital ○ Analog ○ Custom Expression ○ Dynamic Script

Variable or Expression:

Digital set	
Open	■ #000000
Close	■ #000000

Figuur 5.17

1. If you select **Dynamic Property Type** as **Digital**, you will see the configuration dialog box as shown in Figuur 5.18.



Dynamic Color Property Setting

Choose Dynamic Property

● Digital ○ Analog ○ Custom Expression ○ Dynamic Script

Variable or Expression:

Digital set

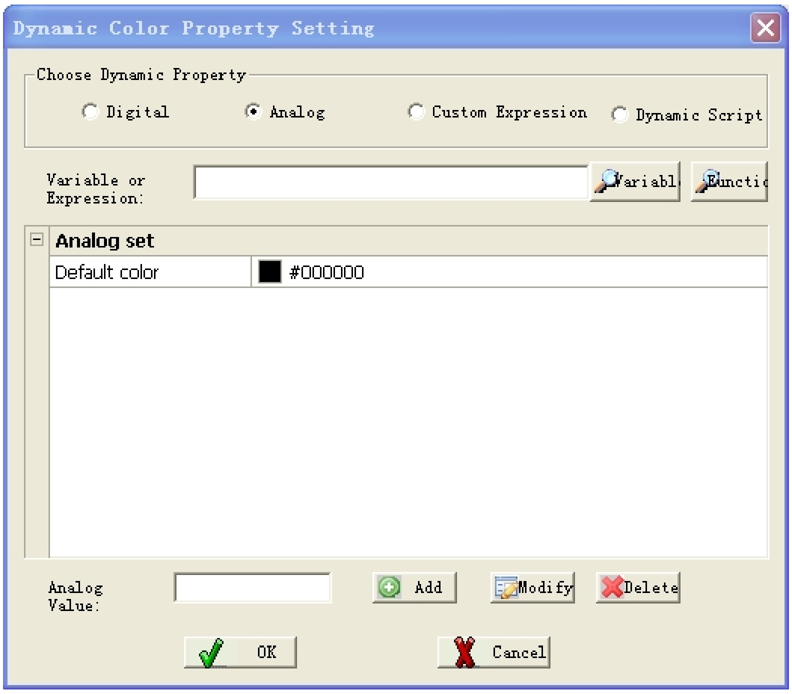
Open	■ #000000
Close	■ #000000

Figuur 5.18

The configuration parameters are described as follows:

- **Variable or Expression**: Covers all the bit database variables or all the expressions with the return value as bit.
- **Digital Setting/On**: Sets the color used when the bit variable or value of the expression is not 0 (TRUE).
- **Digital Setting/Off**: Sets the color used when the bit variable or value of the expression is 0 (FALSE).

2. If you select **Dynamic Property Type** as **Analog**, you will see the configuration dialog box as shown in Figuur 5.19.


Dynamic Color Property Setting

Choose Dynamic Property

○ Digital ● Analog ○ Custom Expression ○ Dynamic Script

Variable or Expression:

Analog set	
Default color	■ #000000

Analog Value:

Figuur 5.19

The configuration parameters are described as follows:

- **Variable or Expression**: Covers all the int/float database variables and all the expressions with the return value as int/float.
- **Analog Setting**: Sets the threshold value for the analog. This parameter sets the colors used when the analog threshold is more than or equal to a specific value. Note: Threshold values must be set incrementally from small to big.

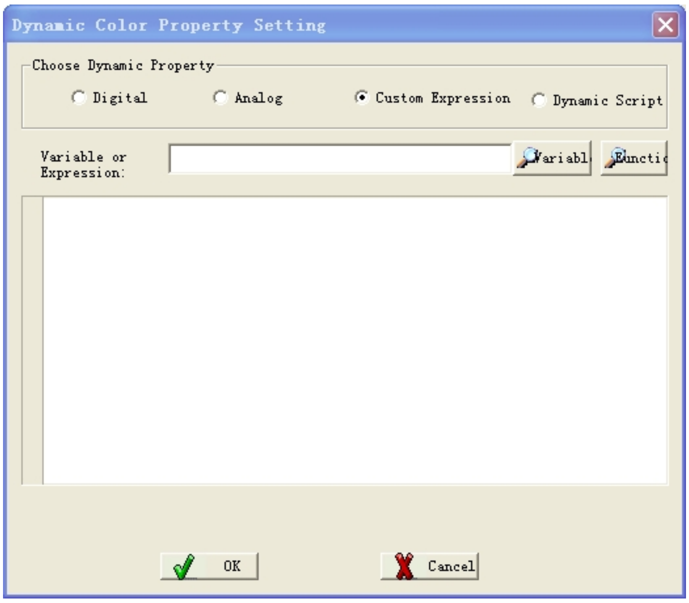
  For example:

  Default color: Black;
  10: Red;
  20: Green

Which means:

1) When the analog is < 10, the default color (black) is displayed;
2) When the analog is >= 10 and < 20, the red color is displayed;
3) When the analog is >=20, the green color is displayed.

3. If you select **Dynamic Property Type** as **Customized Expression**, you will see the configuration dialog box as shown in Figuur 5.20.



Figuur 5.20

The configuration parameter **Variable or Expression** is described as follows:

You can directly type an expression requesting that he return value of the expression must be an RGB value.

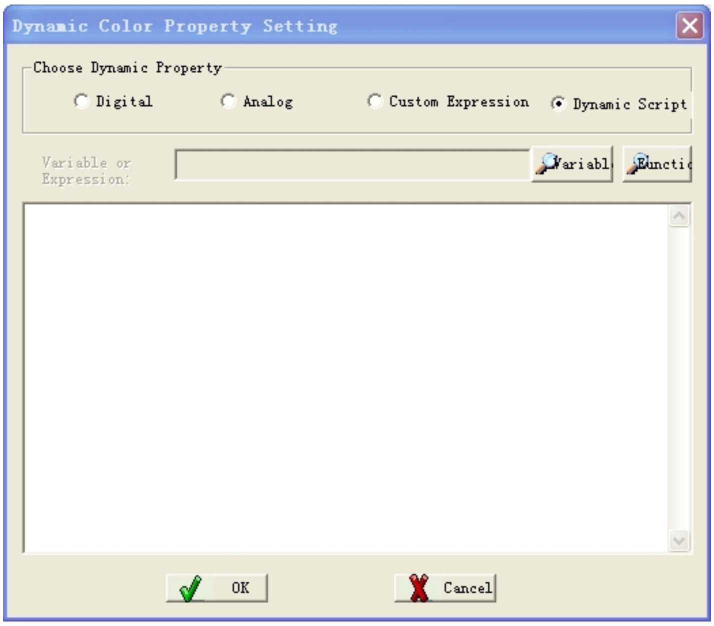
For example, **$test1.data2 > 10 ? 0x0000FF : 0x00FF00**.

Which means:

1) When the analog **$test1.data2** is > 10, the color is red (**0x0000FF** stands for the RGB value for the red color);

2) When the analog **$test1.data2** is <= 10, the color is green (**0x00FF00** stands for the RGB value for the green color).

4. If you select **Dynamic Property Type** as **Dynamic Script**, you will see the configuration dialog box as shown in Figuur 5.21.



Figuur 5.21

You can type the dynamic script in the editting area, requesting that the return value of the dynamic script is an RGB value.

For example:

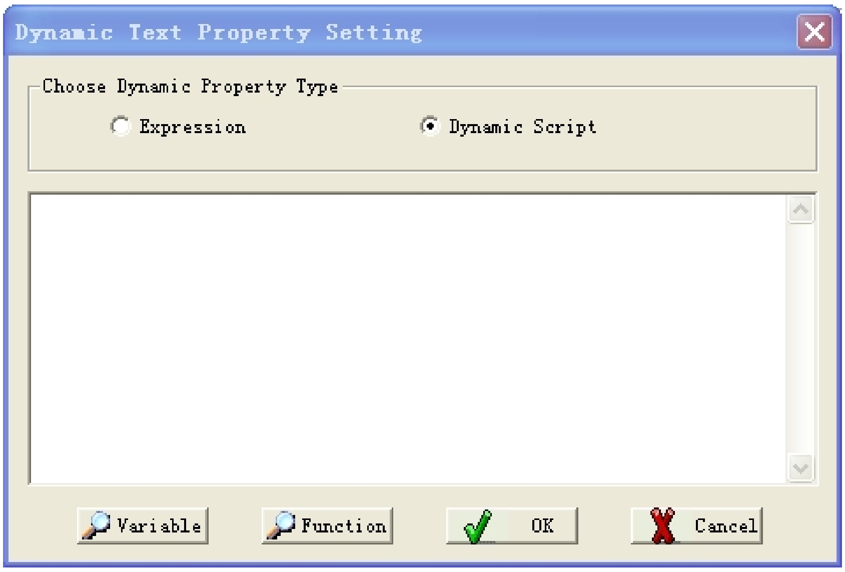
```
if ($test1.data2 > 10)
{
    return 0x0000FF;
}
else
{
    return 0x00FF00;
}
```

Which means:

1) When the analog **$test1.data2** is > 10, the color is red (**0x0000FF** stands for the RGB value for the red color);
2) When the analog **$test1.data2** is <= 10, the color is green (**0x00FF00** stands for the RGB value for the green color).

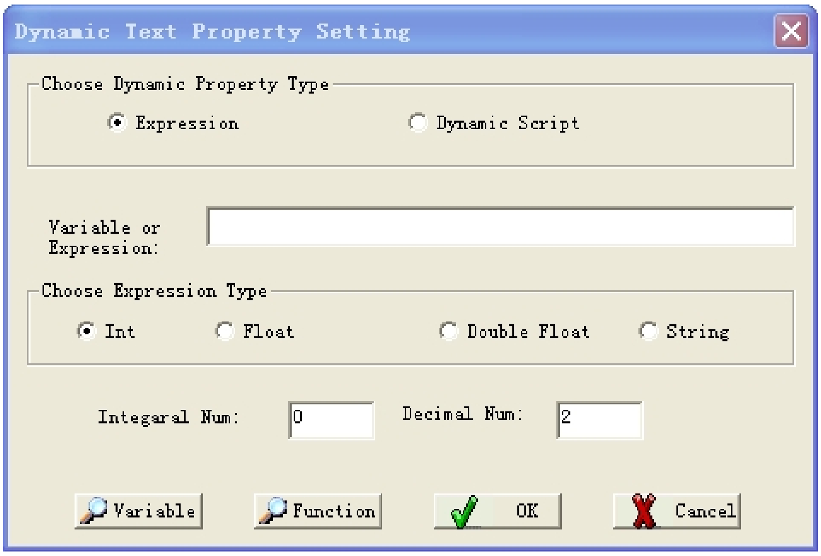
5.3.2.2 Dynamic Text Properties

ConFiguur the dynamic **Text** properties in the **Dynamic Text Properties Setting** dialog box as shown in Figuur 5.22.



Figuur 5.22

1. If you select **Dynamic Property Type** to **Expression**, you will see the configuration dialog box as shown in Figuur 5.23.



Choose Dynamic Property Type

● Expression ○ Dynamic Script

Variable or Expression:

Choose Expression Type

● Int ○ Float ○ Double Float ○ String

Integaral Num: 0 Decimal Num: 2

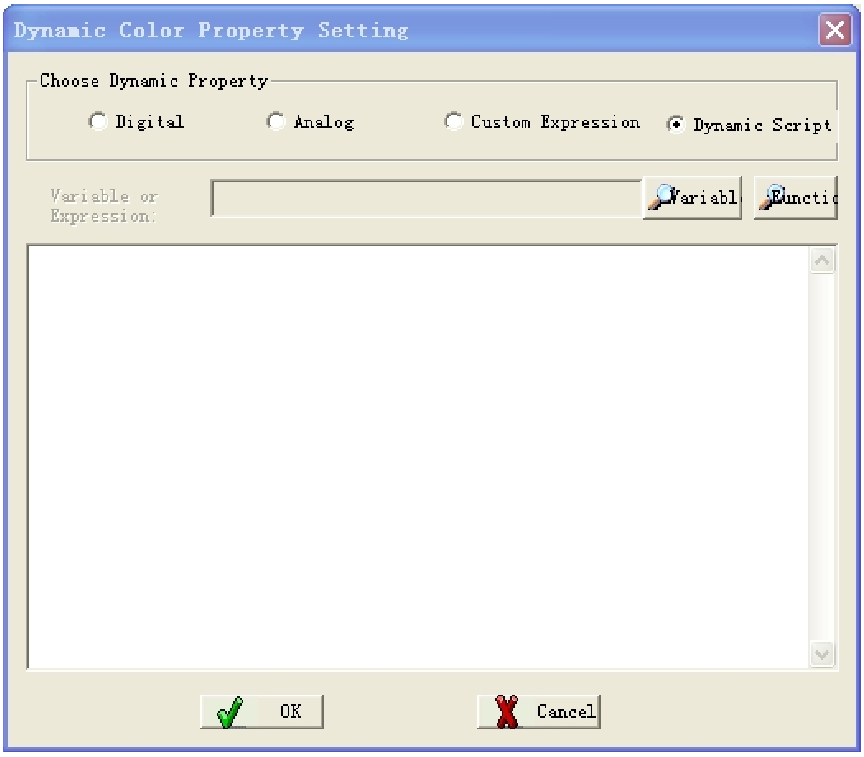
Figuur 5.23

The configuration parameters are described as follows:

- **Variable or Expression**: The database variable or the expression to which a graphic control component is associated.
- **Expression Type**, **Number of Integers**, and **Number of Decimals**: Controls how the variable or value of the expression is displayed as text.
  See the following table for detailed description.

Expression Type	Number of Integers	Number of Decimals	Remarks
Int	Minimum number of integers to be displayed. ● When the number of integers is less than enough, 0 is added on the left to make enough number of integers. ● When the number of integers is more than required, only the specified number of integers will be displayed. ● If you define **Number of Integers** to **0**, all the actual integers will be displayed.	N/A	The value of the variable or expression must be numeric, but not string.
Float		Fixed number of decimals to be displayed. ● When the number of decimals is less than enough, 0 is added on the right to make enough number of decimals. ● When the number of decimals is more than required, only the specified number of decimals will be displayed. ● If you define **Number of Decimals** to **0**, only integers will be displayed; decimals not.	
Double			
String	N/A	N/A	The value of the variable or expression must be string, but not numeric.

2. If you select **Dynamic Property Type** to **Dynamic Script**, you will see the configuration dialog box as shown in Figuur 5.24.



Dynamic Color Property Setting

Choose Dynamic Property

○ Digital ○ Analog ○ Custom Expression ● Dynamic Script

Variable or Expression:

Figuur 5.24

You can type the dynamic script in the editting area, requesting that the return value of the dynamic script must be a string.

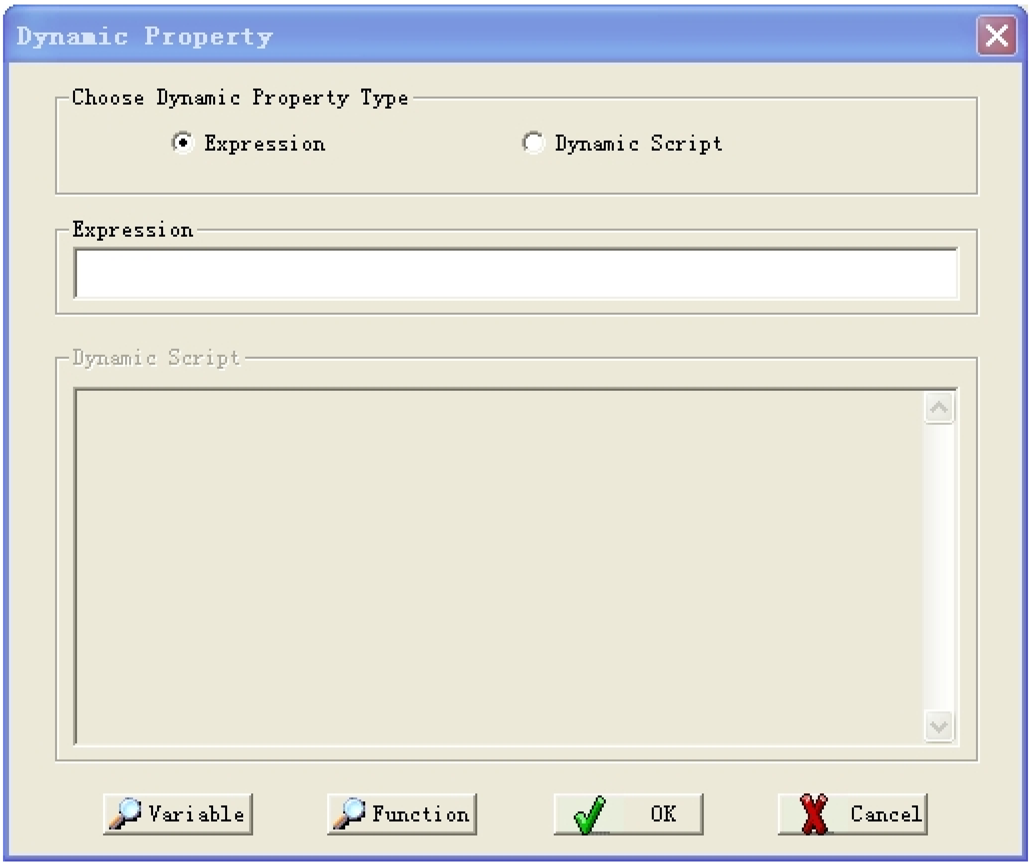
For example:

```
if ($test1.data2 > 10)
{
    return “aaaa”;
}
else
{
    return “bbbb”;
}
```

Which means when the analog **$test1.data2** is > 10, the text displayed is **aaaa**; otherwise, the text displayed is **bbbb**.

5.3.2.3 Other Dynamic Properties

For properties other than the **Color** or **Text** property, conFiguur the dynamic properties in the **Dynamic Properties Setting** dialog box as shown in Figuur 5.25.



Dynamic Property

Choose Dynamic Property Type

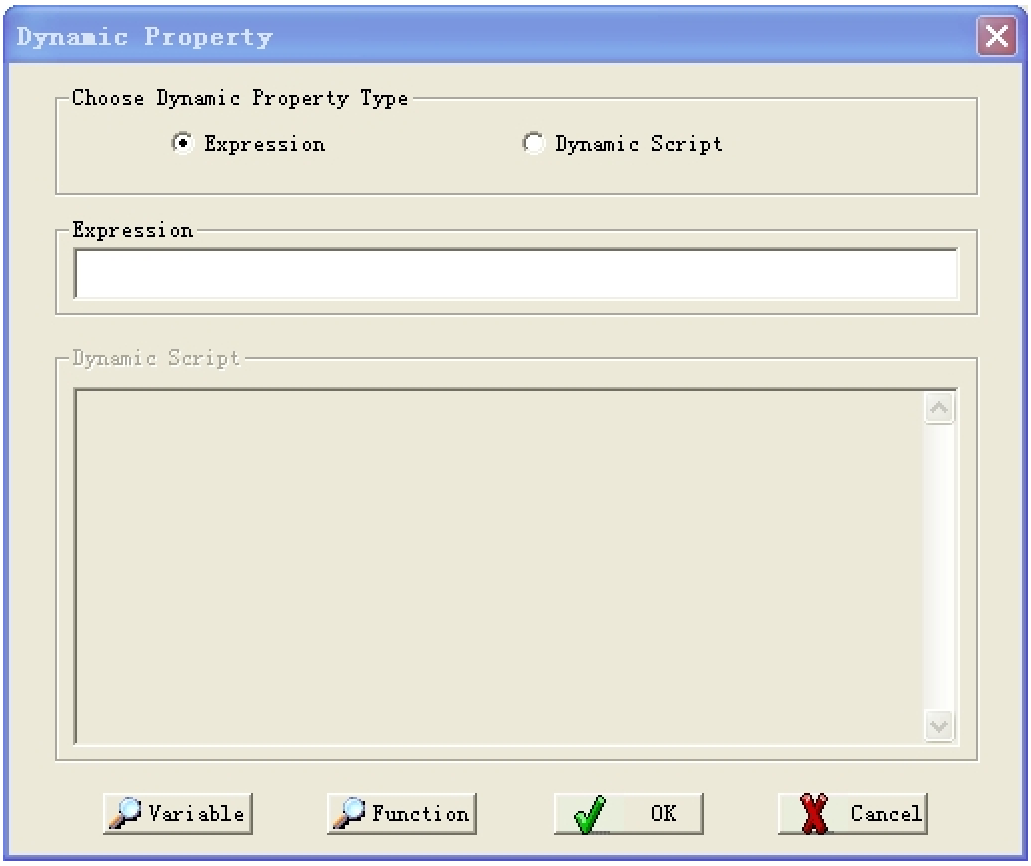
● Expression ○ Dynamic Script

Expression

Dynamic Script

Figuur 5.25

1. If you select **Dynamic Property Type** as **Expression**, you will see the configuration dialog box as shwon in Figuur 5.26.



Dynamic Property

Choose Dynamic Property Type

● Expression ○ Dynamic Script

Expression

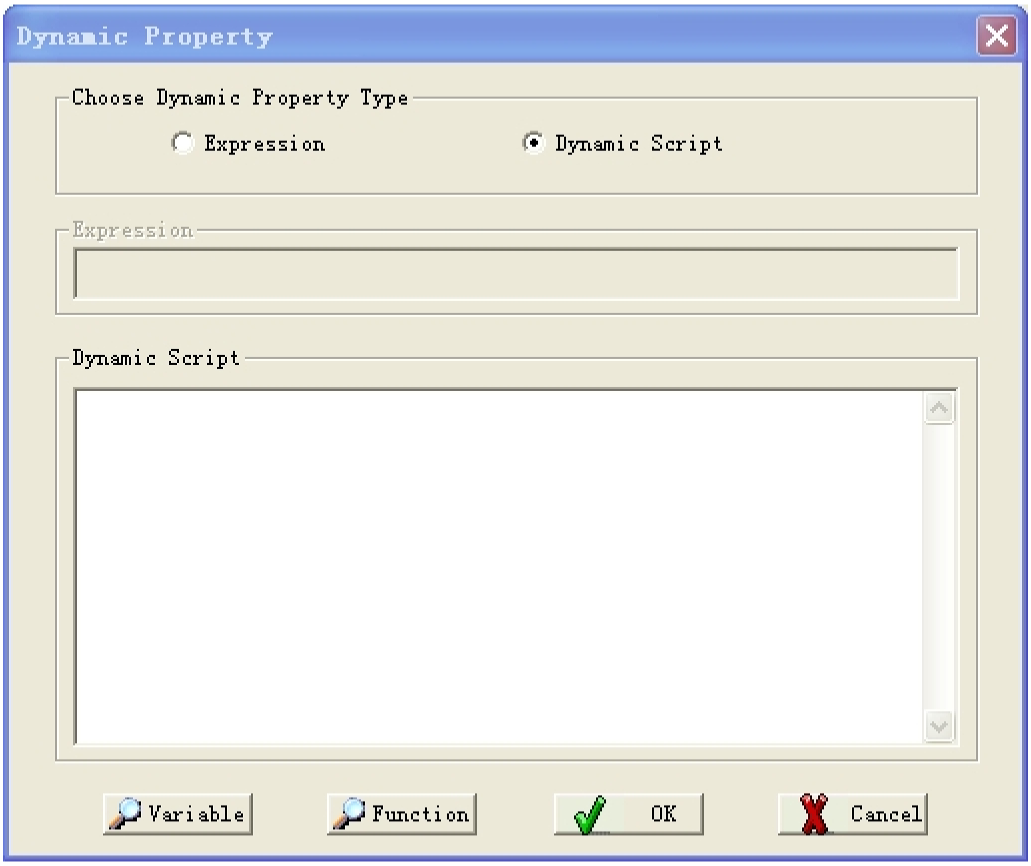
Dynamic Script

Figuur 5.26

The configuration parameter **Expression** is described as follows:

The expression to which a graphic control component is associated. The return value of the expression varies according to the specific characteristics of the property.

2. If you select **Dynamic Property Type** as **Dynamic Script**, you will see the configuration dialog box as shwon in Figuur 5.27.



Choose Dynamic Property Type

○ Expression ● Dynamic Script

Figuur 5.27

You can type the dynamic script in the editting area. The return value of the dynamic script varies according to the specific characteristics of the property.

5.3.3 Basic Properties

5.3.3.1 Name

Names of graphic control components in a window must be unique. No dynamic properties involved.

5.3.3.2 Location

The location properties are those related to the location of a graphic component, including left, top, right, bottom, width, and height. You can conFiguur the left, top, right, and bottom properties dynamic; no dynamic properties for width and height. During the on-site operation, the width and height of a graphic component adjust automatically when the left, top, right, and bottom properties change.

When configuring dynamic properties for the left, top, right, and bottom properties, make sure that the return value of the associated expression or dynamic script is numeric.

5.3.3.3 Visibility

The visibility property defines whether a graphic component is visible. You can conFiguur this property dynamic.

When configuring the dynamic visibility property, make sure that the return value of the associated expression or dynamic script is bit.

When the return value is 0 (FALSE), the graphic component is invisible; when it is not 0 (TRUE), the graphic component is visible.

5.3.3.4 Enability

The enability property is indicated by an enability flag. It defines whether to execute event scripts of a graphic component. You can conFiguur this property dynamic.

When configuring the dynamic enability property, make sure that the return value of the associated expression or dynamic script is bit.

When the return value is 0 (FALSE), the enability flag is off; when it is not 0 (TRUE), the enability flag is on.

5.3.3.5 Flashing

The are two flashing properties: flashing and flashing speed. The flashing speed property defines how fast the flashing goes; the smaller the value, the faster the flashing. Both properties can be conFiguurd dynamic.

When configuring the dynamic flashing property, make sure that the return value of the associated expression or dynamic script is bit. When the return value is 0 (FALSE), the graphic component does not flash; when it is not 0 (TRUE), the graphic component flashes.

When configuring the dynamic flashing speed property, make sure that the return value of the associated expression of dynamic script is numeric.

5.3.3.6 Redrawing

The redrawing property defines whether to automatically redraw a graphic component when properties of the graphic component change. For purposes of minimizing CPU utilization and improving system efficiency, graphic components are not set to be automatically redrew whenever the display of the graphic component is not affected. This property can be conFiguurd dynamic.

When configuring the dynamic property, make sure that the return value of the associated expression or dynamic script is bit.

When the return value is 0 (FALSE), the graphic component will be redrew automatically; when it is not 0 (TRUE), the graphic component will not be redrew

5.3.3.7 Positioning

There are two positioning properties: horizontal and vertical. They define the horizontal and vertical positioning of a graphic component in a window.

When these two properties are modified, the changes actually are reflected on the left and top properties. When you modify the left or top property without modifying the right or bottom property, the right or bottom property of the graphic component does not change; instead, the width or height of the graphic component changes automatically. However, different from modifying the left or top property, modifying the horizongtal or vertical positioning property does not change the width or height of a graphic component; instead, the right or bottom property of the graphic component changes automatcially.

When configuring the dynamic horizontal and vertical positioning properties, make sure that the return value of the associated expression of dynamic script is numeric.

5.3.4 Control-Specific Properties

5.3.4.1 Window

Property	Description	Remarks on Dynamic Properties
Title	Defines whether the window title is displayed. When you select **Display the title**, the title entered during the creating of a new interface window in **Project Manager** will be displayed as the window title.	No dynamic properties for windows.
Centering	Defines whether the window is centered in the screen.	
Background Color	Defines the background color of the window.	
Security Level	See Hoofdstuk 15 Access Management.	
Security Handling		

5.3.4.2 Rectangle

Property	Description	Remarks on Dynamic Properties
Frame Color	Sets the color of the frame of the rectangle.	The return value of the expression or dynamic script is an RGB value.
Filling Color	Sets the color of filling of the rectangle.	

5.3.4.3 Line

Property	Description	Remarks on Dynamic Properties
Color	Sets the color of lines.	The return value of the expression or dynamic script is an RGB value.
Line Width	Sets the width of lines.	The return value of the expression or dynamic script is int.
Direction	Sets the direction of lines; for example, from top left to bottom right or from bottom left to top right.	The return value of the expression or dynamic script is 1 or 2: ● 1: from top left to bottom right ● 2: from bottom left to top right
Type	Sets the type of lines; for example, solid lines, dotted lines, or dashed lines.	The return value of the expression or dynamic script is 1, 2 or 3: ● 1: solid lines ● 2: dotted lines ● 3: dashed lines

5.3.4.4 Oval

Property	Description	Remarks on Dynamic Properties
Frame color	Sets the color of the frame of the oval.	The return value of the expression or dynamic script is an RGB value.
Filling color	Sets the color of the filling of the oval.	

5.3.4.5 Text

Property	Description	Remarks on Dynamic Properties
Color	Sets the color of the text.	The return value of the expression or dynamic script is an RGB value.
Text	Sets the text content.	See section 5.3.2.2 Dynamic Text Properties.
Text Length	Defines the maximum text length (number of bytes).	No dynamic properties.
Alignment	Defines how the text is aligned.	The return value of the expression or

Property	Description	Remarks on Dynamic Properties
		dynamic script is 1, 2, or 3: ● 1: Left Alignment ● 2: Right Alignment ● 3: Center Alignment
Font Size	Defines the font size.	The return value of the expression or dynamic script is int: ● 0: Default font size ● Other: User-defined font size

5.3.4.6 Image

Property	Description	Remarks on Dynamic Properties
File Name	Defines the name of the image file. When you select an image file not stored in the project directory, the system automatically copies the file to the project directory. And then the image control automatically associates the copied image file. If you edit an original image file not stored in the project directory, the changes will not be automatically reflected in the image control. In this case, you need to manually add the editted image file.	The return value is a string with the name of the image file (file path not included). The image file must be stored in the project directory.
Loading Mode	Defines how the image is loaded. ● At startup: The image is loaded at the startup of the application. Once loaded, the image is kept in the memory. In this way, the image is loaded quite fast during the application operation. However, it reduces the speed of the application startup. Besides, it utilizes the memory considerably. ● During operation: The image is loaded during the application operation. In this way, the image is loaded into the memory only when it is necessary to display the image. In other words, the memory utilized by loading the image will be released when it is not necessary to display the image. This loading mode ensures faster application startup, and reduces the memory utilization. However, the speed for loading the image is very slow.	The return value is 1 or 2: ● 1: At startup ● 2: During operation
Transparency	Defines whether to make the image background transparent.	The return value is 0 or 1: ● 0: Not transparent ● 1: Transparent

Property	Description	Remarks on Dynamic Properties
Transparency Color	Defines the color of the transparent background.	The return value of the expression or dynamic script is an RGB value.

5.3.4.7 Incremental Rectangle

Property	Description	Remarks on Dynamic Properties
Start Color	Defines the start color for the incremental color change.	The return value of the expression or dynamic script is an RGB value.
End Color	Defines the end color for the incremental color change.	
Direction	Defines the direction of the incremental color change.	The return value of the expression or dynamic script is 32 or 47: ● 32: Horizontal > Center> Left and Right ● 47: Left > Bottom > Right > Top > Center

Drawing an incremental rectangle utilizes too much CPU resources. Therefore, it is not recommended to draw big incremental rectangles during configuration; for example, to draw an incremental rectangle which covers the whole window.

5.3.4.8 Incremental Oval

Property	Description	Remarks on Dynamic Properties
Start Color	Defines the start color for the incremental color change.	The return value of the expression or dynamic script is an RGB value.
End Color	Defines the end color for the incremental color change.	
Direction	Defines the direction of the incremental color change.	The return value of the expression or dynamic script is 0 or 2: ● 0: Horizontal > Center> Left and Right ● 2: Center > circumference

Drawing an incremental oval utilizes too much CPU resources. Therefore, it is not recommended to draw big incremental ovals during configuration; for example, to draw an incremental oval which covers the whole window.

5.3.4.9 Incremental Triangle

Property	Description	Remarks on Dynamic Properties
Start Color	Defines the start color for the incremental color change.	The return value of the expression or dynamic script is an RGB value.
End Color	Defines the end color for the incremental color change.	
Direction	Defines the direction of the incremental color change.	The return value of the expression or dynamic script is 1 or 4: ● 1: Upward ● 4: Towards the left

Drawing an incremental triangle utilizes too much CPU resources. Therefore, it is not recommended to draw big incremental ovals during configuration; for example, to draw an incremental triangle which covers the whole window.

5.3.4.10 Timer

Property	Description	Remarks on Dynamic Properties
Timing Interval	Defines the timing interval for the timer (unit: ms). When the scheduled time is reached, the system will execute the script defined in the **Timing** event. Note: The timer is timing constantly as long as the system is running. It has nothing to do with whether the window with the timer is displayed or not.	No dynamic properties.

5.3.4.11 Data Input

Property	Description	Remarks on Dynamic Properties
Variable Name	Defines the name of a database variable to which this graphic control is associated. This variable is numeric instead of string.	No dynamic properties.
Color	Defines the color in which the data is displayed.	The return value of the expression or dynamic script is the RGB value of the defined color.
Number of Integers	Defines the minimum number of integers to be displayed. ● When the number of integers is less than enough, 0 is added on the left to make enough number of integers.	The return value of the expression or dynamic script is int.

Property	Description	Remarks on Dynamic Properties
	● When the number of integers is more than required, only the specified number of integers will be displayed. ● If you define **Number of Integers** to **0**, all the actual integers will be displayed.	
Number of Decimals	The meaning of this property varies according to the type of the database variable to which the **Data Input** graphic control is associated. 1. When the variable type is float, this property defines a fixed number of decimals to be displayed. ● When the number of decimals is less than enough, 0 is added on the right to make enough number of decimals. ● When the number of decimals is more than required, only the specified number of decimals will be displayed. ● If you define **Number of Decimals** to **0**, only integers will be displayed; decimals not. When the variable type is int, this property means the same as the description for **Number of Integers**. For details, see the above cell.	The return value of the expression or dynamic script is int.
Minimum Value	Defines the minimum value of the data input.	The return value of the expression or dynamic script is numeric.
Maximum Value	Defines the maximum value of the data input.	The return value of the expression or dynamic script is numeric.
Prompt	Defines the prompt information to be displayed as the title of the **Data Input** window.	The return value of the expression or dynamic script is string.
Password Display	Defines how the password is displayed: ● If you set this property as **Password Display**, the password entered in the **Data Input** window will be displayed as a string of *****. ● Otherwise, the password entered is displayed as how it is.	The return value of the expression or dynamic script is bit.
Alignment	Defines how the data is aligned.	The return value of the expression or dynamic script is 1, 2 or 3: ● 1: Left Alignment ● 2: Right Alignment ● 3: Center Alignment
Font Size	Defines the font size.	The return value of the expression or dynamic script is int:

Property	Description	Remarks on Dynamic Properties
		● 0: Default font size ● Other ints: Defined font size
Integer Type	This property is valid only when the database variable to which the **Data Input** graphic control is associated to is int. The description below is only for int variables: ● If you select **Standard (no decimals)**: The **Number of Decimals** property becomes invalid. Any entered decimals will be discarded automatically. For example, if you enter **12.34**, the value of the associated int variable will be 12 and displayed as **12**. ● If you select **Decimals added automatically**: This property maps a entered float data (maybe with decimals) to an int variable. The mapping relationship is reflected on **Number of Decimals**, as shown by the equation below: Value of Variable = User Input * $10^{\text{Number of Decimals}}$ For example, when you set **Number of Decimals** to **2**, if you enter **12.34**, then the value of the associated int variable is 1234 (which is $12.34*10^2$), and displayed as **12.34**.	The return value of the expressionor dynamic script is 0 or 1: ● 0: Standard (no decimals) ● 1: Decimals added automatically

5.3.4.12 Text Input

Property	Description	Remarks on Dynamic Properties
Variable Name	Defines the name of a database variable to which this graphic control is associated. This variable is string.	No dynamic properties.
Color	Defines the color in which the text is displayed.	The return value of the expression or dynamic script is the RGB value of the defined color.
Prompt	Defines the prompt information to be displayed as the title of the **Text Input** window.	The return value of the expression or dynamic script is string.
Password Display	Defines how the password is displayed: ● If you set this property as password display, the password entered in the **Text Input** window will be displayed as a string of *. ● Otherwise, the password entered is displayed as how it is.	The return value of the expression or dynamic script is bit.
Alignment	Defines how the text is aligned.	The return value of the expression

Property	Description	Remarks on Dynamic Properties
		or dynamic script is 1, 2, or 3: ● 1: Left Alignment ● 2: Right Alignment ● 3: Center Alignment
Font Size	Defines the font size.	The return value of the expression or dynamic script is int: ● 0: Default font size ● Other ints: Defined font size

5.3.4.13 Button

Property	Description	Remarks on Dynamic Properties
Title	Defines the title of a button.	No dynamic properties.

5.3.4.14 Checkbox

Property	Description	Remarks on Dynamic Properties
Title	Defines the title of a checkbox.	No dynamic properties.
Associated Variable	Defines the database variable to which the **Checkbox** graphic control is associated. This variable is int. When the checkbox is checked, the value of the associated variable becomes **1**; otherwise, it is **0**. Accordingly, if you set the value of the associated variable to **1**, the checkbox will be checked; if you set it to **0**, the checkbox will not be checked.	The return value of the expression or dynamic script is string. The string refers to the name of the variable to which the **Checkbox** graphic control is associated.

5.3.4.15 Combo Box

Property	Description	Remarks on Dynamic Properties
Data in the Drop-Down List	Defines the data to be displayed in the drop-down list of a combo box. A semi-colon **;** is used between the data; for example, **aaa;bbb;ccc**.	No dynamic properties.
Associated Variable	Defines the database variable to which the **Combo Box** graphic control is associated. This variable is int. The value of the assocated variable refers to the serial number of the data in the drop-down list. The serial number starts from 0, followed by 1, 2… from top to	The return value of the expression or dynamic script is string. The string refers to the name of the variable to which the **Combo Box** graphic control is associated.

Property	Description	Remarks on Dynamic Properties
	bottom. For example: ● When you select the first data in the drop-down list of a combo box, the value of the associated variable becomes **0**; ● When you select the second data, the value of the associated variable becomes 1; ● And so on and so forth.	
Height of Data in the Drop-Down List	Defines the height between any two data in the drop-down list.	No dynamic properties.

5.3.4.16 Vector Text

Different from the **Text** control, the **Vector Text** control provides more fontsyou're your selection.

Property	Description	Remarks on Dynamic Properties
Font	Defines the font for the text.	No dynamic properties.
Text	Defines the text content.	No dynamic properties.
Alignment	Defines how the text is aligned.	The return value of the expression or dynamic script is 1, 2, or 3. ● 1: Left Alignment ● 2: Right Alignment ● 3: Center Alignment

5.3.4.17 Alarm Window

See Hoofdstuk 10 Alarms.

5.3.4.18 Real-Time Trend

See Section 8.2 Real-Time Data Records.

5.3.4.19 Historical Trend

See Section 9.2 Historical Data Records.

5.3.4.20 Historical Data List

See Section 9.2 Historical Data Records.

5.3.4.21 Graphics Library

See Hoofdstuk 18 Gallery Controls.

5.3.5 Events

Based on categories of graphic control components, there are the following types of events, as described in the following table.

Graphic Component	Event	Trigger Condition
Window	Open	When you open a window
	Close	When you close or hide a window
Timer	Timing	When the scheduled time is reached
Other Controls	Press	When you press the left button of the mouse on a graphic control
	Release	When you release the left button of the mouse on a graphic control

5.4 System Variables for the Interface

Database Name	Variable Name	Data Type	Default Value	Description
System	HmiLoopCount	ulong		The count value of the interface refresh. The value of this variable adds 1 every time the interface refreshes.
	HideMainWindow	bit	The default value of this variable is: ● 0: Simulated operation on PC ● 1: Operation on HMI	● 1: Menus on the specified Windows window are hidden. ● 0: Menus on the specified Windows window are displayed.
	HmiHeartbeat	bit		The heart beat of the HMI during operation. The value of this variable changes between 0 and 1 during the interface refresh.
	HmidbDefCycleTime	ulong	500	The cycle time for the interface refresh (unit: ms).

5.5 System Functions for the Interface

5.5.1 hmi_window_show

Original Function: int **hmi_window_show**(**char** **window_name*)

Function Description: To show a specified window.

Return Values: 0 Failed
1 Successful

Parameter: *window_name*: Name of the target window you want to display.

Example: hmi_window_show("test")

5.5.2 hmi_window_hide

Original Function: int **hmi_window_hide**(**char** **window_name*)

Function Description: To close a specified window.

Return Values: 0 Failed
1 Successful

Parameter: *window_name*: Name of the target window you want to close.

Example: hmi_window_hide("test")

5.5.3 hmi_window_show_modal

Original Function: int **hmi_window_show_modal**(**char** **window_name*)

Function Description: To display a modal dialog box.

Return Values: 0 Failed
1 Successful

Parameter: *window_name*: Name of a window.

Example: hmi_window_show_modal("test")

5.5.4 hmi_window_exit_modal

Original Function: int **hmi_window_exit_modal**(**char** **window_name*)

Function Description: To exit the **Modal** dialog box. Call this function when you want to exit a modal dialog box.

Return Values: 0 Failed
1 Successful

Parameter: *window_name*: Name of a window.

Example: hmi_window_exit_modal("test")

5.5.5 data_input_window

Original Function: int **data_input_window**(**char** ***varname*, **char** ***caption*, **double** *minvalue,* **double** *maxvalue*, **int** *dec_num*)

Function Description: Function for data input. When you call this function, the **Data Input** window will be displayed. The data you enter in this window will be assigned as the value for the parameter ***varname***.

Return Value: 0 Failed

1 Successful

Parameters: *varname*: Name of a variable. The data you enter in the **Data Input** window will be assigned as the value for this parameter.

caption: Prompt information to be displayed as the title of the **Data Input** window.

minvalue: Minimum value allowed for the data entered.

maxvalue: Maximum value allowed for the data entered.

dec_num: Number of decimals.

Example: data_input_window("test.data", "test", 0, 100, 2)

5.5.6 data_input_window_pwd

Original Function: int **data_input_window_pwd**(**char** ***varname*, **char** ***caption*, **double** *minvalue,* **double** *maxvalue*, **int** *dec_num,* **int** *passwd*)

Function Description: Function for data input (password display option available). When you call this function, the **Data Input** window will be displayed. The data you enter in this window will be assigned as the value for the parameter ***varname***.

Return Value: 0 Failed

1 Successful

Parameters: *varname*: Name of a variable. The data you enter in the **Data Input** window will be assigned as the value for this parameter.

caption: Prompt information to be displayed as the title of the **Data Input** window.

minvalue: Minimum value allowed for the data entered.

maxvalue: Maximum value allowed for the data entered.

dec_num: Number of decimals.

passwd: 1: Password Display; 0: Normal Display.

Example: data_input_window_pwd("test.data", "test", 0, 100, 2, 1)

5.5.7 text_input_window

Original Function: int **text_input_window**(**char ****varname*, **char ****caption,* **int** *passwd*)

Function Description: Function for text input. When you call this function, the **Text Input** window will be displayed. The text you enter in this window will be assigned as the value for the parameter ***varname***.

Return Values: 0 Failed
1 Successful

Parameters: *varname*: Name of a variable. The text you enter in the **Text Input** window will be assigned as the value for this parameter.

caption: Prompt information to be displayed as the title of the **Text Input** window.

passwd: 1: Password Display; 0: Normal Display.

Example: text_input_window("test.data", "test", 1)

5.5.8 msgbox

Original Function: int **msgbox**(**char ****caption***, char ****text***, int** *type*)

Function Description: Function for displaying a message box.

Return Values: MSG_IDOK: You can click on the **OK** button.
MSG_IDCANCEL: You can click on the **Cancel** button.
MSG_IDABORT: You can click on the **Abort** button.
MSG_IDRETRY: You can click on the **Retry** button.
MSG_IDIGNORE: You can click on the **Ignore** button.
MSG_IDYES: You can click on the **Yes** button.
MSG_IDNO: You can click on the **No** button.

Parameters: *caption*: Title of a window.

text: Message content.

type: Type of a message box, valued as follows:

MSG_MB_OK: The **OK** button is displayed.

MSG_MB_OKCANCEL: The **OK** and **Cancel** buttons are displayed.

MSG_MB_YESNO: The **Yes** and **No** buttons are displayed.

MSG_MB_RETRYCANCEL: The **Retry** and **Cancel** buttons are displayed.

MSG_MB_ABORTRETRYIGNORE: The **Abort**, **Retry** and **Ignore** buttons are displayed.

MSG_MB_YESNOCANCEL: The **Yes**, **No** and **Cancel** buttons are displayed.

MSG_MB_ICONSTOP: The **Stop** icon is displayed.

MSG_MB_ICONQUESTION: The **Question** icon is displayed.

MSG_MB_ICONEXCLAMATION: The **Exclamation** icon is displayed.

MSG_MB_ICONINFORMATION: The **Information** icon is displayed.

MSG_MB_DEFBUTTON1: The first button is defined as default.

MSG_MB_DEFBUTTON2: The second button is defined as default.

MSG_MB_DEFBUTTON3: The third button is defined as default.

Example: msgbox("Error", "Open file failed", MSG_MB_OK)

5.5.9 hmi_center_window

Original Function: int **hmi_center_window**(**char** **window_name*)

Function Description: Function for displaying the window in the center.

Return Values: 0 Failed

1 Successful

Parameter: *window_name*: Name of a window.

Example: hmi_center_window("test")

Chapter 6 Parameters

6.1 Overview

Parameter configuration provides the following two functions:

1) Modifying the default values of system variables, so as to change the system behaviours;
2) Saving the user-defined data, the values of which are still stored in the system even after you restart the system.

The following two sections describe these two functions in details.

6.1.1 Modifying System Variables

EASY defines some system variables to control the system operation. For example, the system variable **$system. HmidbDefCycleTime** defines the cycle time for interface refresh. Each system variable has a factory default value. You can modify the initial values of these system variables to satisfy some special needs.

To modify the default value of a system variable, you need to add a parameter in parameter configuration and associate it with the system variable. For details about adding a parameter, please see section 6.2 Adding a Parameter.

6.1.2 Saving User-Defined Data

In actual operation, you might need to modify some data and have them saved in the system permanently, so that they stay in the system even after you shut down or restart the system; for example, some key configuration data. Different from system variables, you can define these data in the real-time database.

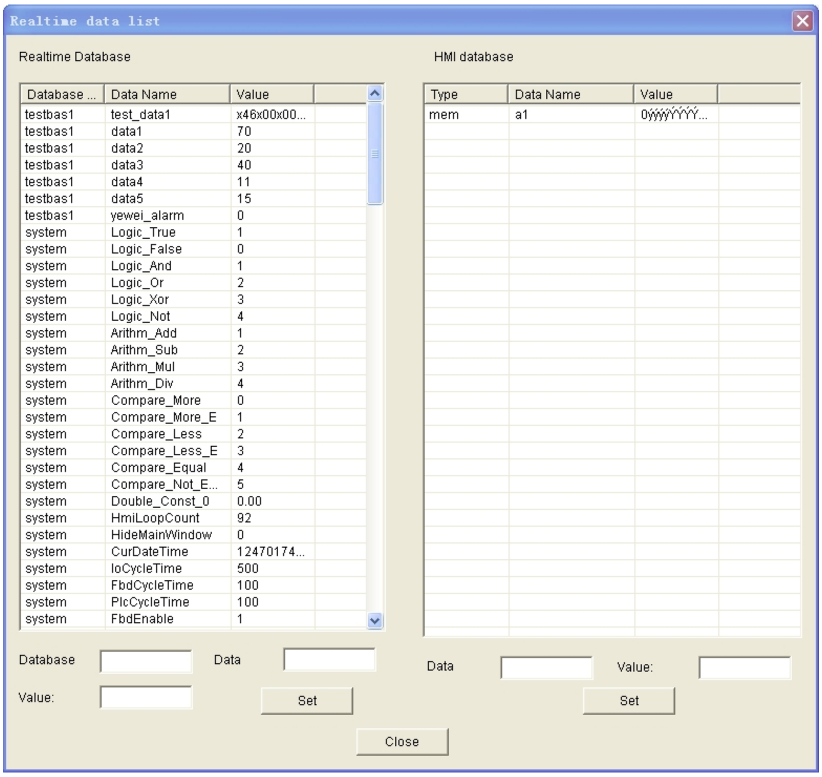
For saving these user-defined data, EASY provides the following solution:

1. Add a parameter in parameter configuration, and associate this parameter with the real-time data you want to save in the system permanently.

   For details about adding a parameter, please see section 6.2 Adding a Parameter.

   Every time when a new parameter is added, the system will add an internal data in the HMI database. These internal data are generated by the system automatically, and are not reflected on the configuration interface. However, you can view them in the **HMI Database** list on the right side of the **Real-Time Data**

Monitoring window during offline or online simulation, as shown in Figuur 6.1.



Realtime Database

Database ...	Data Name	Value
testbas1	test_data1	x46x00x00...
testbas1	data1	70
testbas1	data2	20
testbas1	data3	40
testbas1	data4	11
testbas1	data5	15
testbas1	yewei_alarm	0
system	Logic_True	1
system	Logic_False	0
system	Logic_And	1
system	Logic_Or	2
system	Logic_Xor	3
system	Logic_Not	4
system	Arithm_Add	1
system	Arithm_Sub	2
system	Arithm_Mul	3
system	Arithm_Div	4
system	Compare_More	0
system	Compare_More_E	1
system	Compare_Less	2
system	Compare_Less_E	3
system	Compare_Equal	4
system	Compare_Not_E...	5
system	Double_Const_0	0.00
system	HmiLoopCount	92
system	HideMainWindow	0
system	CurDateTime	12470174...
system	IoCycleTime	500
system	FbdCycleTime	100
system	PlcCycleTime	100
system	FbdEnable	1

HMI database

Type	Data Name	Value
mem	a1	0ýýýýÝÝÝ...

Figuur 6.1

At the system startup, the user-defined real-time data which are defined in parameters will be copied to the corresponding HMI memory automatically.

The reason why the system automatically generates the data in the HMI memory is mainly for the consideration of users' potential needs of "cancellation". For example, if you modify a parameter on the interface, the corresponding data in the real-time database will be modified accordingly; however, the corresponding data in the HMI memory remains the same as before your modification. If you feel like cancelling the modification, you can call the **rtdb_param_mem_to_rtdb** function to copy the data from the HMI memory to the real-time database.

2. To save a data after you modify it, you can call the **sys_save_param** function to save the data to a device.

   To prepare for potential future cancellation, you can call the **rtdb_param_rtdb_to_mem** function to copy the modified data to the HMI memory, so as to keep the data in the real-time database and that in the HMI memory consistent, ready for the next time cancellation.

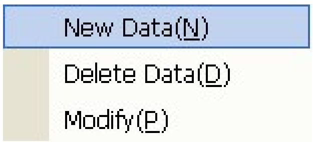
3. To cancel the data modification, you can call the **rtdb_param_mem_to_rtdb** function to copy the data automatically generated in the HMI memory to the real-time database.

6.2 Adding a Parameter

Each added parameter must be associated with a variable, either a system variable defined internally in the system, or the user-defined data defined in the real-time database. If a parameter is associated with a system variable, the parameter is added to modify the default value of the system variable (see sction 6.1.1 Modifying System Variables for details). If a parameter is associated with the user-defined real-time data, the parameter is added to store the real-time data permanently in the system (see section 6.1.2 Saving User-Defined Data for details).

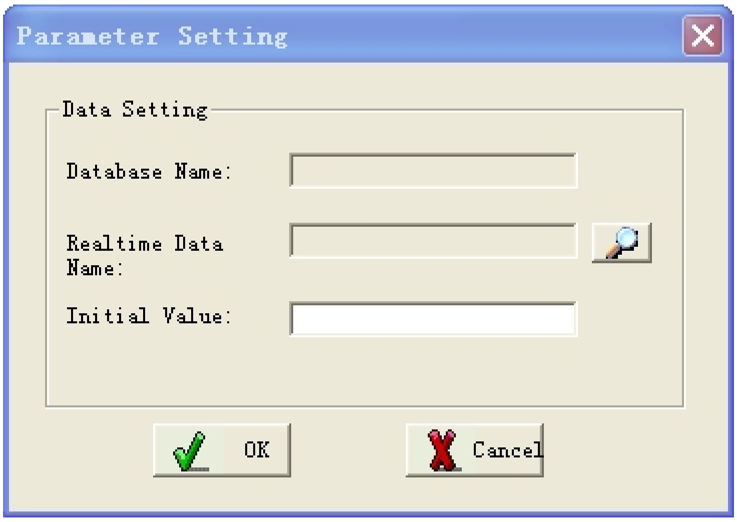
To add a parameter, do as follows:

1) Select **Parameter** on the left side of the **Project Manager** window, and then right-click in the list pane on the right side of the window, and you'll see a right-click menu as shown in Figuur 6.2.



Figuur 6.2

2) Select **Add Data**, and you'll see a dialog box as shown in Figuur 6.3.



Figuur 6.3

The configuration parameters are described as follows:

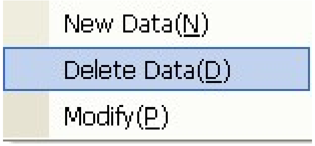
- **Database Name**: Name of the database which stores the variable to which the parameter is associated.
- **Real-Time Data Name**: Name of the variable to which the parameter is associated.

- **Initial Value**: Initial value of the variable to which the parameter is associated. The initial value is 0 if no specific initial value is conFiguurd here.
  Note: If the initial value conFiguurd in the real-time database is different from the initial value here, the system uses the initial value conFiguurd here.

6.3 Deleting a Parameter

To delete a parameter, do as follows:

1) Select **Parameter** on the left side of the **Project Manager** window, and then right-click on the parameter you want to delete in the list pane on the right side of the window.
   You'll see a right-click menu as shown in Figuur 6.4.



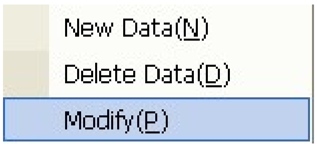
Figuur 6.4

2) Select **Delete Data**, and the selected parameter will be deleted.

6.4 Modifying a Parameter

To modify a parameter, do as follows:

1) Select **Parameter** on the left side of the **Project Manager** window, and then right-click on the parameter which you want to modify in the list pane on the right side of the window.
   You'll see a right-click menu as shown in Figuur 6.5.



Figuur 6.5

2) Select **Modify** to modify the parameter according to your needs.

6.5 System Functions for Parameter Configuration

6.5.1 rtdb_param_mem_to_rtdb

Original Function: int **rtdb_param_mem_to_rtdb**()

Function Description: To copy the data from the HMI memory to the real-time database as the value of parameters.

Return Value: 0 Failed

1 Successful

Parameter: None

Example: rtdb_param_mem_to_rtdb()

6.5.2 rtdb_param_rtdb_to_mem

Original Function: int **rtdb_param_rtdb_to_mem**()

Function Description: To copy the value of parameters from the real-time database to the HMI memory.

Return Value: 0 Failed

1 Successful

Parameter: None

Example: rtdb_param_rtdb_to_mem()

6.5.3 sys_save_param

Original Function: int **sys_save_param**()

Function Description: To save parameter data to devices.

Return Value: 0 Failed

1 Successful

Parameter: None

Example: sys_save_param()

Chapter 7 C Language Programming

7.1 Overview

EASY supports powerful C language programming. Considering that all user-defined scripts in the system are based on the C language, you can make full use of the powerful, flexible and highly efficient C language programming to achieve complicated applications most of which are possible only in the Industrial Personal Computer (IPC).

At present, EASY supports the C language programming in the following aspects:

- Direct C language programming in the scripts for dynamic properties and events for graphic components
- Support of user-defined external C-language source files and library files
- Support of user-defined external expansion modules (which allows you to use your own communication protocols) (see Hoofdstuk 14 Expansion Module Programming for details)

7.2 Script Programming

During the configuration, you can conFiguur dynamic property and event scripts for graphic components. All of these scripts for dynamic properties and events are compiled based on the C language.

In these scripts, you can:

- Use all the rules defined according to the C language syntax; for example, to define static variables or array variables, or to use the ? expressions in the scripts.
- Directly call the standard function library provided by the C language; for example, to call the functions such as **strcmp**, **sprintf**, **malloc**, or **fopen**.
- Directly access the data defined in the real-time database and interface database. For accessing the data in the real-time database, see section 3.5 Data Referencing for details. For accessing the data in the interface database, see section 4.4 Data Referencing for details.
- Call the internal system functions provided by the EASY system; for example, the **hmi_window_show** function.
- Directly call the user-defined functions defined in external C-Language source files and library files. For details, see section 7.3 External C Language Source

Files and Library Files.

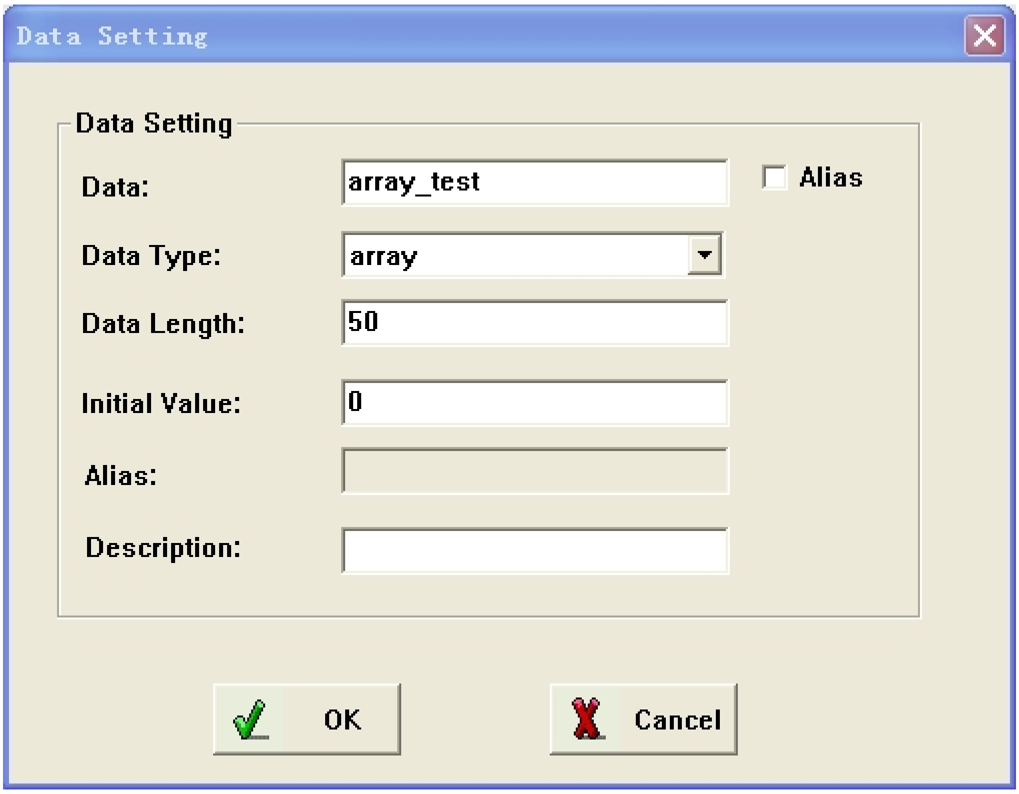
Programming in the scripts in EASY is not much different from the usual C language programming. Just pay attention to the usage of the following two aspects:

- Array Variables
- **$$HmiCommit** and **$$HmiReload**

7.2.1Array Variables

EASY provides array variables, which are similar to the arrays in the C language. The difference lies only in the data type: The C language supports defining arrays for various types of data; for example, you can define int arrays (for example, int array[10]) or float arrays (for example, float array[20]). However, the array variables in EASY are all uchar. Therefore, while defining array variables in the real-time database, consider the data length as the number of bytes.

The following Figuur shows an example of defining an array variable with the name as **array_test** in the real-time database. The data length is **50**, as shown in Figuur 7.1.



Data Setting

Data Setting

Data: array_test

□ Alias

Data Type: array

Data Length: 50

Initial Value: 0

Alias:

Description:

Figuur 7.1

You can consider this example as defining the array **unsigned char array_test[50]** in the C language.

After defining array variables, you can operate on the array variables in the real-time database as how you operate on the arrays of the C language. For example, you can carray out the following operation (supposing that the database name is **test**):

```
$test.array_test[2] = 5;
memcpy($test.array_test, "\x10\x11\x12\x13\x14", 5);
*((int*)($test.array_test + 4)) = 0x1234;
```

Note: At present, the array variables can only be used in interface configuration scripts and external C language applications.

7.2.2 **$$HmiCommit** and **$$HmiReload**

7.2.2.1 Processing the Script

EASY allows you to conFiguur dynamic property and event scripts for graphic components. The EASY system process these scripts as follows:

1. Before executing a user-compiled script, the system first scans the script for the data referenced from the real-time database and the interface database (namely, the variables starting with **$**), obtains the values of these data from the database and save them to temporary variables, and then replaces all the referenced data with the corresponding temporary variables.
2. The system executes the user-defined script.
3. The system determines whether the user-defined script changes the values of the database variables. If yes, the system writes the new values into the database.

For example, the following is a user-defined script:

```
$test.int_var1++;
$test.float_var1 = 10.5;
```

The system processing of the script above can be translated into the following pseudo code (**get_data_from_database** and **set_data_to_database** are pseudo codes: **get_data_from_database** means to read data from the database, and **set_data_to_database** means to write data into the database):

```
int tmp_var1 = get_data_from_database("test.int_var1");
int old_tmp_var1 = tmp_var1;
float tmp_var2 = get_data_from_database("test.float_var1");
float old_tmp_var2 = tmp_var2;

tmp_var1++;
tmp_var2 = 10.5;

IF (tmp_var1 != old_tmp_var1)
```

```
        set_data_to_database("test.int_var1", tmp_var1);
    ENDIF
    IF (tmp_var2 != old_tmp_var2)
        set_data_to_database("test.float_var1", tmp_var2);
    ENDIF
```

The example above shows that the system accesses the database only at the beginning (obtaining the values of the referenced variables) and the end (saving the modified values into the database) of the script, but not the whole process of executing the script. This, to some extent, ensures the system efficiency, saving the trouble of accessing the database after executing each line of the script.

Generally speaking, this way of script processing ensures high system efficiency. However, it also creates some problems; for example, the data values changed by the execution of the script are not written into the database immediately, but only at the end of the script.

For solving this kind of problems, EASY provides the **$$HmiCommit** and **$$HmiReload** functions.

7.2.2.2 $$HmiReload

Call the **$$HmiReload** function to reaccess the database for the values of all the data referenced in the script and then assign these values to the variables currently used in the script (actually the temporary variables).

Take the following script for example:

```
data_input_window("test.float_var1", "test", 0, 100, 2);
if ($test.float_var1 > 50.0)
{
    $test.int_var1 = 1;
}
else
{
    $test.int_var1 = 0;
}
```

In which, **data_input_window** is a data input function provided by the EASY system. After calling this function, you will see the **Data Input** window where you can enter data. In this example, the data you enter will be assigned to the database variable **test.float_var1**.

The system processing of the script above can be translated into the pseudo codes as follows:

```
int tmp_var1 = get_data_from_database("test.int_var1");
int old_tmp_var1 = tmp_var1;
float tmp_var2 = get_data_from_database("test.float_var1");
float old_tmp_var2 = tmp_var2;

data_input_window("test.float_var1", "test", 0, 100, 2); /* Note: This function operates on the database by directly writing the value you enter to the database variable test.float_var1 */
if (tmp_var2 > 50.0)   /*Here tmp_var2 is still used as the value of the variable test.float_var1; however, the value of the database variable test.float_var1 has been modified by the data_input_window function. The value assignment here is obviously wrong*/
{
    tmp_var1 = 1;
}
else
{
    tmp_var1 = 0;
}

IF (tmp_var1 != old_tmp_var1)
    set_data_to_database("test.int_var1", tmp_var1);
ENDIF
IF (tmp_var2 != old_tmp_var2)
    set_data_to_database("test.float_var1", tmp_var2);
ENDIF
```

In the pseudo codes above displayed in bold, the **data_input_window** function writes the data entered directly to the database variable **test.float_var1**. However, the IF condition followed immediately **if (tmp_var2 > 50.0)** doesn’t reaccess the database to obtain the modified value of the variable **test.float_var1**, but instead uses **tmp_var2** (value obtained from the database at the beginning of the script) as the value of the variable. Obviously, the value assignment is wrong. In this case, you can call the **$$HmiReload** function to reaccess the database forcibly.

The modified script is as follows:

```
data_input_window("test.float_var1", "test", 0, 100, 2);
$$HmiReload;
if ($test.float_var1 > 50.0)
{
      $test.int_var1 = 1;
}
else
{
      $test.int_var1 = 0;
}
```

The system processing of the above modified script is translated into the following pseudo codes:

```
int tmp_var1 = get_data_from_database("test.int_var1");
int old_tmp_var1 = tmp_var1;
float tmp_var2 = get_data_from_database("test.float_var1");
float old_tmp_var2 = tmp_var2;
```

data_input_window("test.float_var1", "test", 0, 100, 2); /* Note: This function operates on the database by directly writing the value you enter to the database variable test.float_var1 */

tmp_var1 = get_data_from_database("test.int_var1");
tmp_var2 = get_data_from_database("test.float_var1");

if (tmp_var2 > 50.0) /*Here tmp_var2 is the modified value of the database variable test.float_var1 after the data_input_window function is called*/

```
{
      tmp_var1 = 1;
}
else
{
      tmp_var1 = 0;
}

IF (tmp_var1 != old_tmp_var1)
```

```
                set_data_to_database("test.int_var1", tmp_var1);
        ENDIF
        IF (tmp_var2 != old_tmp_var2)
                set_data_to_database("test.float_var1", tmp_var2);
        ENDIF
```

In the pseudo codes above, the part in italic indicates that the **$$HmiReload** function is called, which means that the system reaccesses the database to obtain the values of the variables **test.int_var1** and **test.float_var1** and assign them accordingly to **tmp_var1** and **tmp_var2**. In this way, **tmp_var2** matches correctly to the modified value of the variable **test.float_var1**, thus the further value assignment of the variable **test.float_var1** would be correct.

7.2.2.3 $$HmiCommit

Call the **$$HmiCommit** function to write the current values of all the referenced data in to the database.

Take the following script for example:

```
$test.float_var1 = 20.5;
user_func();
```

In which, **user_func** is a user-defined function. It can be defined as follows:

```
void user_func()
{
        ……
        if ((*(float *)hmidb_get_data_value("test.float_var1")) >= 20.0)
        {
                ……
        }
        ……
}
```

In which, **hmidb_get_data_value** is a system function provided by EASY. This function is for obtaining the data value from the database. The system processing of the script above can be translated in the following pseudo codes:

```
float tmp_var1 = get_data_from_database("test.float_var1");
float old_tmp_var1 = tmp_var1;
```

```
tmp_var1 = 20.5;
user_func();

IF (tmp_var1 != old_tmp_var1)
    set_data_to_database("test.float_var1", tmp_var1);
ENDIF
```

Let’s take a look at the bold codes in the pseudo codes above. The script is supposed to first set the value of the variable **test.float_var1** to **20.5**, and then call the function **user_func**, and further the function **hmidb_get_data_value** to obtain the current value (which should be **20.5**) of the variable **test.float_var1**, and then process it accordingly based on the value obtained.

But from the pseudo codes above, we can see that **$test.float_var1 = 20.5;** is replaced by **tmp_var1 = 20.5;** in the script. During the system processing, the value of **tmp_var1** is modified; however, the modified value is not written into the database immediately (as stated ealier, the modified value is written into the database only at the end of the script). Therefore, the value of **test.float_var1** in the database is still the value before the modification. The value of **test.float_var1** obtained by the function **hmidb_get_data_value** of the function **user_func** is also the value before the monidication, but not the user-defined value **20.5**.

To adres this issue, you can call the function **$$HmiCommit** to force the system to write the data into the database immediately.

The modified script is as follows:

```
$test.float_var1 = 20.5;
$$HmiCommit;
user_func();
```

The system processing of the modified script can be translated into the following pseudo codes:

```
float tmp_var1 = get_data_from_database("test.float_var1");
float old_tmp_var1 = tmp_var1;

tmp_var1 = 20.5;

        set_data_from_database("test.float_var1", tmp_var1);user_func();
```

```
IF (tmp_var1 != old_tmp_var1)
      set_data_to_database("test.float_var1", tmp_var1);
ENDIF
```

The italic codes above is the result of calling the function **$$HmiCommit**. It forces the system to immediately write the value of **tmp_var1** into the database data **test.float_var1**. In this case, when the function **user_func** accesses the database for the value of **test.float_var1**, it obtains the latest value.

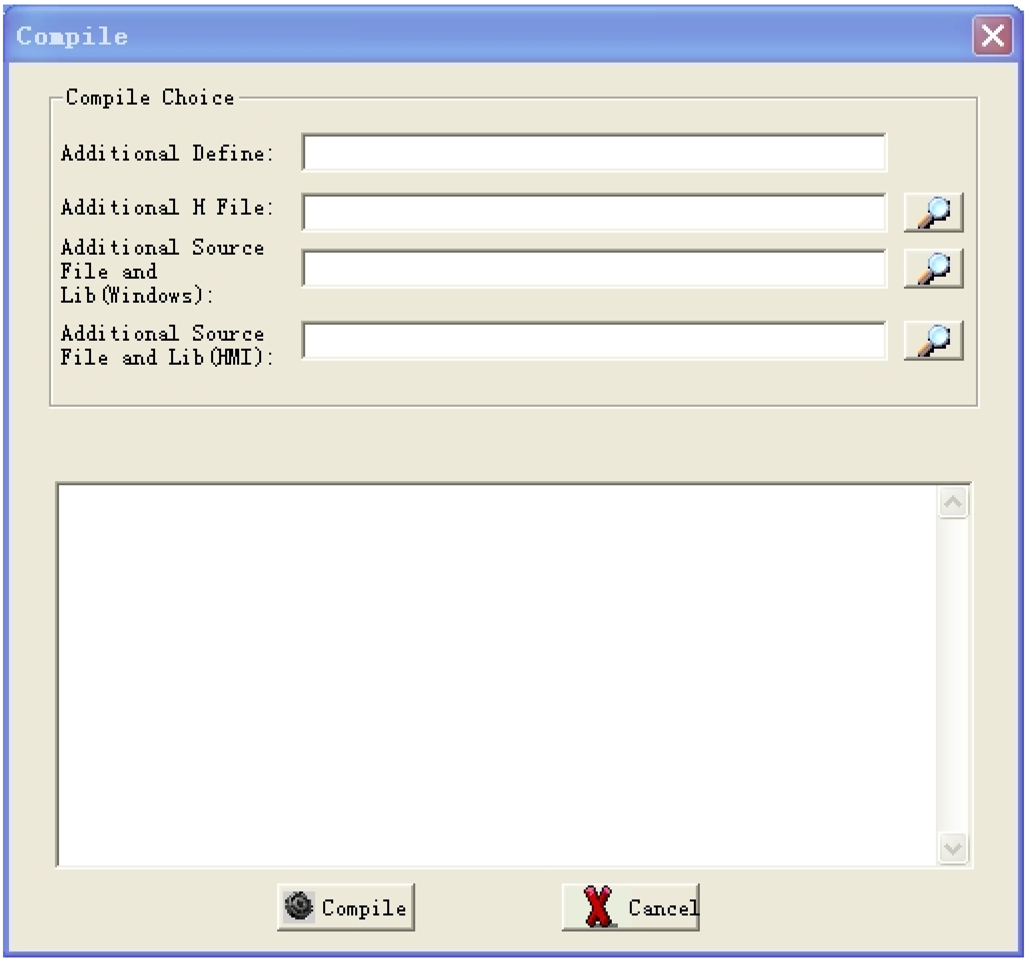
7.3 External C Language Source Files and Library Files

7.3.1Overview

You can define one or more complicated algorithms or common functions into one or more external C source or library files. You just simply assign these external C source or library files during the compiling, and EASY can integrate these files into the system, so that they can be called directy by the dynamic property and event scripts of graphic components during interface configuration.

To assign the external C source or library files during the compiling, do as follows:

In the **Project Manager** window, select **Tools** and then **Compile a Project**, or simply click on the compiling button ⚙ in the tool bar, and you will see the **Compile a Project** dialog box as shown in Figuur 7.2.


Compile

Compile Choice

Additional Define:

Additional H File:

Additional Source File and Lib(Windows):

Additional Source File and Lib(HMI):

Figuur 7.2

If no external C source or library files are used in the project, then you do not need to conFiguur any of the configuration items in **Compiling Options**. If they are used, then you need to conFiguur one of more of the configuration items, which are described as follows:

- **Additional Definition**: Equal to the compiling options in the C language; for example, **-D_DEBUG –D_MYDEF –Ic:\myinclude**.
- **Additional Header File**: To define the header file to be used in the project. Only one header file can be added.
- **Additoinal Source files and libraries (Windows)**: To define the external C source or library files to be added for running on the Windows platform (for both offline and online simulation). Multiple files can be added at the same time.
- **Additoinal Source files and libraries** (**HMI**): To define the external C source or library files to be added for running on the HMI. Multiple files can be aded at the same time.

7.3.2Examples

This section uses an example to explain how to use external C source or library files during the EASY configuration.

This example shows the following function: to implement the BubbleSort (BS) algorithm in the external C source file, and then call this algorithm during the HMI configuration to sort the data in the real-time database.

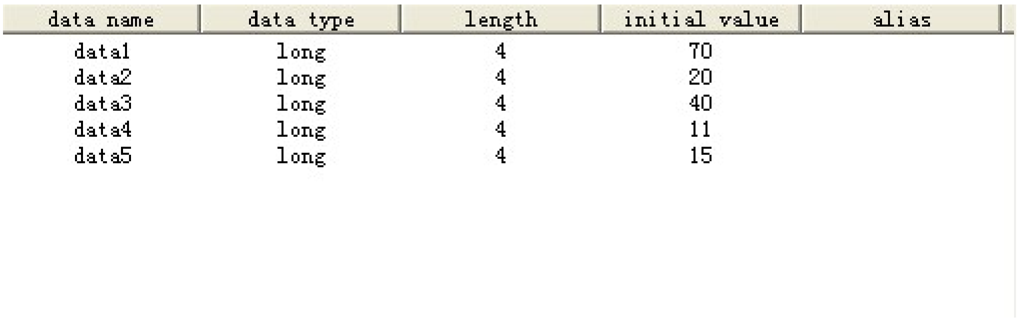
1. Create a C source file **sort.c**, and implement the BS algorithm on the long and char data arrays. The contents of the source file go as follows:

```
void   bubble_sort( long array[], int length)
{
    unsigned char exchange; /* To record whether the element exchange occurs in the first round traversal */
    long temp;
    int i, j;
    for (i = 1; i < length; i++)
    {
        exchange = 0 ;
        for (j = length-1; j >= i; j--)
        {
            if (array[j] < array[j-1])
            {
                exchange = 1;
                temp = array[j];
                array[j] = array[j-1];
                array[j-1] = temp;
            }
        }
        /* The sorting ends if no element exchange in this round of traversal. */
        if   ( 0   == exchange)
            break ;
    }
}
```

2. Add an external C header file **sort.h**, which contents go as follows:

```
#ifndef _SORT_H_
#define _SORT_H_

extern void bubble_sort(long array[], int length);

#endif
```

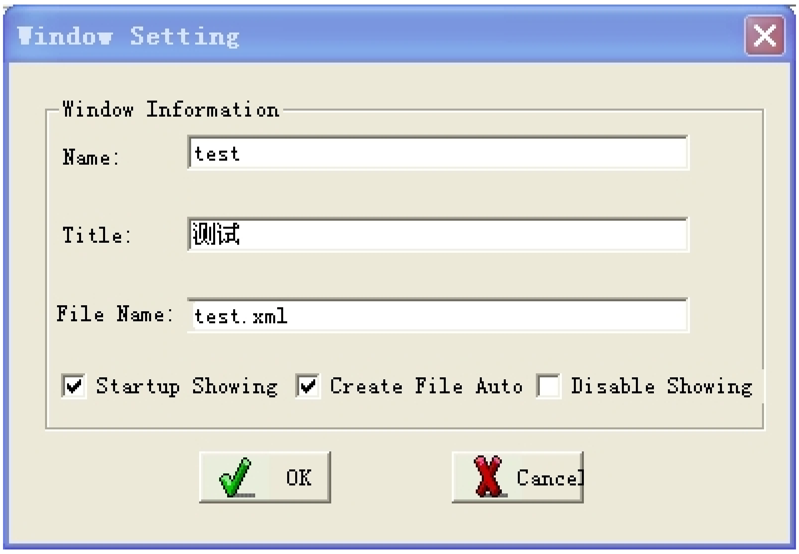
3. In the **Project Manager** window, add a new database **test** under the **Real-Time Database** node, and then add 5 long data, **data1** to **data5**, with the initial values as 70, 20, 40, 11, and 15 individually. After you add all the 5 data, you will see a window as shown in Figuur 7.3.



data name	data type	length	initial value	alias
data1	long	4	70	
data2	long	4	20	
data3	long	4	40	
data4	long	4	11	
data5	long	4	15	

Figuur 7.3

4. In the **Project Manager** window, add an interface **test**, as shown in Figuur 7.4.



Window Setting

Window Information

Name: test

Title: 测试

File Name: test.xml

☑ Startup Showing ☑ Create File Auto □ Disable Showing

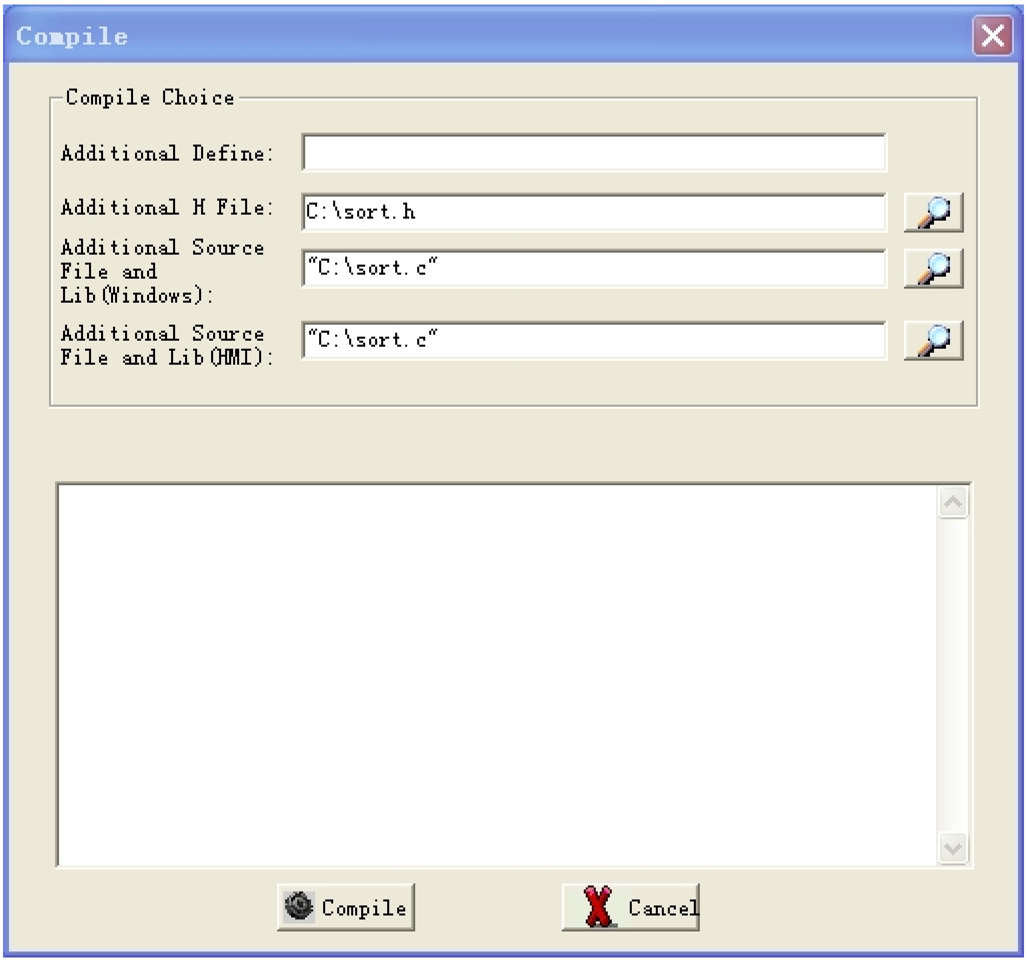
Figuur 7.4

5. In the **test** interface, add a button graphic component, and then compile the following script for the **Press** event for this button:

```
long array[5];

array[0] = $test.data1;
array[1] = $test.data2;
array[2] = $test.data3;
array[3] = $test.data4;
array[4] = $test.data5;

bubble_sort(array, 5);

$test.data1 = array[0];
$test.data2 = array[1];
$test.data3 = array[2];
$test.data4 = array[3];
$test.data5 = array[4];
```

6. In the **Project Manager** window, select **Tools** and then **Compile a Project**, or click on the compiling button ⚙ in the tool bar, and you'll see a dialog box as shown in Figuur 7.5.

   You need to conFiguur the four configuration items accordingly. (Suppose that the **sort.c** and **sort.h** files are kept in the **C:\** directory).



Additional Define:

Additional H File: C:\sort.h

Additional Source File and Lib(Windows): "C:\sort.c"

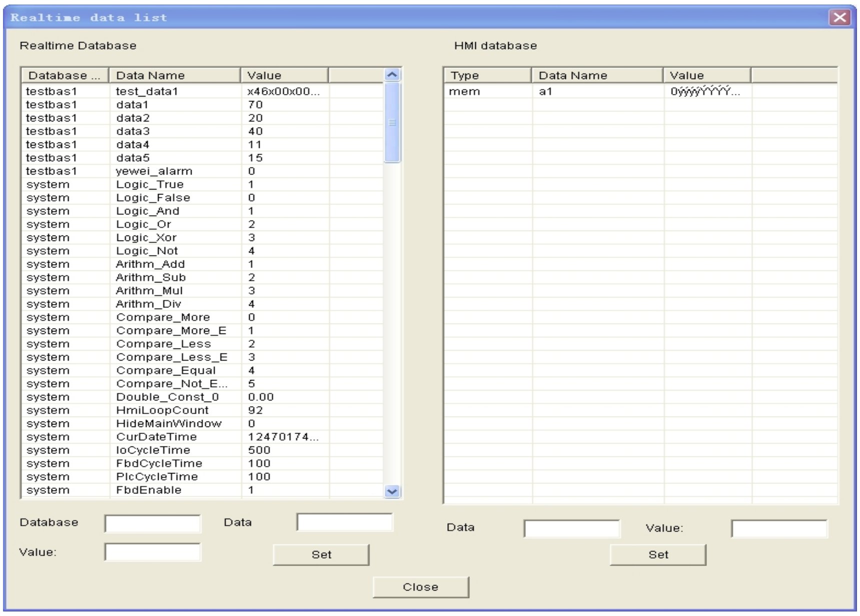
Additional Source File and Lib(HMI): "C:\sort.c"

Figuur 7.5

Click on the **Start Compiling** button.

7. Select **Tools** and **Offline Simulation** to show the configuration interface. Select **Window** and then **Real-Time Display** to display the real-time data monitoring window, as shown in Figuur 7.6.

   In this window, you can see the values of data1 to data5 are 70, 20, 40, 11, 15 accordingly:

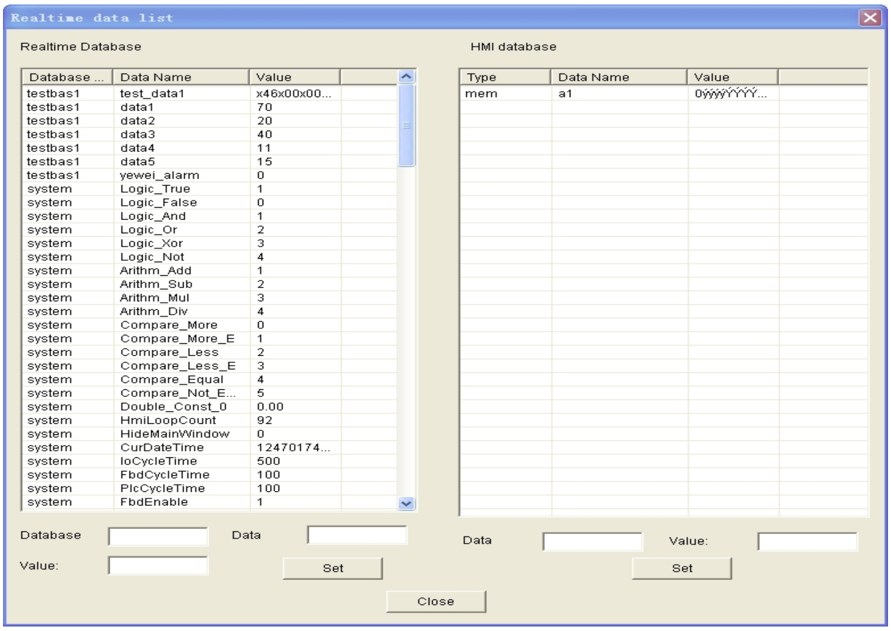


Database ...	Data Name	Value
testbas1	test_data1	x46x00x00...
testbas1	data1	70
testbas1	data2	20
testbas1	data3	40
testbas1	data4	11
testbas1	data5	15
testbas1	yewei_alarm	0
system	Logic_True	1
system	Logic_False	0
system	Logic_And	1
system	Logic_Or	2
system	Logic_Xor	3
system	Logic_Not	4
system	Arithm_Add	1
system	Arithm_Sub	2
system	Arithm_Mul	3
system	Arithm_Div	4
system	Compare_More	0
system	Compare_More_E	1
system	Compare_Less	2
system	Compare_Less_E	3
system	Compare_Equal	4
system	Compare_Not_E...	5
system	Double_Const_0	0.00
system	HmiLoopCount	92
system	HideMainWindow	0
system	CurDateTime	12470174...
system	IoCycleTime	500
system	FbdCycleTime	100
system	PlcCycleTime	100
system	FbdEnable	1

Type	Data Name	Value
mem	a1	0ýýýÝÝÝÝ...

Figuur 7.6

Click on the button in the window, and you can see that the values of data1 to data5 change to 11, 15, 20, 40, and 70 accordingly, as shown in Figuur 7.7. The data values are sorted in sequence.



Database ...	Data Name	Value
testbas1	test_data1	x46x00x00...
testbas1	data1	70
testbas1	data2	20
testbas1	data3	40
testbas1	data4	11
testbas1	data5	15
testbas1	yewei_alarm	0
system	Logic_True	1
system	Logic_False	0
system	Logic_And	1
system	Logic_Or	2
system	Logic_Xor	3
system	Logic_Not	4
system	Arithm_Add	1
system	Arithm_Sub	2
system	Arithm_Mul	3
system	Arithm_Div	4
system	Compare_More	0
system	Compare_More_E	1
system	Compare_Less	2
system	Compare_Less_E	3
system	Compare_Equal	4
system	Compare_Not_E...	5
system	Double_Const_0	0.00
system	HmiLoopCount	92
system	HideMainWindow	0
system	CurDateTime	12470174...
system	IoCycleTime	500
system	FbdCycleTime	100
system	PlcCycleTime	100
system	FbdEnable	1

Type	Data Name	Value
mem	a1	0ýýýÝÝÝÝ...

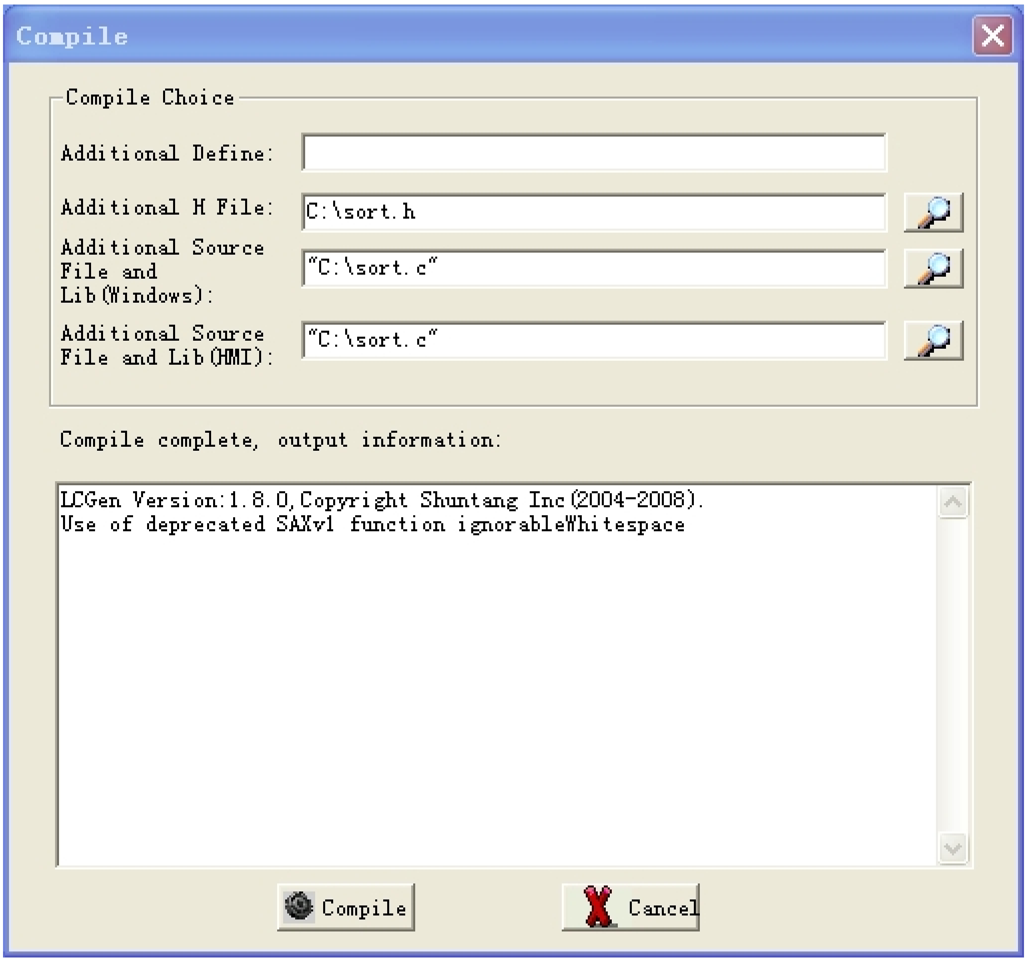
Figuur 7.7

7.4 Script Compiling

After configuration, the project needs to be compiled before it can run. If the compiling is successful, the following information will be displayed in the **Compile a Project** dialog box, as shown in Figuur 7.8:

LCGen Version:1.7.0,Copyright EASY Inc(2004-2008).

Use of deprecated SAXv1 function ignorableWhitespace


Compile
Compile Choice
Additional Define:
Additional H File:
Additional Source File and Lib(Windows):
Additional Source File and Lib(HMI):
Compile complete, output information:
Compile
Cancel

Additional H File: C:\sort.h

Additional Source File and Lib(Windows): "C:\sort.c"

Additional Source File and Lib(HMI): "C:\sort.c"

```
LCGen Version:1.8.0,Copyright Shuntang Inc(2004-2008).
Use of deprecated SAXv1 function ignorableWhitespace
```

Figuur 7.8

If the above information is not displayed, it means the compiling fails; in this case, you need to find out what causes the failure.

Let’s take an example here to explain how to analyze what causes the compiling failure based on the error information.

Suppose that you add a button **button1** in the **test** interface and that the following script is conFiguurd for the **Press** event for this button:

```
$test.data1 = 1
```

The problem of this script is that the statement doesn’t end with the semi-colon **;**, which is required by the syntax of the C language. This script will cause the following error prompt during the project compiling:

```
LCGen Version:1.7.0,Copyright EASY Inc(2004-2008).
Use of deprecated SAXv1 function ignorableWhitespace
C:\Documents and Settings\JiangJian\Desktop\test_c\compile\event_funcs.c: In function `widget_test_button1_click':
C:\Documents and Settings\JiangJian\Desktop\test_c\compile\event_funcs.c:26: error: syntax
```

```
    error before '}' token
    C:\Documents and Settings\JiangJian\Desktop\test_c\compile\event_funcs.c: In function
    `widget_test_button1_click':
C:\Documents and Settings\JiangJian\Desktop\test_c\compile\event_funcs.c:26: error: syntax error
before '}' token
```

The above error prompt indicates which script causes the compiling failure. The string in bold **widget_test_button1_click** is divided into four groups by the underscore, which are explained as follows:

- **widget**: Fixed prefix.
- **test**: Name of an interface.
- **button1**: Name of a graphic control component.
- **click**: Dynamic property or event of the graphic control component. It refers to the **Press** event here.

The above error prompt indicates that some error occurs during the compiling of script for the **Press** event of the **button1** button in the **test** interface.

7.5 Script Commissioning

7.5.1Overview

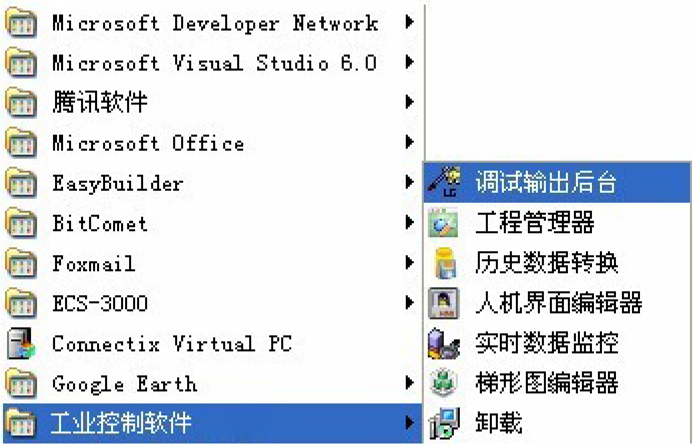
During the program development, you might get some script-related issues; for example, the script fails to run or the script execution turns out different from what you expected. In this case, you might want to print out some important information for analyzing what causes the script execution error.

Considering this, EASY provides the **Commissioning Output Background** tool. You can add the printing commissioning information anywhere in the script, and these information will be exported to the commissioning window of the **Commissioning Output Background** tool.

The following of this section describes in details how to use this tool.

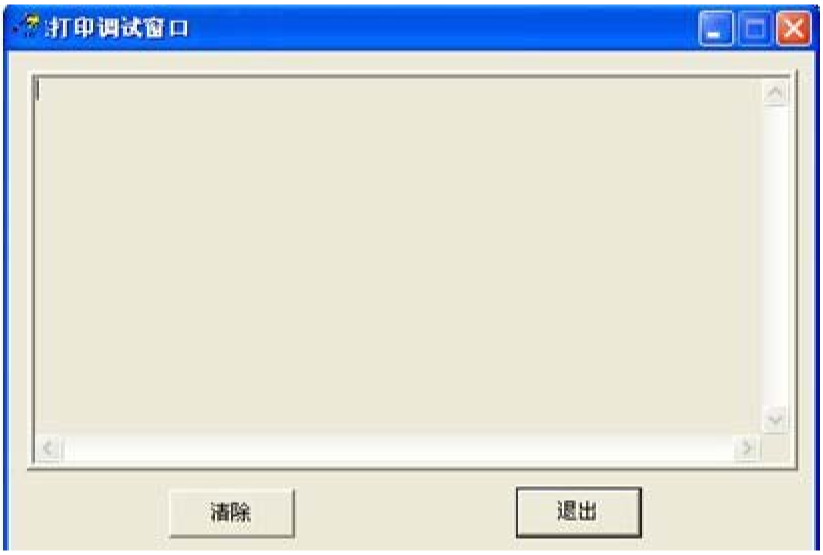
1. Start the **Commissioning Output Background** tool, as follows:

After EASY is installed, click on **Start** > **EASY Industrial Control Software** > **Commissioning Output Background**, as shown in Figuur 7.9.



Figuur 7.9

And then you'll see the **EASY Printing Commissioning** Wnidow, as shown in Figuur 7.10.



Figuur 7.10

The printing commissioning information will be exported and displayed in this window.

2. Set the IP adres of the commissioning host (namely, the computer on which the **Commissioning Output Background** tool is running.)

You can call the function **debug_printf** to export the commissioning information. But before that, you must call the function **debug_set_ip** to set the IP adres of the commissioning host on which the **Commissioning Output Background** tool is running.

You can call the function **debug_set_ip** any time, just on one condition that it's called before the function **debug_printf**. In addition, the function **debug_set_ip** can be called more than one time, and each calling will overwrite the IP adres set by the previous calling. For details about the function **debug_set_ip**, see section 7.5.2 System Functions for Script Commissioning.

3. Call the function **debug_printf** to export the commissioning information whenever necessary.

For details about the function **debug_printf**, see section 7.5.2 System Functions for Script Commissioning.

7.5.2System Functions for Script Commissioning

7.5.2.1 debug_set_ip

Original Function: void **debug_set_ip**(**const char** **ip*)

Function Description: To set the IP adres of the host on which the **Commissioning Output Background** tool is running.

Return Value: None.

Parameter: *ip*: IP adres of the host on which the **Commissioning Output Background** tool is running.

Example: debug_set_ip("127.0.0.1");

7.5.2.2 debug_printf

Original Function: void **debug_printf**(**const char** **format*, …)

Function Description: To export the printing commissioning information to the commissioning host. Use this function in the same way as the library function **printf** in the standard C language.

Return Value: None.

Parameter: *format*: String for format control. Same as for the library function **printf** in the standard C language.

…: Optional parameter. Same as for the library function **printf** in the standard C language.

Example: debug_printf("i=%d\n", i);

Chapter 8 Real-Time Trend Curves

8.1 Overview

In the real-time operation, you might often need to observe how the data changes within a period of time. A very simple and straightforward way is to draw real-time trend curves for these data. The change in the curve shows vividly the data change trend. Because of this, the real-time trend curve is a very important part in the industrial control system.

EASY provides powerful real-time trend curve functions. It not only provides you an easy way to draw real-time trend curves, but also allows you to save the data for future data analysis according to your needs.

EASY offers the following two solutions for implementing the real-time trend curve functions:

1. Function interface provided by the system

   EASY provides a set of function interfaces. You can select the programming that best suits your needs to realize real-time trend curves.

   These are the advanced functions, highly flexible and suitable for some special situations, and thus are not introduced in details here.

2. System default method

   The system default method for realizing real-time trend curves is simple and easy to use. You just need to do some simple settings during the configuration; almost no programming involved at all.

   The following of this Hoofdstuk describes this method in details.

If you select the system default method for realizing real-time trend curves, please follow the procedure below:

1. Create real-time data records for the data you want to draw real-time trend curves.
2. During the interface configuration, use the **Real-Time Trend Curve** control to display the real-time trend curves of the selected data.
3. (Optional step) Save the real-time data records if necessary, and use the **Real-Time Trend Curve** control to view the real-time trend curves of the selected data.

The following sections will describe these three steps in details.

8.2 Definition of Real-Time Data Records

You can define one or more real-time data records, and each can contain one or more data for which real-time trend curves needs to be drawn. All the data in one real-time data record share some common properties; for example, they will all be collected with the same time cycle.

You can also define a data into different real-time data records. In this case, this data has various different properties. For example, a data needs to be collected 100 times every 1 second on one real-time trend curve, but needs to be collected 200 times every 5 seconds on another real-time trend curve. In this case, you will need to define this data into two separate data records.

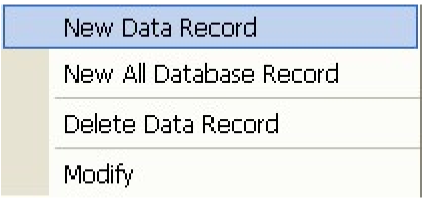
During the interface configuration, each **Real-Time Trend Curve** control associates one real-time data record only. However, this **Real-Time Trend Curve** control can display all the data defined in the real-time data record to which this control associates.

8.2.1Creating a Data Record

To create a data record, do as follows:

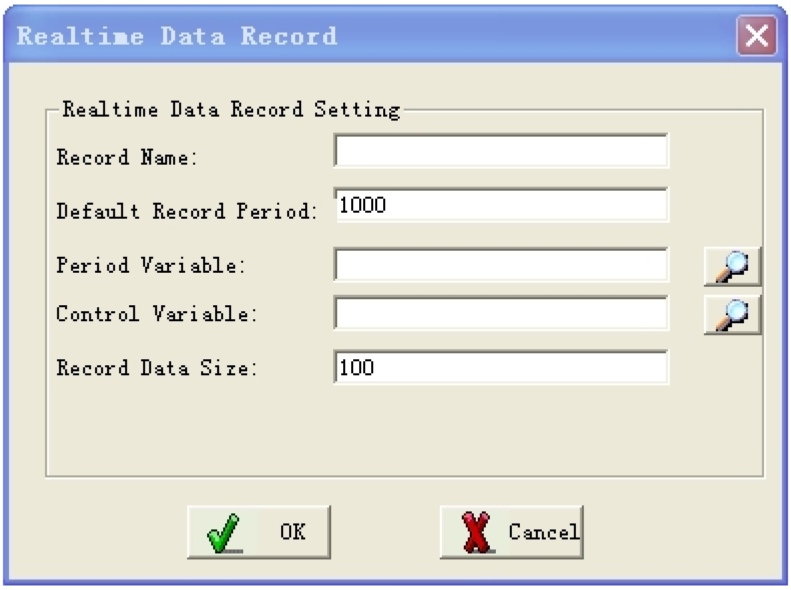
1) Select the **Real-Time Data Record** node on the left side of the **Project Manager** window, and right-click on it.

   And you will see a right-click menu as shown in Figuur 8.1.



Figuur 8.1

2) Select **Create a Data Record**, and you will see a dialog box as shown in Figuur 8.2.



Realtime Data Record

Realtime Data Record Setting

Record Name:

Default Record Period: 1000

Period Variable:

Control Variable:

Record Data Size: 100

Figuur 8.2

The parameters in Figuur 8.2 are described as follows:

- **Record Name**: Name of a real-time data record.
- **Default Record Cycle**: Default time cycle for collecting data (unit: ms), which defines the time cycle for the system to collect data for all the data defined in a real-time data record.
- **Cycle Variable**: You can associate a time cycle for data collection to a real-time database variable. In this case, the time cycle can change dynamically during the system operation. If no cycle variable is defined, the value set for **Default Record Cycle** will be used as the collection cycle; otherwise, the value set for **Cycle Variable** will be used as the collection cycle, which turns the value set for **Default Record Cycle** invalid.
- **Control Variable**: controls how a data record operates. It is valued as below:

0: means to start collecting data for all the data defined in the real-time data record.

1: means to stop collecting data for all the data defined in the real-time data record.

2: means to empty the data collected for all the data defined in the real-time data record.

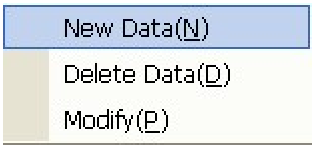
- **Data Collection Volume**: defines how many times the data is to be collected for all the data defined in the data record. For example, if you set **Default Record Cycle** to **1000** and **Data Collection Volume** to **100**, that means data will be collected for all the data defined in the data record every 1000 ms and 100 data will be collected the most. If more than 100 data are collected, the later collected data will overwrite the previously collected data. For example, if the 101st data is collected, the 1st collected data will be replaced.

8.2.2Adding Data

After you create a data record, you will need to add data into this data record. To add data, do as follows:

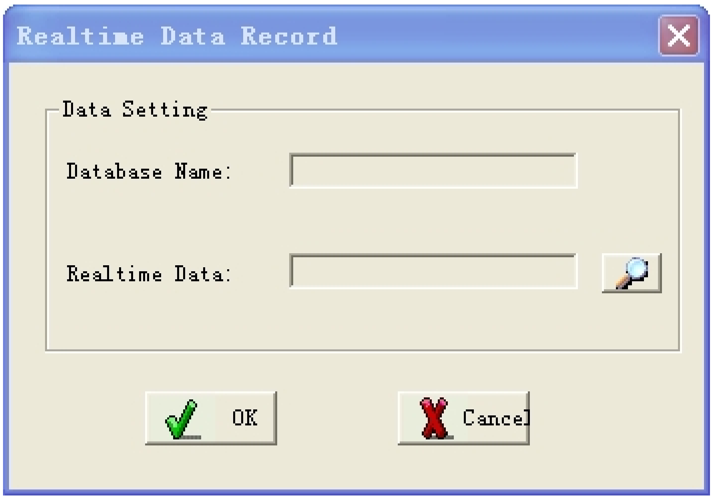
1) Select a real-time data record on the left side of the **Project Manager** window, and right-click in the list pane on the right side of the window.

   And you will see a right-click menu as shown in Figuur 8.3.



Figuur 8.3

2) Select **Add Data**, and you will see a dialog box as shown in Figuur 8.4.



Figuur 8.4

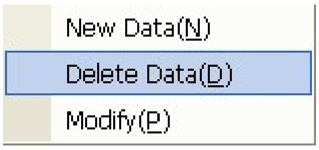
3) Select data from the real-time database to add into this data record.

8.2.3Deleting Data

To delete a data defined in a real-time data record, do as follows:

1) Select a real-time data record on the left side of the **Project Manager** window, and right-click on the data you want to delete in the list pane on the right side.

   And you will see a right-click menu as shwon in Figuur 8.5.



Figuur 8.5

2) Select **Delete Data**.

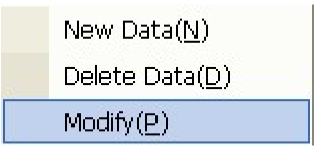
   The selected data will be deleted from the data record.

8.2.4Modifying Data

To modify a data defined in a real-time data record, do as follows:

1) Select a real-time data record on the left side of the **Project Manager** window, and right-click on a data you want to modify in the list pane on the right side of the window.

   And you will see a right-click menu as shown in Figuur 8.6.



Figuur 8.6

2) Select **Modify**.

   And you can modify the data according to your needs.

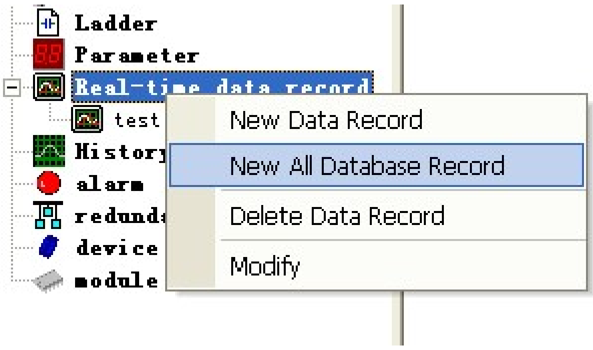
8.2.5Creating a Whole Database Record

A **Real-Time Whole Database Record** allows you to add all data defined in the real-time database into a data record, which saves the trouble of adding the data for drawing real-time trend curves one by one.

To create a whole database record, do as follows:

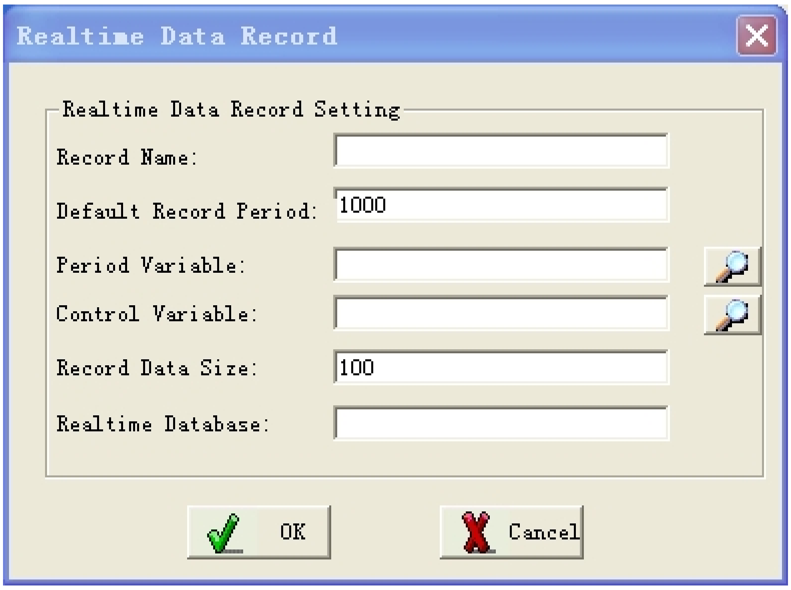
1) Select the **Real-Time Data Record** node on the left side of the **Project Manager** window and right-click on it.

   And you will see a right-click menu as shown in Figuur 8.7.



Figuur 8.7

2) Select **Create a Whole Database Record**, and you will see a dialog box as shown in Figuur 8.8.



Realtime Data Record

Realtime Data Record Setting

Record Name:

Default Record Period: 1000

Period Variable:

Control Variable:

Record Data Size: 100

Realtime Database:

Figuur 8.8

The parameters in Figuur 8.8 are described as follows:

- **Record Name**: Name of a real-time data record.
- **Default Record Cycle**: Default time cycle for collecting data (unit: ms), which defines the time cycle for the system to collect data for all the data defined in a real-time data record.
- **Cycle Variable**: You can associate a time cycle for data collection to a real-time database variable. In this case, the time cycle can change dynamically during the system operation. If you set **Cycle Variable**, the value set for **Default Record Cycle** will become invalid.
- **Control Variable**: controls how a data record operates. It is valued as below:

0: means to start collecting data for all the data defined in the real-time data record.

1: means to stop collecting data for all the data defined in the real-time data record.

2: means to empty the data collected for all the data defined in the real-time data record.

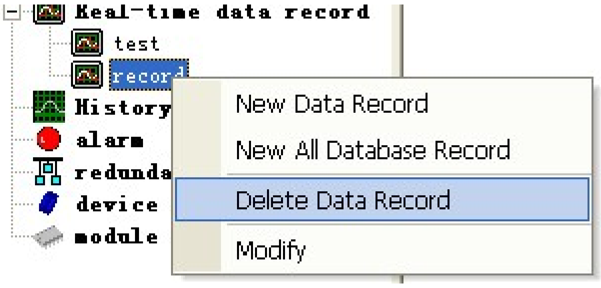
- **Data Collection Volume**: defines how many times the data is to be collected for all the data defined in the data record. For example, if you set **Default Record Cycle** to **1000** and **Data Collection Volume** to **100**, that means data will be collected for all the data defined in the data record every 1000 ms and 100 data will be collected the most. If more than 100 data are collected, the later collected data will overwrite the previously collected data. For example, if the 101st data is collected, the 1st collected data will be replaced.
- **Real-Time Database Name**: Name of the real-time database from where all the data will be selected and added to this whole database record.

8.2.6Deleting a Data Record

To delete a data record, do as follows:

1) Select a data record on the left side of the **Project Manager** window, and right-click on it.

   And you will see a right-click menu as shwon in Figuur 8.9.



Figuur 8.9

2) Select **Delete a Data Record**.

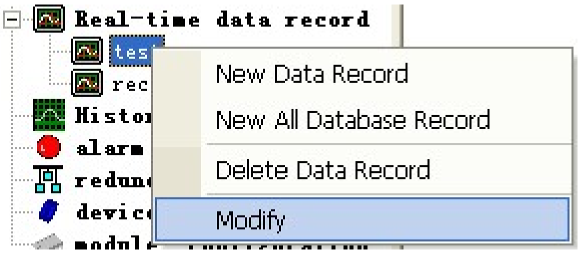
   And the selected data record will be deleted.

8.2.7Modifying a Data Record

To modify a data record, do as follows:

1) Select a data record you want to modify on the left side of the **Project Manager** window, and right-click on it.

   And you will see a right-click menu as shwon in Figuur 8.10.



Figuur 8.10

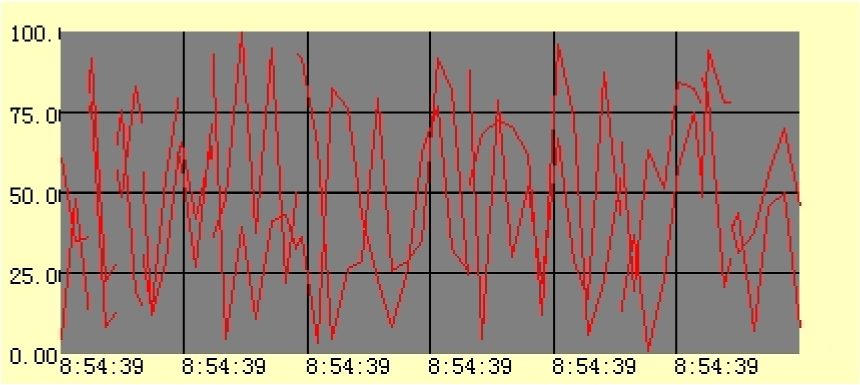
2) Select **Modify**.

   And you can modify the data according to your needs.

8.3 Control - Real-Time Trend Curves

8.3.1Overview

In the **Interface Editor** window, click on the **Real-Time Trend** button in the tool set on the left side, and move the cursor to the editting area on the right side, and you can see the cursor become a cross. Drag the mouse in the editting area to draw a rectangle, and real-time trend curves will be displayed in this rectangle, as shown in Figuur 8.11.



Figuur 8.11

In the middle of the **Real-Time Trend Curve** control is a drawing area with gridlines. The real-time trend curves will be displayed within this area. On the left side of the gridlines is the X-axis (for time), and at the bottom is the Y-axis (for value). You can select a real-time trend curve object (8 small rectangles will appear on the sides of the object once selected) to move the object or change the size of the object.

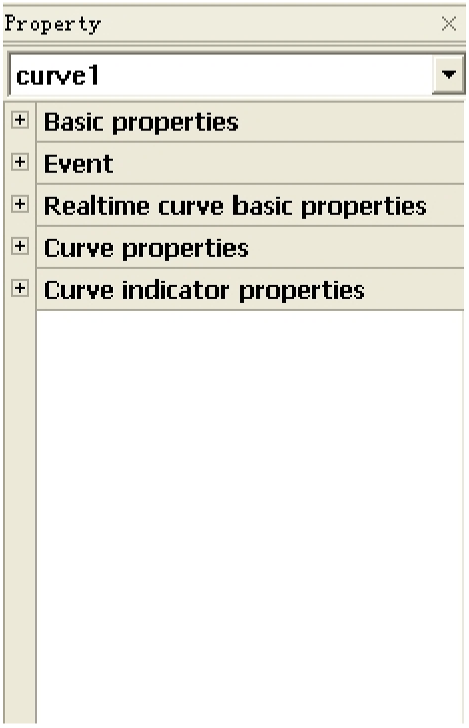
The gridlines are composed of two parts: the ones vertical to the X-axis and the ones vertical to the Y-axis. You can set the numbers of gridlines for each direction. For example, if you set the number of vertical gridlines to 5, then the whole curve area will be divided into 6 identical areas.

8.3.2Properties of Real-Time Trend Curves

Select a real-time trend curve with a left click, and you will see the **Property List** pane displayed on the right side of the editting area, listing all the properties of the selected real-time trend curve.

Real-time trend curves have the following five property nodes, as shown in Figuur 8.12:

- Basic Properties
- Events
- Real-Time Trend Basic Properties
- Real-Time Trend Curve Properties
- Real-Time Trend Indicator Line Properties



Property

curve1

- Basic properties
- Event
- Realtime curve basic properties
- Curve properties
- Curve indicator properties

Figuur 8.12

For details about basic properties, see section 5.3.3 Basic Properties. For details about events properties, see secton 5.3.5 Events.

8.3.2.1 Basic Properties

A **Real-Time Trend Curve** control can display curves for multiple data. This section describes the basic properties of all the curves.

Property	Description	Remarks on Dynamic Properties
Minimum Value	Value of the startpoint on the Y-axis.	The return value is numeric. Changing these two values will zoom or move the curve vertically.
Maximum Value	Value of the endpoint on the Y-axis.	
Maximum Horizontal Points	Maximum data points in the horizontal direction.	No dynamic properties.
Horizontal Points	Data points distributed in the horizontal direction.	The return value is an integer. This property and the property **Maximum Horizontal Points** together zoom the curves horizontally.
Number of Vertical Lines	Number of the gridlines vertical to the Y-axis.	The return value is an integer.
Number of Horizontal Lines	Number of the gridlines vertical to the X-axis.	The return value is an integer.
Horizontal Spacing	Spacing between the curve drawing area and the left or right margin of the curve control.	The return value is an integer.
Vertical Spacing	Spacing between the curve drawing area and the top	The return value is an integer.

Property	Description	Remarks on Dynamic Properties
	or bottom margin of the curve control.	
Background Color	Background color of the curve control.	The return value of the expression or dynamic script is the RGB value of the defined color.
Background Color of Curves	Background color of the curve drawing area of the curve control.	
Color of Vertical Lines	Color of the gridlines vertical to the Y-axis.	
Color of Horizontal Lines	Color of the gridlines vertical to the X-axis.	
Color of Text	Color of the text beside the X-axis and the Y-axis.	
Number of Curves	Defines the number of curves to be displayed (16 the most).	No dynamic properties.
Data Source	Source of the data for drawing the trend curve: 1) **Real-Time Record in Memory**: The data selected for drawing the trend curve come from the current values of the data in the real-time data record. 2) **File**: You can save real-time data records to files. If you set the data source to **Files**, then the saved trend curves will be displayed.	No dynamic properties.
Trend Name	Valid when **Data Source** is set to **Real-Time Records in Memory**. The trend name indicates the name of the real-time data record. It must be defined in the real-time data record in **Project Manager**.	No dynamic properties.
File Storage Location	Valid when **Data Source** is set to **File**. This property indicates the location whereh the record file is stored, either the internal flash or the CF card.	The return value is 0 or 1: • 0: Internal flash • 1: CF card
File Name	Valid when **Data Source** is set to **File**. This property indicates the name of the record file.	The return value is a string which contains the name of the record file.
Start Point	Valid when **Data Source** is set to **File**. This property indicates from which data point of the record the curve starts to display.	The return value is int. Changing the value of this property will move the curve horizontally.

8.3.2.2 Properties of Curves

This section describes the data to which each curve is associated and the color of the curve. In total, 16 curves can be conFiguurd the most.

Property	Description	Remarks on Dynamic Properties
Variable Name	Defines the name of the variable to which the curve is associated. This variable must be defined in the real-time data record. If no variable is defined here, then the corresponding	The return value is a string, which is the name of the variable data to which the curve is associated. If a blank string (“”) is returned, it means the curve will not be displayed.

	curve will not be displayed.	
Curve Color	Defines the color of the curve.	The return value of the expression or the dynamic script is the RGB value of the defined color.

8.3.2.3 Properties of Indicator Lines

Property	Description	Remarks on Dynamic Properties
Allow Indicator Line	Defines whether the control provides indicator lines.	The return value is bit, as follows: • 0: Not allow • None 0: Allow
Indicator Line Color	Defines the color of the indicator line.	The return value of the expression or the dynamic script is the RGB value of the defined color.
Time Variable	The time data where the indicator line points to is saved to this variable.	The return value is a string, which is the name of the time variable.
Data Value Variable	The data value of the curve where the indicator line points to is saved to this variable.	The return value is a string, which is the name of the data value variable.

8.4 Saving Real-Time Data Records

EASY allows you to save the data defined in real-time data records and to view the curves using the **Real-Time Trend Curve** control. You can call the function **rtdb_log_save_file** to save real-time data records. For details, please see section 8.5 System Functions for Real-Time Trend Curves.

You can also view the curves for the saved data using the **Real-Time Trend Curve** control. (You must set **Data Source** to **File**. For details, see section 8.3.2.1 Basic Properties.) Alternatively, you can call the function **rtdb_get_log_data_from_file**. For details about this function, see section 8.5 System Functions for Real-Time Trend Curves.

8.5 System Functions for Real-Time Trend Curves

8.5.1 rtdb_log_save_file

Original Function: int **rtdb_log_save_file**(**char ****logname*, **int** *save_dir*, **char ****filename*)

Function Description: To save a real-time data record into a file.

Return Value: 0 Failed

1 Successful

Parameters: *logname*: Name of a real-time data record.

save_dir: 0: HMI internal flash; 1: CF card.

filename: Name of the file for saving the real-time data record.

Example: rtdb_log_save_file("real", 0, "recfile.log")

8.5.2 rtdb_get_log_data

Original Function: int **rtdb_get_log_data**(**char ****logname***, char ****dataname***, u8 ****buf***, int** *log_number*)

Function Description: To obtain data from the current real-time data record.

Return Value: 0: Failed

Other values: Actual volume of data collected.

Parameters: *logname*: Name of a real-time data record.

dataname: Name of the data variable for which data is to be collected.

buf: Buffer for the collected data. You need to assign space for the butter in advance.

log_number: Volume of data to be collected.

Example: rtdb_get_log_data("real", "test.data1", buf, 100)

8.5.3 rtdb_get_log_data_from_file

Original Function: int **rtdb_get_log_data_from_file**(**int** *file_path,* **char** **filename,* **char ****dataname,* **u8** **buf,* **int** *log_number,* **int** *start_pt*)

Function Description: To obtain data from the saved real-time record file.

Return Value: 0: Failed

Other values: Actual volume of data collected.

Parameters: *file_path*: 0: HMI internal flash; 1: CF card.

filename: Name of the file where the real-time data record is saved.

dataname: Name of the data variable for which data is to be collected.

buf: Buffer for the collected data. You need to assign space for the butter in advance.

log_number: Volume of data to be collected.

start_pt: Start point from where data is collected.

Example: rtdb_get_log_data_from_file(0, " recfile.log ", "test.data1", buf, 100, 0)

Chapter 9 Historical Data Processing

9.1 Overview

The data saving function is of vital importantance to any industrial system. As the industrial automation becomes more and more popular and advanced, the demands and requirements for saving and accessing important data of industrial sites become more and more complicated as well. The traditional HMIs disclose more and more disadvantages, for example:

- Inability of saving large quantities of data
- Slow saving speed
- High risks of data loss
- Short saving period
- Huge space occupation for the saved data
- Slow access speed

Therefore, for large-scale systems with high requirements, the issue of saving and accessing historical data becomes more and more crucial.

Considering this development tendency, EASY HMI comes out with the idea of high-speed historical database, which supports as high-speed as millisecond saving and inquiring of historical data. EASY adopts the most advanced data compression and search technologies, which achieves the compression ratio of the database lower than 20%, greatly saving the disk space. In addition, the data inquiry speed is considerably increased, allowing you to query the data at any time. Besides, you can download the data at any time to an external device, such as a thumb drive or external hard drive, which solves the issue of data loss.

In the EASY system, all the data variables that can be defined in the real-time database, such as the discrete, int, real type, and string variables, support historical data saving. EASY supports the following three modes of historical data saving:

- Timed Saving (minimum unit: 1ms)
- Saving at Data Change
- Variable-Triggered Saving

9.2 Historical Data Records

9.2.1Overview

Before saving the collected data into the database, you need to create historical data records first. You can define one or more historical data records, and each can contain one or more data to be saved.

While adding the data to be saved into a historical data record, besides defining the name of the data, you also need to define the data saving mode: timed saving, data change saving, or variable-triggered saving. When the defined saving condition of the defined saving mode is satisfied, the system will save the data automatically.

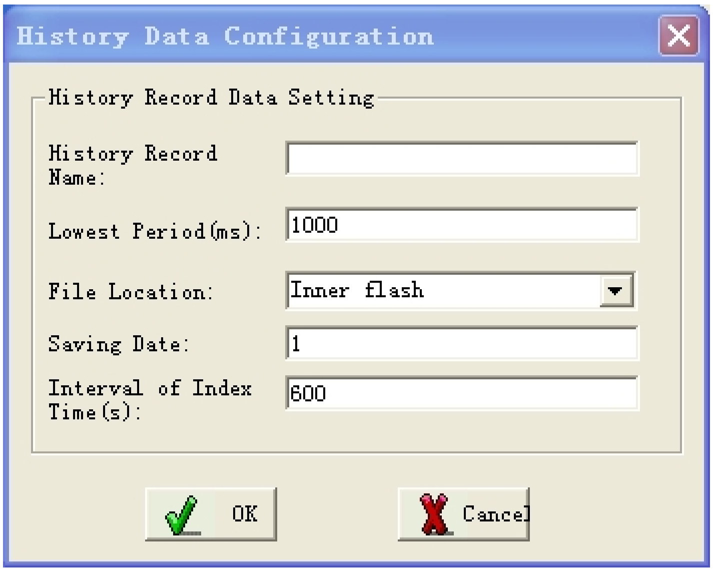
All the data in a historical data record share some common properties; for example, they will be collected at the same time cycle. You can also define a data into different historical data records. In this case, this data will have various different properties. For example, you want to collect a data 100 times at the interval of 1s for one real-time trend curve; while for the same data, you want to collect it 200 times at the interval of 5s for another. In this case, you can define this data into two individual data records.

During the interface configuration, each **Historical Trend Curve** or **Historical Data List** control associates one historical data record only. However, this **Historical Trend Curve** or **Historical Data List** control can display all the data defined in the historical data record to which the control associates.

9.2.2Creating a Historical Data Record

To create a historical data record, do as follows:

1) Select the **Historical Data Record** node on the left side of the **Project Manager** window.
2) Right-click on it and select **Add a Historical Data Record**.
   And you will see the dialog box as shown in Figuur 9.1.



History Data Configuration

History Record Data Setting

History Record Name:

Lowest Period(ms): 1000

File Location: Inner flash

Saving Date: 1

Interval of Index Time(s): 600

Figuur 9.1

The parameters in Figuur 9.1 are described as follows:

- **Historical Record Name**: Name of a historical data record.
- **Minimum Time Cycle**: Minimum time cycle for processing and saving the data defined in a historical data record.
  The system checks periodically (depending on the minimum time cycle defined here) whether the conditions set for saving the individual data defined in a historical data record are satisfied. If satisfied, the system will save the data into the historical database.
  In other words, this parameter actually defines the finest level of granularity for processing a historical data record. For example, suppose you set this parameter to **1s** and set the mode for saving a data in a historical data record to **Variable-Triggered**. If the related variable jumps multiple times within 1s, then the system records only the value of the last jump, while the previous jumps will not be captured.
- **File Saving Location**: defines where to save the file of a historical data record. You can choose to save the file into the internal flash or the C Fcard.
- **Data Saving Days**: defines the maximum days of saving the historical data in the historical database.
  When the defined number of days expires, the system will automatically delete the historical data collected.
- **Index interval**: defines the interval for creating index for the saved historical data. Creating index will speed up the search of the historical data. However, it will increase the space occupied.

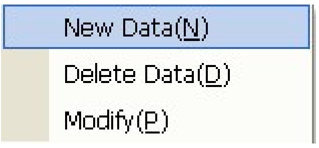
9.2.3Adding Data

After you create a historical data record, you need to add data into this record.

To add data into a record, do as follows:

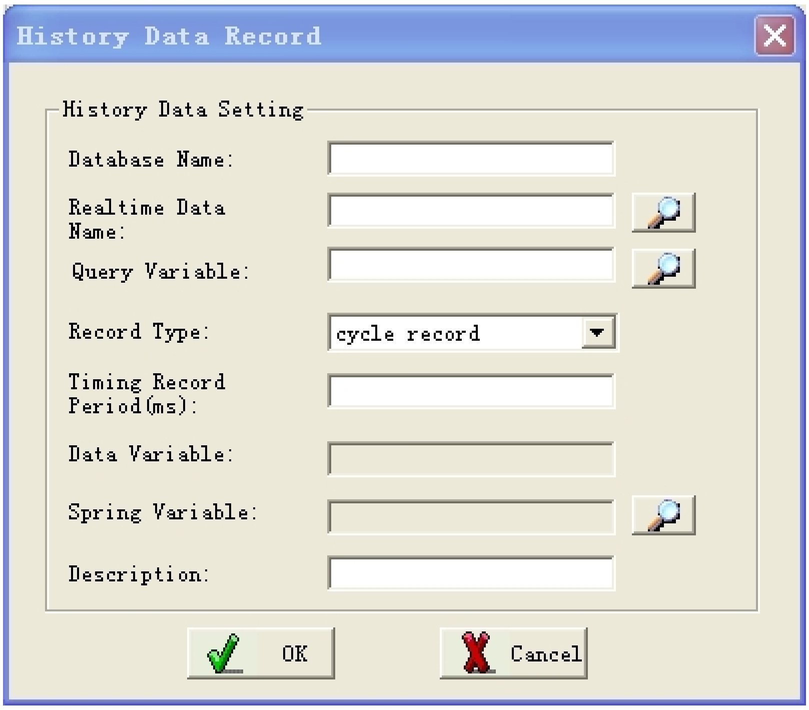
1) Select a historical data record on the left side of the **Project Manager** window,

and then right-click in the list pane on the right side of the window, as shown in Figuur 9.2.



Figuur 9.2

2) Select **Add Data**, and you will see a dialog box as shown in Figuur 9.3.


History Data Record

History Data Setting

Database Name:

Realtime Data Name:

Query Variable:

Record Type: cycle record

Timing Record Period(ms):

Data Variable:

Spring Variable:

Description:

Figuur 9.3

The parameters in Figuur 9.3 are described as follows:

- **Database Name**: Name of the database from where you select the data for adding into the record.
- **Real-Time Data Name**: Name of the variable corresponding to the data to be saved.
- **Inquiry Variable Name**: While you query a saved historical data, the queried data value will be saved to this variable. For details, see section 9.5 Historical Data Inquiry. If no inquiry variable is defined, it means there is no need to query this data.
- **Saving Mode**: defines the condition for saving a specific data.
  The system supports the following three saving modes:
  - **Timed Saving**: The system saves the data value into the historical database at a specified interval, no matter whether the data value changes.
  - **Saving at Data Change**: The values of variables keep changing during the system operation. The system saves the changed value of a variable only when the difference between the current value and the previous value is greater than the defined data change value.
    For example, you want to save the value of a real type variable, and you set

the data change value to 1. Suppose that the first value of this real type variable is saved as 10 in the system.

When the value of this variable changes to 10.9, this value will not be saved because 10.9-10=0.9<1 (namely, the difference between the current value and the previous value is less than the defined data change value). When the value changes to 12, the changed value 12 will be saved to the historical record because 12-10.9=1.1>1 (namely, the difference is greater than the defined data change value).

  - **Variable-Triggered Saving**: When the value of the defined trigger variable becomes **1**, the system starts saving the data. After the data is saved, the system automatically resets the value of the trigger variable to **0**.

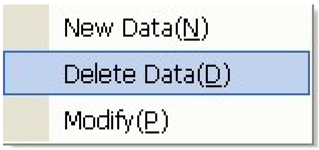
- **Timed Saving Interval**: When you set **Saving Mode** to **Timed Saving**, this parameter defines the interval for saving the selected data.
- **Data Change Value**: When you set **Saving Mode** to **Saving at Data Change**, this parameter defines the value of the data change.
- **Trigger Variable**: When you set **Saving Mode** to **Variable-Triggered Saving**, this parameter specifies the trigger variable.
- **Description**: This parameter can have following two types of values:
  - Descriptive text for the data;
  - Name of the relative field to be displayed in the **Historical Data List** control.

9.2.4Deleting Data

To delete a data defined in a historical data record, do as follows:

1) Select the name of the historical data record on the left side of the **Project Manager** window, and right-click on the data you want to delete in the list pane on the right side.

   And you will see a right-click menu as shown in Figuur 9.4.



Figuur 9.4

2) Select **Delete Data**.

   And the selected data will be deleted.

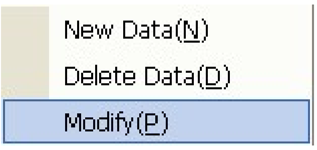
9.2.5Modifying Data

To modify a data defined in a historical data record, do as follows:

1) Select the name of the historical data record on the left side of the **Project Manager** window, and right-click on the data you want to modify in the list pane

on the right side.

And you will see a right-click menu as shown in Figuur 9.5.



Figuur 9.5

2) Select **Modify**.

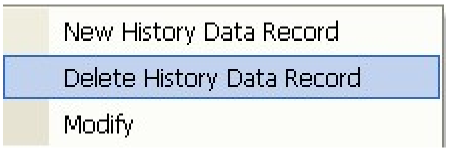
And you can modify the data according to your needs.

9.2.6Deleting a Historical Data Record

To delete a historical data record, do as follows:

1) Select the name of the historical data record on the left side of the **Project Manager** window, and right-click on it.

And you will see a right-click menu as shown in Figuur 9.6.



Figuur 9.6

2) Select **Delete a Historical Data Record**.

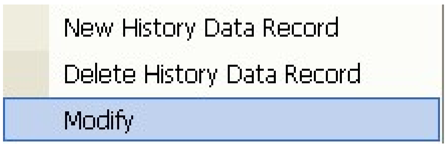
And the selected historical data record will be deleted.

9.2.7Modifying a Historical Data Record

To modify a historical data record, do as follows:

1) Select the name of the historical data record on the left side of the **Project Manager** window, and right-click on it.

And you will see a right-click menu as shown in Figuur 9.7.



Figuur 9.7

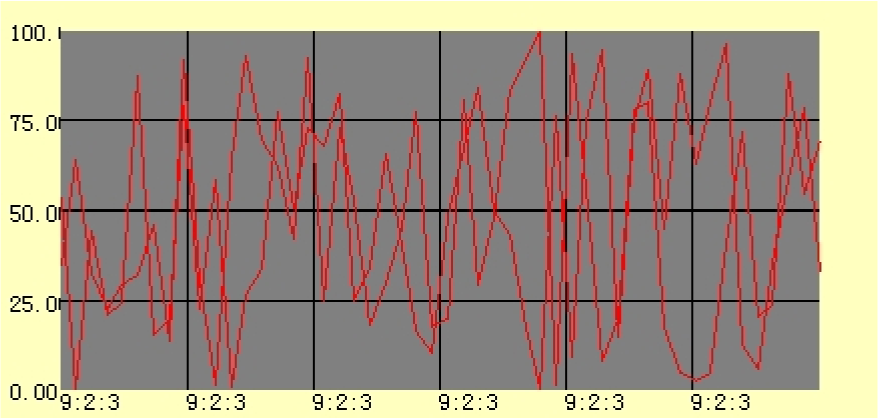
2) Select **Modify**.

And you can modify the historical data record according to your needs.

9.3 Control - Historical Trend Curves

9.3.1Overview

In the **Interface Editor** window, click on the **Historical Trend** button in the tool set on the left side and move the cursor to the editting area on the right side, you will see the cursor becomes a cross. Drag the cursor to draw a rectangle, and the historical trend curves will be displayed in this rectangle, as shown in Figuur 9.8.



Figuur 9.8

In the middle of the **Historical Trend Curve** control is a drawing area with gridlines. The historical curves will be displayed within this area. On the left side of the gridlines is the X-axis (for time), and at the bottom is the Y-axis (for value). You can select a historical trend curve object (8 small rectangles will appear on the sides of the object once selected) to move the object or change the size of the object.

The gridlines are composed of two parts: the ones vertical to the X-axis and the ones vertical to the Y-axis. You can set the numbers of gridlines for each direction. For example, if you set the number of vertical gridlines to 5, then the whole curve area will be divided into 6 identical areas.

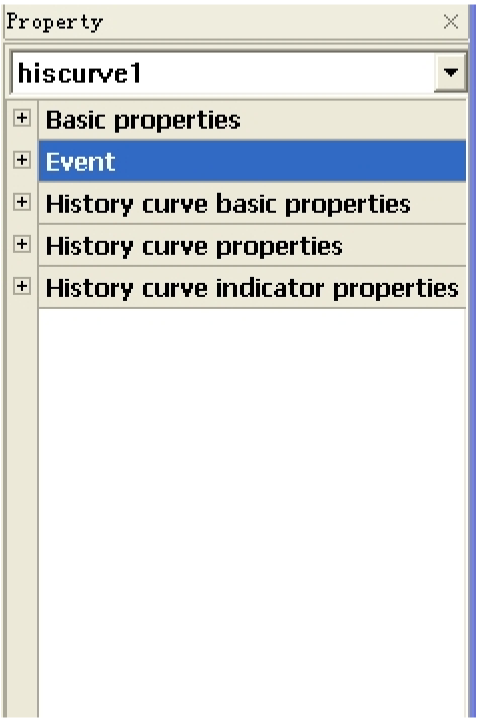
9.3.2Properties of Historical Trend Curves

After historical trend curves are drawn, select any of the curves with a left click, and you will see the **Property List** pane displayed on the right side of the editing area, listing all the properties of the selected historical curve.

Historical curves have the following five property nodes, as shown in Figuur 9.9:

- Basic Properties
- Events

- Historical Trend Basic Properties
- Historical Trend Curve Properties
- Historical Trend Indicator Line Properties


Property
hiscurve1
Basic properties
Event
History curve basic properties
History curve properties
History curve indicator properties

Figuur 9.9

For details about basic properties, see section 5.3.3 Basic Properties. For details about events properties, see secton 5.3.5 Events.

9.3.2.1 Basic Properties

A **Historical Trend Curve** control can display curves for multiple data. This section describes the basic properties of all the curves.

Property	Description	Remarks on Dynamic Properties
Minimum Value	Value of the startpoint on the Y-axis.	The return value is numeric. Changing these two values will zoom or move the curve vertically.
Maximum Value	Value of the endpoint on the Y-axis.	
Horizontal Points	Data points distributed in the horizontal direction.	No dynamic properties.
Number of Vertical Lines	Number of the gridlines vertical to the Y-axis.	The return value is an integer.
Number of Horizontal Lines	Number of the gridlines vertical to the X-axis.	The return value is an integer.

Horizontal Spacing	Spacing between the curve drawing area and the left or right margin of the curve control.	The return value is an integer.
Vertical Spacing	Spacing between the curve drawing area and the top or bottom margin of the curve control.	The return value is an integer.
Background Color	Background color of the curve control.	The return value of the expression or dynamic script is the RGB value of the defined color.
Background Color of Curves	Background color of the curve drawing area of the curve control.	
Color of Vertical Lines	Color of the gridlines vertical to the Y-axis.	
Color of Horizontal Lines	Color of the gridlines vertical to the X-axis.	
Color of Text	Color of the text beside the X-axis and the Y-axis.	
Start Time: (year, month, date, hour, minute, second)	Defines the range of data displayed by the historical trend curves.	The return value is an integer.
End Time: (year, month, date, hour, minute, second)		The return value is an integer.
Historical Record Name	Indicates the name of the historical data record for which the curves are drawn. These curves are displayed for all the data defined in the historical data record. This name is defined when you create the historical data record in the **Project Manager** window.	No dynamic properties.
Redrawing Variable	Name of the redrawing variable of the **Historical Trend Curve** control. The value of this variable is bit. When the value of this variable is 1, the system starts redrawing all the curves covered in the control. After the redrawing is complete, the system automatically resets the value of this variable to 0.	The return value is a string, which is the name of the redrawing variable.
Number of Curves	Defines the number of curves to be displayed (16 the most).	No dynamic properties.

9.3.2.2 Properties of Curves

This section describes the data to which each curve is associated and the color of the curve. In total, 16 curves can be conFiguurd the most.

Property	Description	Remarks on Dynamic Properties
Variable Name	Defines the name of the variable to which the curve is associated. This variable must be defined in the historical data record. If no variable is defined here, then the corresponding curve will not be displayed.	The return value is a string, which is the name of the variable data to which the curve is associated. If a blank string (“”) is returned, it means the curve will not be displayed.
Curve Color	Defines the color of the curve.	The return value of the expression or the dynamic script is the RGB value of

Property	Description	Remarks on Dynamic Properties
		the defined color.

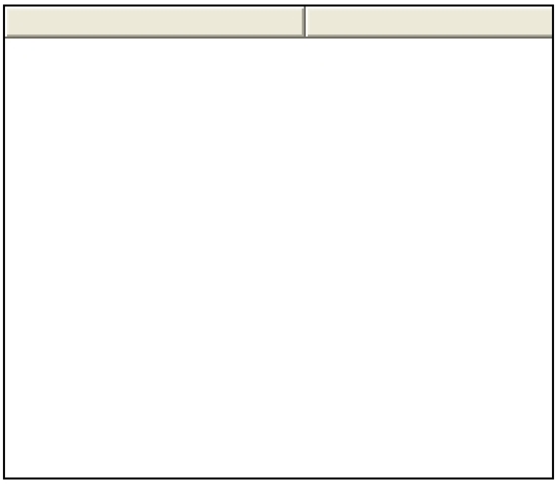
9.3.2.3 Properties of Indicator Lines

Property	Description	Remarks on Dynamic Properties
Allow Indicator Line	Defines whether the control provides indicator line.	The return value is bit, as follows: • 0: Not allow • None 0: Allow
Indicator Line Color	Defines the color of the indicator line.	The return value of the expression or the dynamic script is the RGB value of the defined color.
Time Variable	The time data where the indicator line points to is saved to this variable.	The return value is a string, which is the name of the time variable.
Data Value Variable	The data value of the curve where the indicator line points to is saved to this variable.	The return value is a string, which is the name of the data value variable.

9.4 Control - Historical Data List

The **Historical Data List** control displays all the saved historical data in a list.

In the **Interface Editor** window, click on the **Historical Data List** button ▤ in the tool set on the left side. Move the cursor to the editting area, and you can see the cursor become a cross. Drag the mouse to draw a rectangle, and the historical data list control will be displayed in this rectangle, as shown in Figuur 9.10.



Figuur 9.10

After you draw the historical data list control, select it with a left click, and you will see the **Property List** of the control on the right side of the editting area.

The properties of the historical data list control are described as follows:

Property	Description	Remarks on Dynamic Properties
Number of List Rows	Defines the maximum number of rows to be displayed in the historical data list. The extra rows of historical data will not be displayed.	No dynamic properties.
Height Between List Items	Defines the height between the list items.	No dynamic properties.
Display Field	Defines the field to be displayed in the data list. It can be defined by the designer to satisfy special needs. The default display field is the time the data is collected. Note: Click in the blank area on the right side of the display field, and a list of historical data variables will be displayed. All the data in the historical variable list must be defined in the historical data record created in the **Project Manager** window. In other words, all the data defined in the historical data record will be displayed in the list. You can check the variables which you want to display in the historical data list, and you can use the **Move Up** and **Move Down** buttons to adjust the locations of the selected variables.	No dynamic properties.
Redrawing Variable	Name of the redrawing variable of the **Historical Data List** control. The value of this variable is bit. When the value of this variable is 1, the system re-accesses the historical data and redraws the historical data list control. After the redrawing is complete, the system automatically resets the value of this variable to 0.	No dynamic properties.
Historical Record Name	Indicates the name of the historical data record for which the historical data list is drawn. The historical data list displays all the data defined in the historical data record. This name is defined when you create the historical data record in the **Project Manager** window.	No dynamic properties.
Start Time: (year, month, date, hour, minute, second)	Defines the range of data to be displayed by the historical data list control, as follows: • The start time is all 0 while the end time is not all 0: to display all the data records ending at the end time. • The start time is not all 0 while the end time is all 0: to display all the data records starting from the start time. • The start time is all 0 and the end time is all 0: to display all the data defined in the historical data record. • The start time is not all 0 and the end time is not all 0: to display the data records starting from the start time and ending at the end time.	The return value is an integer.
End Time: (year, month, date, hour, minute, second)		

9.5 Historical Data Inquiry

Besides dislaying the historical data with the **Historical Trend Curve** control and the **Historical Data List** control, EASY also allows you to query the historical data using system functions. The system functions allow you to query any saved historical data easily.

EASY provides the following two ways for inquiring the historical data:

- Specifing inquiry fields
- Obtaining values of the fields of the selected record in the **Historical Data List** control

No matter which way of the above you choose, you will need to define a inquiry variable for the historical data to be queried. The queried results will be saved to the defined inquiry variable. For details about defining the inquiry variable, please see section 9.2.3 Adding Data.

9.5.1Inquiring Historical Data by Specified Fields

As stated before, the system saves a data defined in a historical data record automatically when the defined saving condition is satisfied. Actually, it can be understood that a record is added in the historical database every time when the system saves a data automatically. This record is composed of fields which are all the data defined in a historical data record. In other words, each field is associated to a historical data and its inquiry variable, where the historical data is the data value to be displayed for the field and the individual inquiry variable is for saving the value of the corresponding field when the data matching the defined condition is searched during the historical data inquiry.

The function of inquiring historical data by specified fields is implemented by the system functions **history_query_all** and **history_query_data**. Both system functions require a parameter **query_var_name**, which stands for the name of the historical data to which the field to be queried is associated. Before calling these two functions, you need to assign an initial value to the inquiry variable to which the field is associated, so as to define the inquiry condition. The function called will then search the historical database based on the defined inquiry condition in the following sequence:

1) Obtain the field to which the historical data defined in the parameter **query_var_name** is associated;
2) Search every record in the database to determine whether the historical value of the field is equal to the value of the corresponding inquiry variable.

   If equal, then the record is consiered matching the inquiry condition, and thus the historical value of each field of the record will be assigned to the individually corresponding inquiry variable.

The difference between the functions **history_query_all** and **history_query_data** is that: **history_query_all** searches all the historical data records in the historical database, while **history_query_data** searches only the specified historical data record.

For details about these two functions, please see section 9.8 System functions for Processing Historical Data. The following of this section will take two examples to further explain how to use these two functions.

Suppose that you have created a historical data record with the name as **run_his**, and that you have added the following data, as shown in th table below, into the record (The data saving conditions are not listed in the table below since they are not involved in the discussion here; you can define these conditions according to your needs.):

Name of Historical Data	Name of Inquiry Variable	Description
test.V	test.V_query	Voltage
test.I	test.I_query	Input
test.P	test.P_query	Power

During the interface configuration, suppose that you have added a **Historical Data List** control and associated it to the historical data record **run_his** (by setting **Historical Record Name** in the control to **run_his**). Suppose that the following historical data are generated after a while of system operation:

Record Number	Voltage	Input	Power
1	10.45	5.6	100.5
2	20.12	10.34	200.3
3	25	12	230.3
4	14.23	8.7	230

To query the historical data record with the input value as **10.34**, you need to do the following:

1) Set the value of the inquiry variable **test.I_query** to which the **Input** field is associated to **10.34**;
2) Call the function history_query_data as follows:

history_query_data("test.I", "run_his").

In which, the first parameter query_var_name specifies the field test.I, namely, the historical data corresponding to the Input field, the second parameter run_his is the name of the historical data record.

After you call this function, the system will search through all the historical data in the historical data record run_his until the first record with the Input value as 10.34 (the second record in this example) is found.

Once found, the system automatically saves the value of individual field to the corresponding inquiry variable.

Therefore, in this example, it will be like this after you call the function **history_query_data**:

test.V_query=20.12, test.I_query=10.34, test.P_query=200.3.

9.5.2Obtaining Value for Fields of a Selected Historical Record

After you select a historical data record in the **Historical Data List** control, you can call the function **hislist_query_data** to save the value of the individual field to the corresponding inquiry variable.

For details about this function, see section 9.8 System Variables for Processing Historical Data.

9.6 Downloading Historical Data

Based on your configuration settings, the historical data will be saved into the HMI internal flash or the CF card.

The system generates a file for each historical data record per day. These files are named in the following format: LCHistest200608200700.dat(LCHis+Name of the historical data record+year+month+date+Serial NO.dat. You can download these files to a thumb drive or a PC for futher inquiry.

At present, EASY supports the following two downloading methods:

- Downloading to the thumb drive through the system function
- Downloading to the PC through FTP

9.6.1Downloading Historical Data to a Thumb Drive Using System Function

You can call the system function **sys_history_download()** to export all the historical data files saved in the HMI internal flash or the CF card to a thumb drive.

For details about this function, please see section 9.8 System Functions for Processing Historical Data.

9.6.2Downloading Historical Data to a PC Through FTP

You can selectively download the historical data files saved in the HMI internal flash or the CF card to a PC through FTP.

To do this, you need to log in to the HMI through the FTP client tool. Both the login username and the password are **EASY**.

Once logging in successfully, you will see the following directories:

- **data**: contains the user data saved in the HMI internal flash.

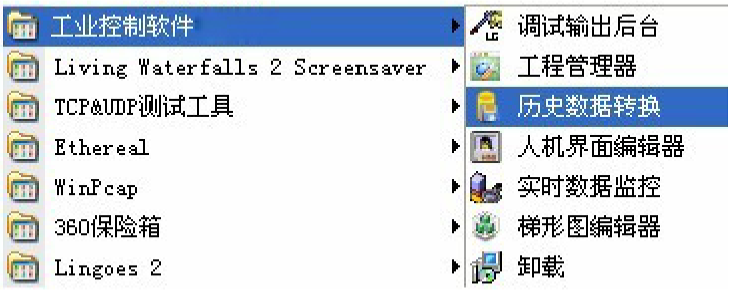
  The user data refers to the data you have saved previously according to your needs.
- **hisdata**: contains the historical data saved in the HMI internal flash.
- **cfcard**: contains the user data and the historical data saved in the CF card.

You can select from the above directories what data to download to your PC.

9.7 Data Format Conversion

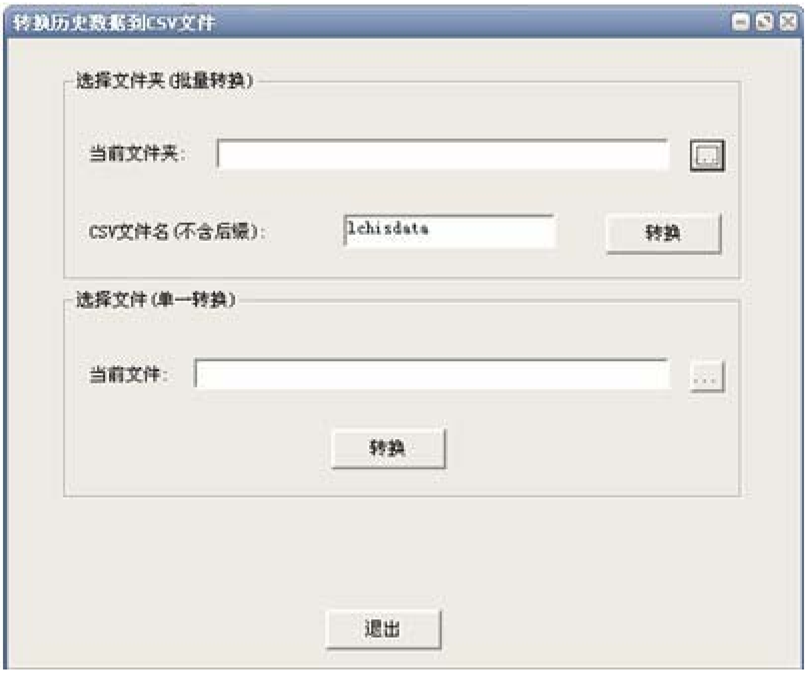
You cannot immediately view the historical data files downloaded to the thumb drive or the PC, because they are in the internal binary format (marked with the extension name **.dat**).

To solve this issue, EASY provides the **Historical Data Conversion** tool to convert these **.dat** files to **.csv** files, which can be recognized by EXCEL. As shown in Figuur 9.11, the installed EASY package contains the **Historical Data Conversion** tool.



Figuur 9.11

Click on **Historical Data Conversion**, and you can see a dialog box as shown in Figuur 9.12.



CSV文件名(不含后缀): lchisdata

Figuur 9.12

The Historical Data Conversion tool works in two ways, whold folder conversion and single file conversion, as described below:

- **Whold Folder Conversion**

For a folder which contains multiple historical data files, you can use this tool to combine all the historical data in all these files into one CSV file, and then open it with EXCEL.

To convert all the files in a folder, do as follows:

1) In Figuur 9.12, click on the button □ in the **Select a Folder** area, and select the folder you want to convert.
2) Set the name for the CSV file after conversion.
   The default file name is **lchisdata**.
3) Click on the **Convert** button.
   All the historical data files in the selected folder will be converted into one CSV file. This CSV file will be saved in the same directory as the selected folder.

- **Single File Conversion**

For a single historical file, you can also convert it into a CSV file and open it with EXCEL.

To convert a single file, do as follows:

1) In Figuur 9.12, click on the button □ in the **Select a File** area, and select the file you want to convert.

2) Click on the **Convert** button.

The selected historical data file will be converted into a CSV file. This CSV file will be saved in the same path as the selected file.

9.8 System Functions for Processing Historical Data

9.8.1 sys_history_download

Original Function: int **sys_history_download**()

Function Description: To download historical data historical data from the HMI internal flash or the CF card to a thumb drive.

Return Value: 0 Failed

1 Successful

Parameter: None.

Example: sys_history_download()

9.8.2 history_query_all

Original Function: int **history_query_all**(**char ****query_var_name*)

Function Description: To query historical data records from all historical databases by the specified fields. For details, see section 9.5.1 Inquiring Historical Data by Specified Fields.

Return Value: 0: Failed

1: Successful

Parameter: *query_var_name*: Name of the historical data with which the field to be searched is associated.

Example: history_query_all("test.query_data1")

9.8.3 history_query_data

Original Function: int **history_query_data**(**char ****query_var_name,* **char ****history_name*)

Function Description: To query historical data records from the defined historical database by the specified fields. For details, see section 9.5.1 Inquiring Historical Data by Specified Fields.

Return Value: 0: Failed

1: Successful

Parameters: *query_var_name*: Name of the historical data with which the field to be searched is associated.

history_name: Name of the historical data record.

Example: history_query_data("test.query_data1", "his")

9.8.4 hislist_query_data

Original Function: int **hislist_query_data**(**char** ***window_name,* **char** ***widget_name*)

Function Description: To query the values of the various fields of the record currently selected in the **Historical List** control.

Return Value: 0: Failed

1: Successful

Parameters: *window_name*: Name of the window where the **Historical List** control is located.

widget_name: Name of the graphic component of the **Historical List** control.

Example: history_query_data("main_pic", "hislist1")

9.8.5 hislist_delete_data

Original Function: int **hislist_delete_data**(**char** ***window_name,* **char** ***widget_name*)

Function Description: To delete the currently selected record from the **Historical List** control.

Return Value: 0: Failed

1: Successful

Parameters: *window_name*: Name of the window where the **Historical List** control is located.

widget_name: Name of the graphic component of the **Historical List** control.

Example: history_delete_data("main_pic", "hislist1")

Chapter 10 Alarms

To ensure safe production and operation on industrial sites, the alarm and event generating and recording are crucially important.

EASY provides a powerful alarm and event system, which is described in details below.

10.1 Overview

Alarms are generated by the system automatically when the values of the specified settings exceed the pre-defined values. They can be taken as warnings agains serious accidents.

Take the oil tank in the refinery for example. If no limit is pre-defined for the oil level during the oil loading, then no alarms will be generated by the system to warn the operators. In this case, overloading might happen, which might cause very serious consequencies. In contrarary, if the oil level limit is pre-defined to trigger system alarms when necessary, measures can be taken accordingly to prevent the accident.

System alarms are processed as follows: When alarms or events occur, the system saves the alarm or event related information in the memory cache (the size of which can be defined). EASY processes alarms and events according to the first-in-first-out principle. That is to say, only the most recent alarm and event information is saved in the memory.

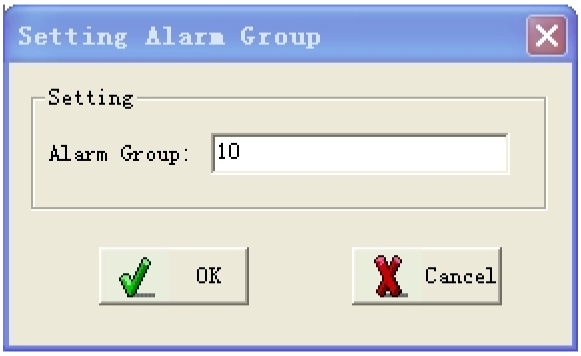
You can view the alarm and event information in the alarm window provided in the HMI.

10.2 Configuring Size of Alarm Cache

The alarm cache is part of the system memory especially for saving alarm information. The size of the alarm cache is configurable.

To conFiguur the size of the alarm cache, do as follows:

1) Select **Alarm Configuration** on the left side of the **Project Manager** window, and right-click on it.
2) Select **Alarm Configuration** from the right-click menu.
   You will see a dialog box as shown in Figuur 10.1.



Alarm Group: 10

Figuur 10.1

The size of the alarm cache is calculated by the number of alarm information records that can be saved in the cache. When the alarm records take more than the defined space, the previous alarm information will be replaced by the new record.

10.3 Alarm Groups

10.3.1 Overview

To have the alarm information displayed, you must define the data for which alarms are to be displayed, alarm conditions, and alarm contents. EASY allows you to add alarm groups and then add alarm data into alarm groups.

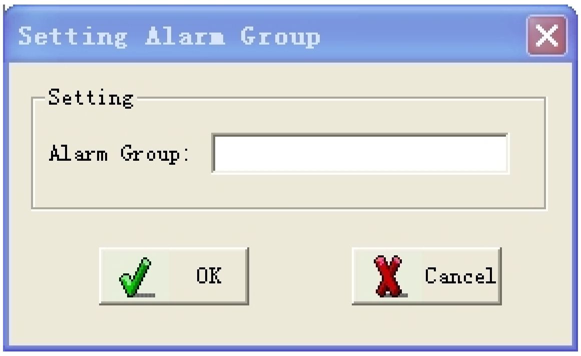
You can define one or more alarm groups, and you can add one or more data for which alarms are to be displayed into each alarm group. When adding alarm data into an alarm group, you need to specify the data name and define the alarm condition and alarm contents as well. When the defined alarm condition is satisfied, the alarm information will be automatically saved into the alarm cache.

During the interface configuration, you can use the **Alarm Window** control to display the alarm information. Each control can be associated with one alarm group only. The **Alarm Window** control associated with the alarm group can display all the alarm data defined in the alarm group.

10.3.2 Adding an Alarm Group

To add an alarm group, do as follows:

1) Select **Alarm Configuration** on the left side of the **Project Manager** window, and right-click on it.
2) Select **Add an Alarm Group** on the right-click menu.
   And you will see the following dialog box as shown in Figuur 10.2.



Figuur 10.2

3) Enter the name of the alarm group, and click on **OK**.

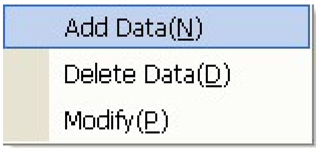
10.3.3 Adding Alarm Data

After you add an alarm group, you need to add alarm data into it.

To add the alarm data, do as follows:

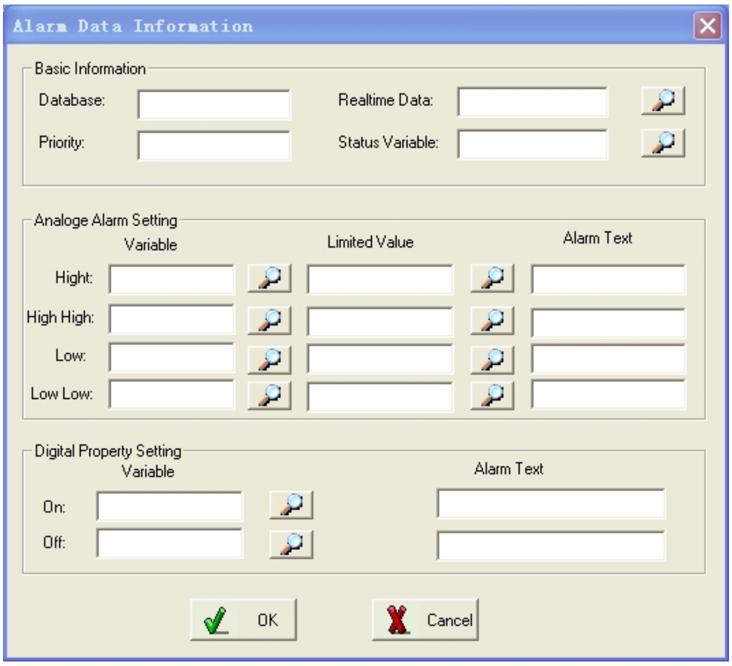
1) Select an alarm group on the left side of the **Project Manager** window, and right-click in the list pane on the right side of the window.

   You will see a right-click menu as shown in Figuur 10.3.



Figuur 10.3

2) Select **Add Data**, and you will see the dialog box as shown in Figuur 10.4.



Figuur 10.4

The **Alarm Data Information** dialog box is composed of three parts:

- Basic Information
- Analog Alarm Configuration
- Digital Alarm Configuration

See the following sections for detailes.

10.3.3.1 Basic Information Configuration

This section defines the basic alarm information, which are valid for both analog and digital alarms.

The configuration parameters for basic alarm information are described as follows:

- **Database Name**: Name of the database from where the alarm data are added from.
- **Real-Time Data Name**: Name of the alarm data.
- **Priority**: Priority of the alarm.
- **Status Variable**: An int variable, which defines the current state of the alarm data. The value of this variable can be one of the following:
  - ➢ 0 --- Normal
  - ➢ 1 --- Confirmed
  - ➢ 2 --- Restored
  - ➢ 3 --- Alarming

The above four states of **Status Variable** are described as follows:

- **Normal**: The value of **Status Variable** is within the defined value range, and no alarms have ever been generated.
- **Confirmed**: The alarm is confirmed. This state indicates that the generated alarm is already confirmed and processed. However, this alarm state still stays in the system.
- **Restored**: The value of the variable is restored to the defined value range, and no alarms will be generated.
- **Alarming**: The value of the variable matches the alarm condition and an alarm is being generated.

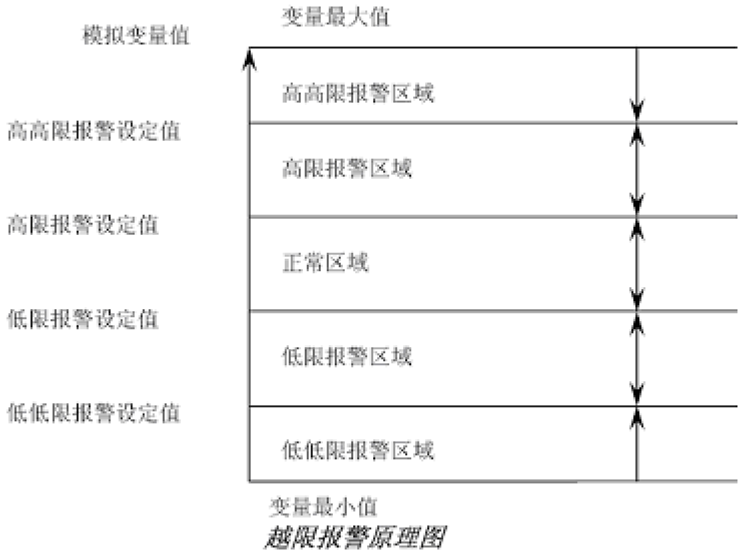
If no status variable is defined, it means that you do not need to obtain the alarm status for the alarm data.

10.3.3.2 Analog Alarm Configuration

The configuration in this section is valid only when the alarm data is analog. The analog alarm data refers to variables of all types in the real-time database except the digital data, including int, real type, and float variables.

Analog alarms are mostly over-limit alarms. Please see below for details.

Over-limit alarms are those generated by the system when the value of the analog alarm data goes beyond the high or low alarm threshold. Over-limit alarms have altogether four alarm thresholds: low low (LL), low (L), high (H), and high high (HH), as shown in Figuur 10.5.

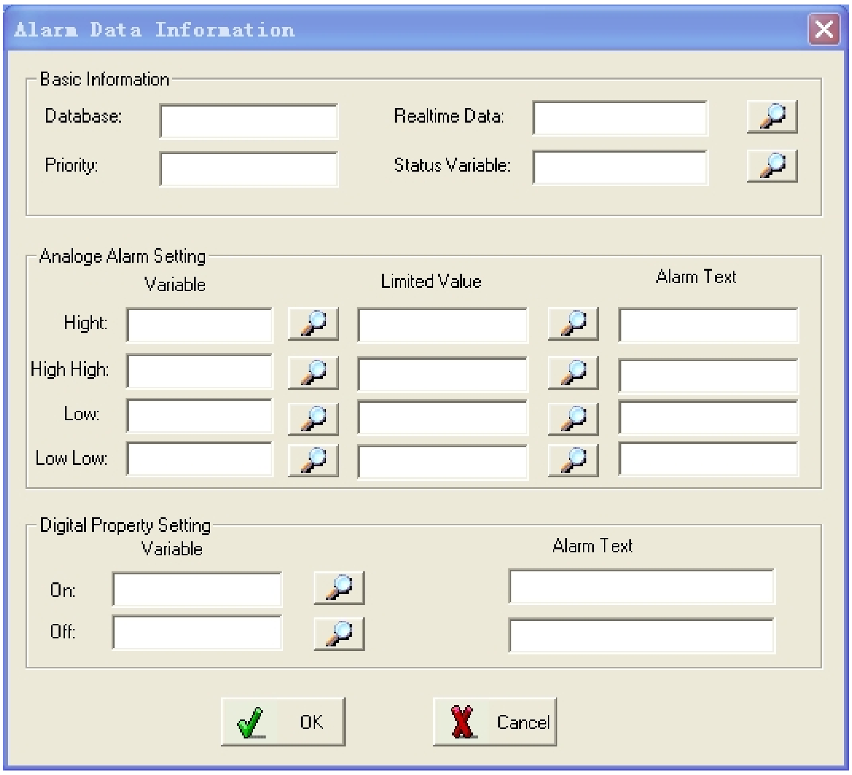


Figuur 10.5

As the value of a variable changes, alarms are generated by the system whenever a threshold is exceeded. However, from the standpoint of the variable, only one over-limit alarm will be generated at one time, since the value of it can only exceed one threshold at one time. For example, if the value of a variable exceeds the HH threshold, only the HH threshold alarm will be generated, while no high threshold alarm will be generated at the same time.

However, in the case of the value of a variable exceeding two thresholds, it depends on whether the two thresholds are of the same type. If they are, then no new alarm will be generated, but it does not mean that the generated alarm will be restored. If not, then the generated alarm will be restored and a new alarm will be generated.

For the four types of over-limit alarms, you can define only one, a few, or all of them. As shown in Figuur 10.6, you can define **Allow Variable**, **Threshold Variable**, and **Alarm Text**, which are described below in details.



Figuur 10.6

- **Allow Variable**: defines whether to allow the system to generate the alarm. You can specify a variable for this. When the value of this variable is 1, the system will generate an alarm when the alarm condition is satisfied. When the value is 0, no alarms will be generated in the system. If no **Allow Variable** is specified, then alarms will be generated whenever necessary.
- **Threshold Variable**: sets threshold values for the alarm variable to be defined. The threshold values are displayed through variables.
- **Alarm Text**: defines the descriptive text of an alarm to be displayed when the alarm condition is satisfied. The alarm text should not be more than 32 bytes.

10.3.3.3 Digital Alarm Configuration

The configuration in this section is valid only for digital alarm data (namely, bit variables).

Digital alarms have the following two states:

- **Open**: An alarm is generated when the variable value becomes 1.
- **Close**: An alarm is generated when the variable value becomes 0.

The digital alarm properties are composed of two columns: **Allow Variable** and **Alarm Text**, which are described in details below:

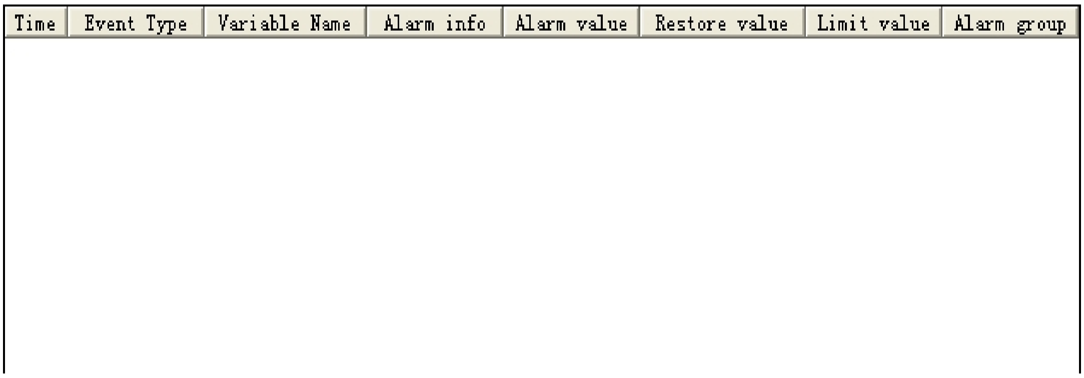
- **Allow Variable**: defines whether to allow the system to generate the alarm. You can specify a variable for this. When the value of this variable is 1, the system will generate an alarm when the alarm condition is satisfied. When the value is 0, no alarms will be generated in the system. If no **Allow Variable** is specified, then alarms will be generated whenever necessary.

- **Alarm Text**: defines the descriptive text of an alarm to be displayed when the alarm condition is satisfied. The alarm text should not be more than 32 bytes.

10.4 Control - Alarm Window

The **Alarm Window** control displays the alarm information in a list.

In **Interface Editor**, click on the **Alarm Window** button ! in the tool set on the left side. Move the mouse to the editting area on the right, and you can see the mouse become a cross. Drag the cross to draw a rectangle. The **Alarm Window** control will be displayed in this rectangle, as shown in Figuur 10.7.



Time	Event Type	Variable Name	Alarm info	Alarm value	Restore value	Limit value	Alarm group

Figuur 10.7

After you draw the **Alarm Window** control, select the list with a left click, and you will see the **Property List** pane on the right side of the editting area, which lists properties of the control.

The properties of the **Alarm Window** control are described in the table below.

Property	Description	Remarks on Dynamic Properties
Rows of Alarms	Defines the maximum rows of alarms to be displayed in the alarm window. When the system generates more than the defined maximum rows of alarms, the previous rows of alarms will be replaced.	No dynamic properties.
Confirmed Color	Defines the color for the **Confirmed** alarms.	The return value of the expression or dynamic script is the RGB value of the defined colors.
Restored Color	Defines the color for the Restored alarms.	
Alarm Color	Defines the color of the alarming alarms.	
Display Field	Defines the fields to be displayed in the **Alarm Window** list.	No dynamic properties.
Height Between	Defines the height between list items.	No dynamic properties.

Property	Description	Remarks on Dynamic Properties
List Items		
Display Mode	Two display modes are available: • Displaying all historical alarm information: displays all historical alarm information regarding alarming, alarm confirmation, and alarm restoration. • Displaying alarming information: displays only the information regarding the alarming data.	No dynamic properties.

10.5 Alarm Confirmation

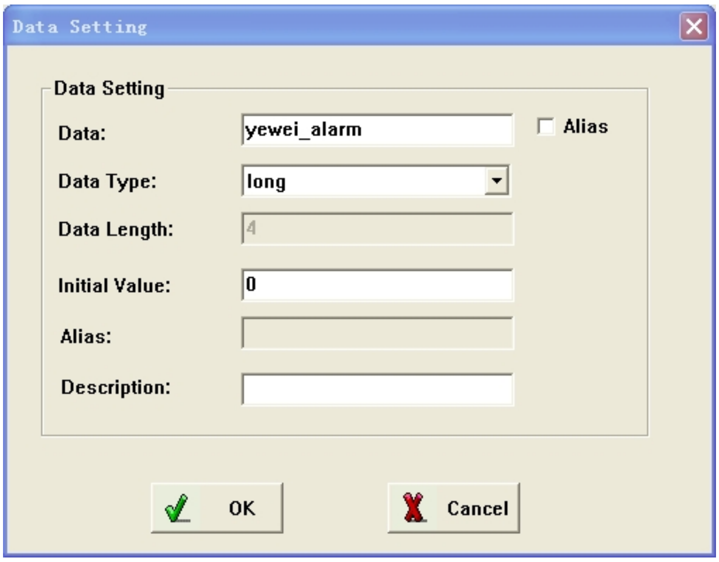
After an alarm is generated, you can call the system function **alarm_confirm** to confirm the alarm, to indicate that you are aware of or have processed the alarm.

Please note that the function **alarm_confirm** confirms only the alarm record selected in the **Alarm Window** control. For more details on this function, please see section 10.7 System Function for System Alarms.

10.6 Configuration Examples

Suppose that you want to set alarm thresholds for the liquid level variable as follows: H = 750, HH = 800, L = 150, and LL = 50. You can set the thresholds as follows:

1. Add an int variable in the real-time database. Set the variable name to **yewei_alarm**, the variable type to **long**, the data length to **4**, and the initial value to **0**, as shown in the **Data Configuration** dialog box in Figuur 10.8.



Data Setting

Data Setting

Data: yewei_alarm

□ Alias

Data Type: long

Data Length: 4

Initial Value: 0

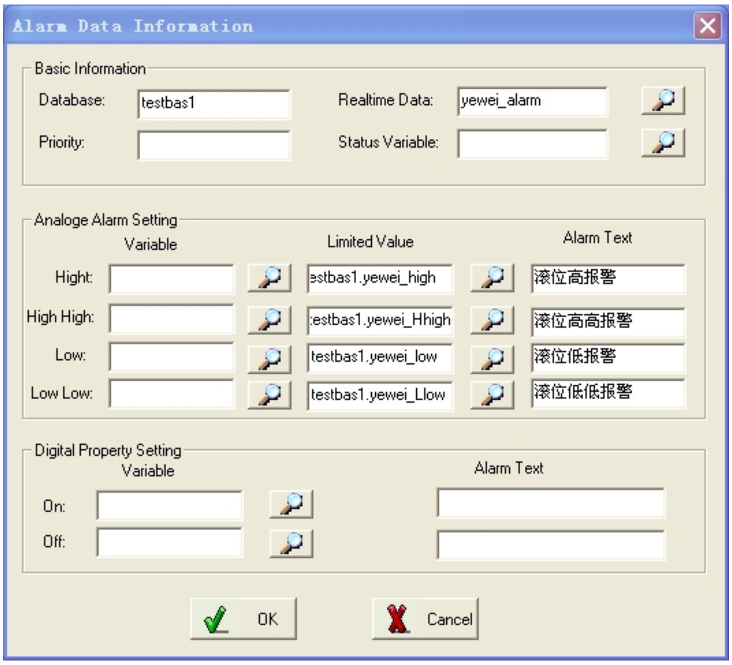
Alias:

Description:

Figuur 10.8

Add other variables **yewei_Hhigh**, **yewei_high**, **yewei_low**, and **yewei_Llow** in the same way, and set their individual initial value to 800, 750, 150, and 50.

2. Add an alarm group **Alarm1** in **Aarm Configuration**. Right-click on the alarm group and open the **Alarm Data Configuration** dialog box. ConFiguur the alarm conditions as shown in Figuur 10.9, and then click on **OK**.

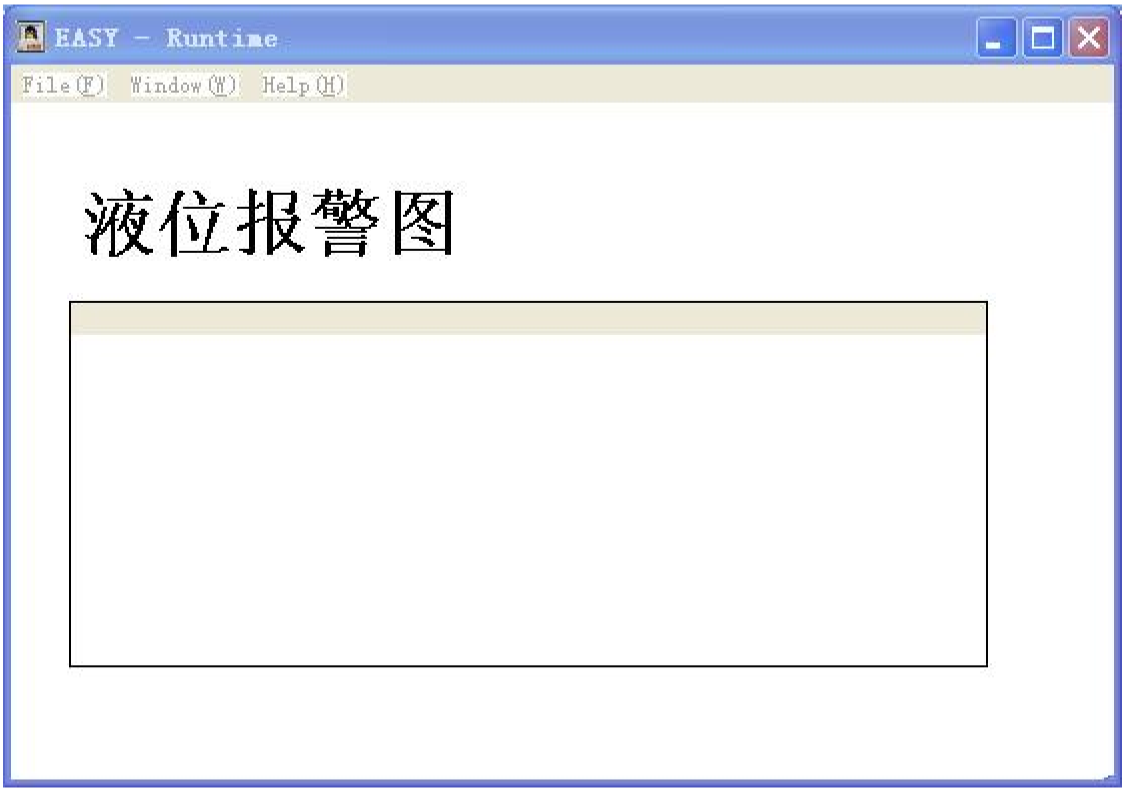


Database: testbas1
Realtime Data: yewei_alarm

	Variable	Limited Value	Alarm Text
Hight:		estbas1.yewei_high	滚位高报警
High High:		:estbas1.yewei_Hhigh	滚位高高报警
Low:		testbas1.yewei_low	滚位低报警
Low Low:		testbas1.yewei_Llow	滚位低低报警

Figuur 10.9

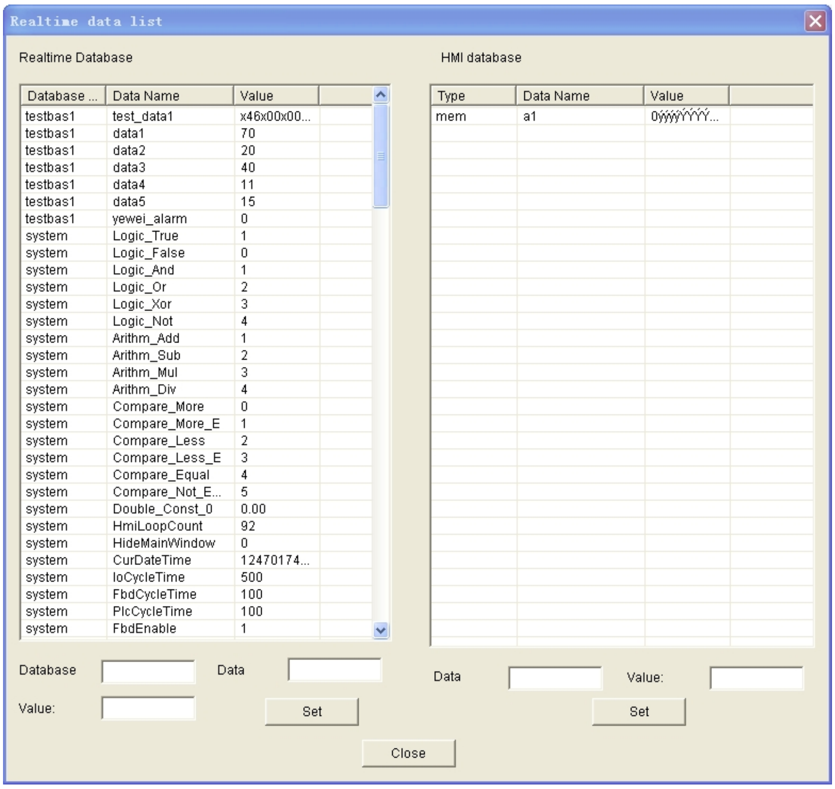
3. Open **Interface Editor**, click on the **Alarm Window** button in the tool set on the left to draw an alarm window in the editting area on the right. Set the alarm window properties according to your needs and then save the interface. When you are back in the **Project Manager** window, compile a project and then select **Offline Simulation**, as shown in Figuur 10.10.



液位报警图

Figuur 10.10

4. During the offlline simulation, select **Window** and then **Real-Time Data Display**, and you will see a dialog box as shown in Figuur 10.11.



Realtime Database

Database ...	Data Name	Value
testbas1	test_data1	x46x00x00...
testbas1	data1	70
testbas1	data2	20
testbas1	data3	40
testbas1	data4	11
testbas1	data5	15
testbas1	yewei_alarm	0
system	Logic_True	1
system	Logic_False	0
system	Logic_And	1
system	Logic_Or	2
system	Logic_Xor	3
system	Logic_Not	4
system	Arithm_Add	1
system	Arithm_Sub	2
system	Arithm_Mul	3
system	Arithm_Div	4
system	Compare_More	0
system	Compare_More_E	1
system	Compare_Less	2
system	Compare_Less_E	3
system	Compare_Equal	4
system	Compare_Not_E...	5
system	Double_Const_0	0.00
system	HmiLoopCount	92
system	HideMainWindow	0
system	CurDateTime	12470174...
system	IoCycleTime	500
system	FbdCycleTime	100
system	PlcCycleTime	100
system	FbdEnable	1

HMI database

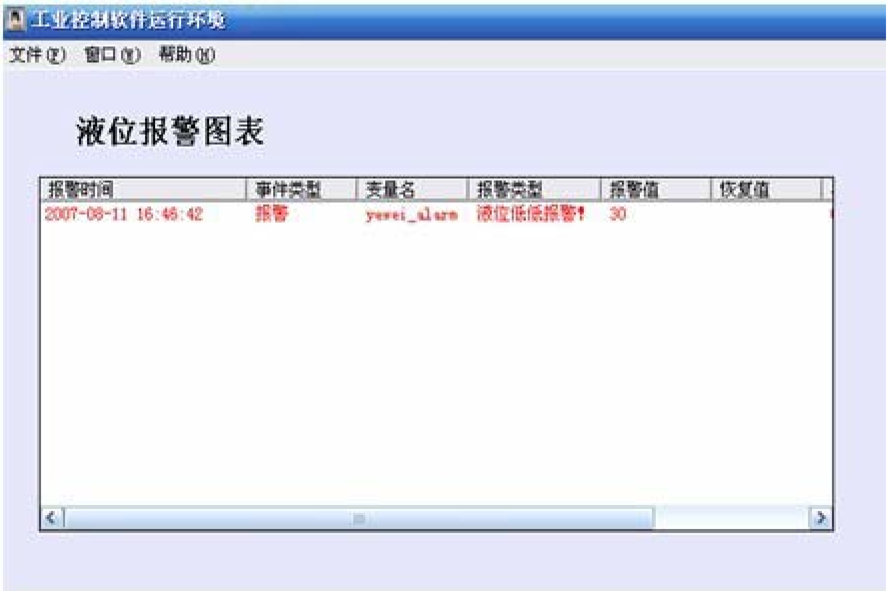
Type	Data Name	Value
mem	a1	0ýýýÝÝÝÝ...

Database
Data
Value:

Data
Value:

Figuur 10.11

Change the value of the variable **yewei_alarm** to **30**, and you will see an alarm record displayed in the alarm window, as shown in Figuur 10.12.

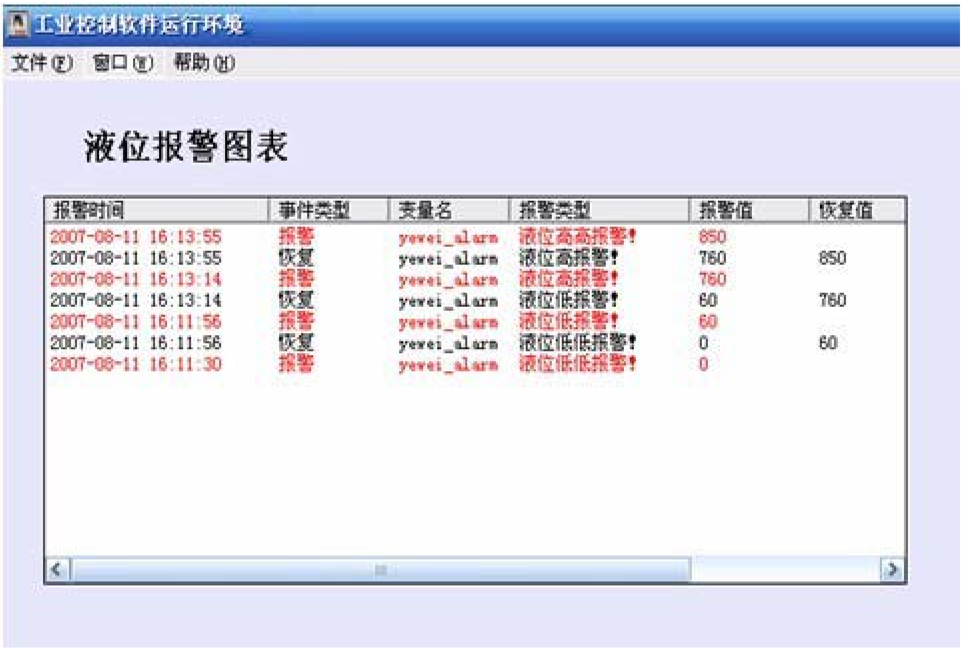


液位报警图表

报警时间	事件类型	变量名	报警类型	报警值	恢复值
2007-08-11 16:46:42	报警	yewei_alarm	液位低低报警!	30	

Figuur　10.12

Change the value to 60, 760, and then 850, and a series of alarms will be generated and displayed in the alarm window, as shown in Figuur 10.13.



液位报警图表

报警时间	事件类型	变量名	报警类型	报警值	恢复值
2007-08-11 16:13:55	报警	yewei_alarm	液位高高报警!	850	
2007-08-11 16:13:55	恢复	yewei_alarm	液位高报警!	760	850
2007-08-11 16:13:14	报警	yewei_alarm	液位高报警!	760	
2007-08-11 16:13:14	恢复	yewei_alarm	液位低报警!	60	760
2007-08-11 16:11:56	报警	yewei_alarm	液位低报警!	60	
2007-08-11 16:11:56	恢复	yewei_alarm	液位低低报警!	0	60
2007-08-11 16:11:30	报警	yewei_alarm	液位低低报警!	0	

Figuur　10.13

As shown above,

- When the variable value is less than or equal to 50, an LL over-limit alarm is generated and displayed;
- When the value is bigger than 50 but less than or equal to 150, the LL alarm is restored, and an L over-limit alarm is generated and displayed;
- When the value is bigger than 150 but less than 750, the L alarm is restored and no

other alarms will be generated;

- When the value is bigger than or equal to 750 but less than 800, a H over-limit alarm is generatd and displayed;
- When the value is bigger than or equal to 800, the H over-limit alarm is restored and an HH over-limit alarm is generated and displayed.

10.7 System Function for System Alarms

10.7.1 alarm_confirm

Original Function: int **alarm_confirm**(**char** ***window_name,* **char** ***widget_name*)

Function Description: To confirm the alarm record currently selected in the **Alarm Window** control.

Return Value: 0: Failed

1: Successful

Parameters: *window_name*: Name of the window where the **Alarm Window** control is located.

widget_name: Name of the graphic component in the alarm window.

Example: alarm_confirm("main_pic", "alarmwnd1")

Chapter 11 Device Configuration

11.1 Overview

During actual operation, you might often need to link various on-site devices, such as PLC or I/O modules. EASY implements the modularized and layered design, which makes it easier for device management and device communication.

Device management of EASY is achieved in the following three layers:

- **Link Layer**: The links here refer to communication links, such as serial ports and network links. The links are categorized into master links and slave links (backup links). Usually, the master link takes the responsibility for communication. When the master link stops working (for example, timeout occurs), the backup link carries on for data transfer.
- **Device Layer**: The devices here refer to individual IO devices, such as the IO data acquisition module, grabber, or PLC. Multiple devices can be conFiguurd on one link.
- **Device Data Layer**: The device data refers to individual data stored in each IO device, such as the IR or HR register in Omron PLC. Each register is linked to a specific real-time database, which achieves the data exchange between the device and the linked real-time database.

11.2 Device Management

Considering that the device management of EASY is implemented in three layers, the device configuration needs to be implemented following the three steps below:

1. Add a communication link.
2. Add devices for each communication link.
3. Add data for each device.

The following sections will describe these three steps in details.

11.2.1 Adding Communication Links

At present, EASY supports the following types of links:

- **Serial Port**: refers to the links which use the serial port for communication.
- **Common Link**: refers to the links which use the virtual link or the internal bus for

communication.

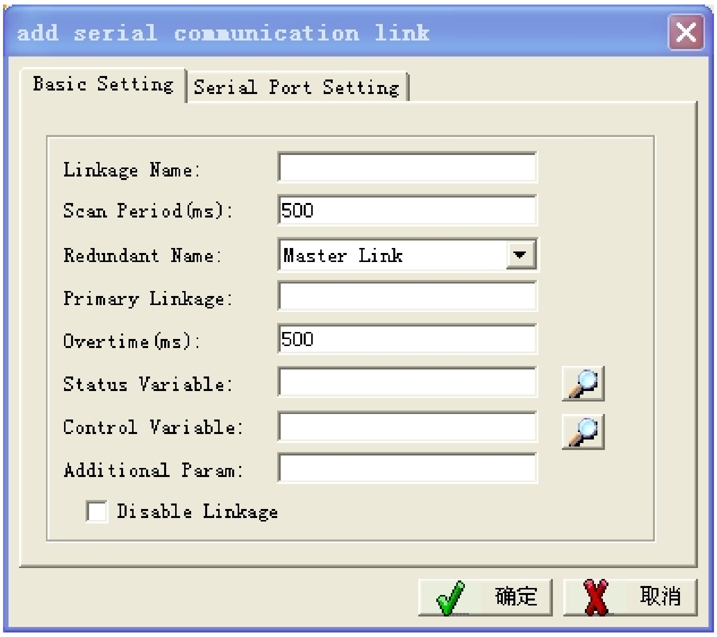
- **TCP Network Client**: refers to the links which use the TCP/IP network for communication and the HMI as the TCP client.
- **TCP Network Server**: refers to the links which use the TCP/IP network for communication and the HMI as the TCP server.

Each type of communication links has their own configuration items depending on their specific characteristics. Meanwhile, all of the links share some common configuration items, which are conFiguurd in **Basic Link Information**.

Let’s take a look at the basic link information configuration first and then the characteristic configuration for each link.

11.2.1.1 Configuring Basic Link Information

To conFiguur the basic link information while adding a communication link, click on the **Basic Link Info** tab in the link configuration dialog box as shown in Figuur 11.1.



Linkage Name:
Scan Period(ms): 500
Redundant Name: Master Link
Primary Linkage:
Overtime(ms): 500
Status Variable:
Control Variable:
Additional Param:
□ Disable Linkage

Figuur 11.1

The configuration parameters in Figuur 11.1 are described as follows:

- **Link Name**: defines the name of the communication link you are going to add.
- **Scan Interval**: defines the interval for scanning this communication link. The system scans all the data stored in all the devices conFiguurd for this link based on the interval defined here. (unit: ms)
- **Redundancy Type**: is for the redundancy configuration for the link. The available redundancy types are **Master Link** and **Backup Link**. For details, see the Redundancy System section. If the added link does not support redundancy, you

need to set **Redundancy Type** to **Master Link**.

- **Master Link Name**: is for the redundancy configuration for the link. This parameter is valid only when **Redundancy Type** is **Backup Link**. It defines the name of the master link to which the backup link is associated. For details, see the Redundancy System section.
- **Timeout Time**: defines how long before the communication link times out. During the communication between the HMI and the link, the link is considered timed-out if the HMI does not get reply from the devices on this link after the period of time specified here. (unit: ms)
- **Status Variable**: must be an int variable. The value of this variable can be one of the following based on the current communication state of the link:
  - ➢ 0 -- Normal
  - ➢ 1 -- Port not available
  - ➢ 2 -- Timeout
  - ➢ 3 -- Waiting (Default communication state of the backup link. When the master link stops working, the status of the backup link changes from **Waiting** to **Normal**.)
  - ➢ 4 -- Suspended (When the value of **Control Variable** becomes **1**, the link scanning is suspended, which means the HMI stops communicating with the devices on the link.

If no status variable is defined, it means that you do not need to get the communication status of the link.

- **Control Variable**: Must be an int variable. The value can be one of the following:
  - ➢ 0 -- Activated
  - ➢ 1 -- Suspended (namely, the link scanning is suspended, which means the HMI stops communicating with the devices on the link)

If no control variable is defined, it means that the link is always activated.

- **Additional Parameter**: defines the additional parameter for the link.
- **Disable the link**: When you check this option, the link will be disabled, which means the system will not load the link and the devices conFiguurd for the link.

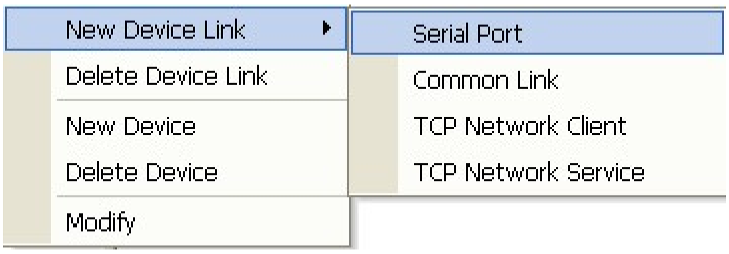
11.2.1.2 Adding a Serial Port

Serial ports refer to the links which use the serial port for communication.

To add a serial port, do as follows:

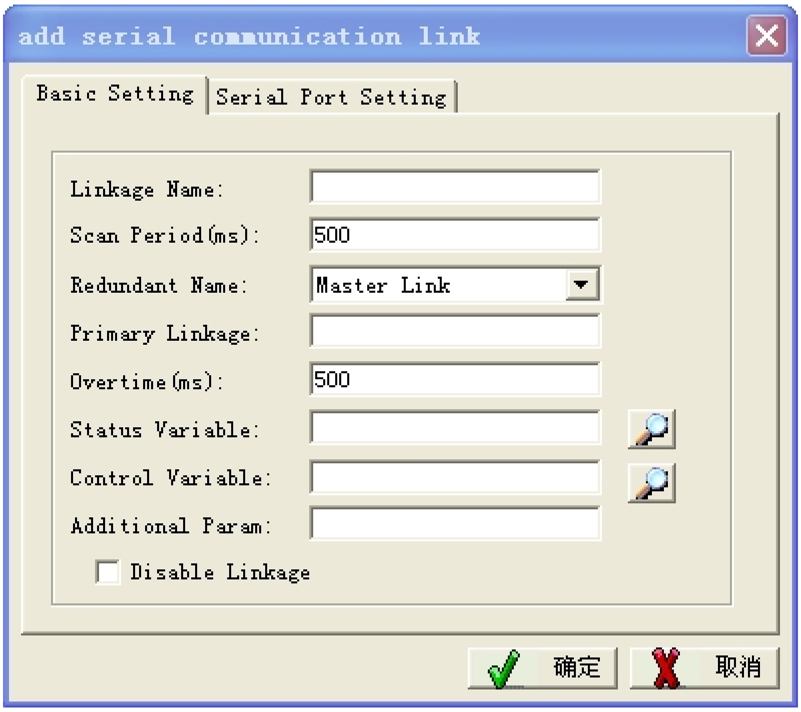
1) Select **Device Configuration** on the left side of the **Project Manager** window, and then right-click on it.

   You will see a right-click menu as shown in Figuur 11.2.



Figuur 11.2

2) Select **Serial Port**, and you will see a dialog box as shown in Figuur 11.3.



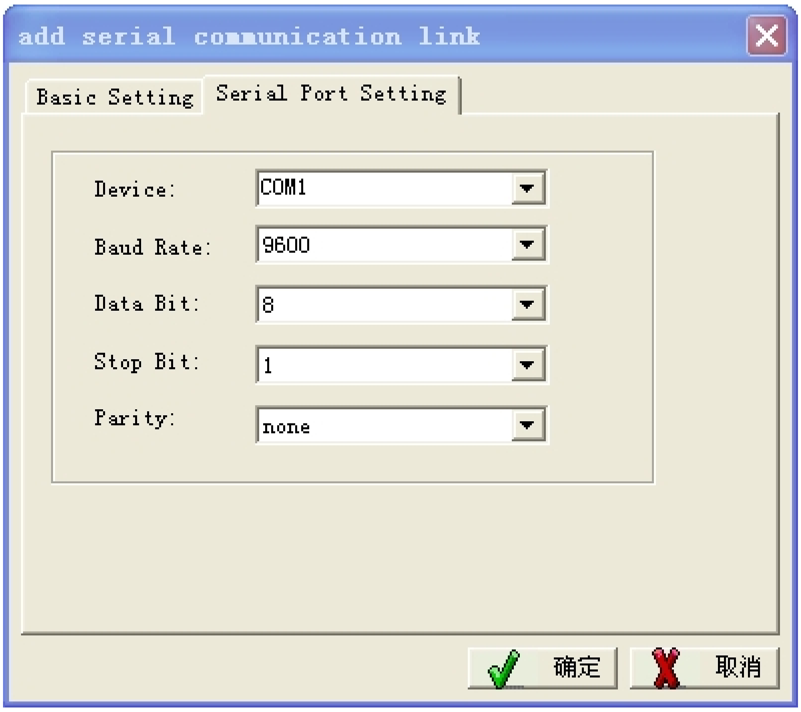
Linkage Name:
Scan Period(ms): 500
Redundant Name: Master Link
Primary Linkage:
Overtime(ms): 500
Status Variable:
Control Variable:
Additional Param:
□ Disable Linkage

Figuur 11.3

3) Click on the **Basic Link Info** tab, and you can conFiguur the basic link information.

For details, see section 11.2.1.1 Configuring Basic Link Information.

4) Click on the **Serial Communication Port Info** tab, and you can conFiguur the parameters for serial port communication as shown in Figuur 11.4.



add serial communication link

Basic Setting | Serial Port Setting

Device: COM1

Baud Rate: 9600

Data Bit: 8

Stop Bit: 1

Parity: none

Figuur 11.4

You can conFiguur **Serial Port Number**, **Baud Rate**, **Data Bit**, **Stop Bit**, and **Verify Mode** of the serial port according to your actual needs.

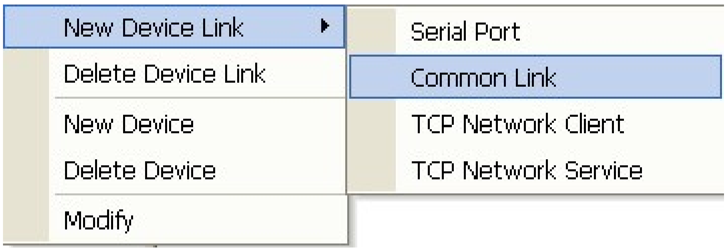
11.2.1.3 Adding a Common Link

Common links refer to the links which use the virtual link or the internal bus for communication. For the virtual link, you can conFiguur virtual IO devices. For details, see section 11.2.2.1 Adding a Device. Regarding the internal bus, it is currently reserved for future expansion since no devices in the system use the internal bus for communication.

To add a common link, do as follows:

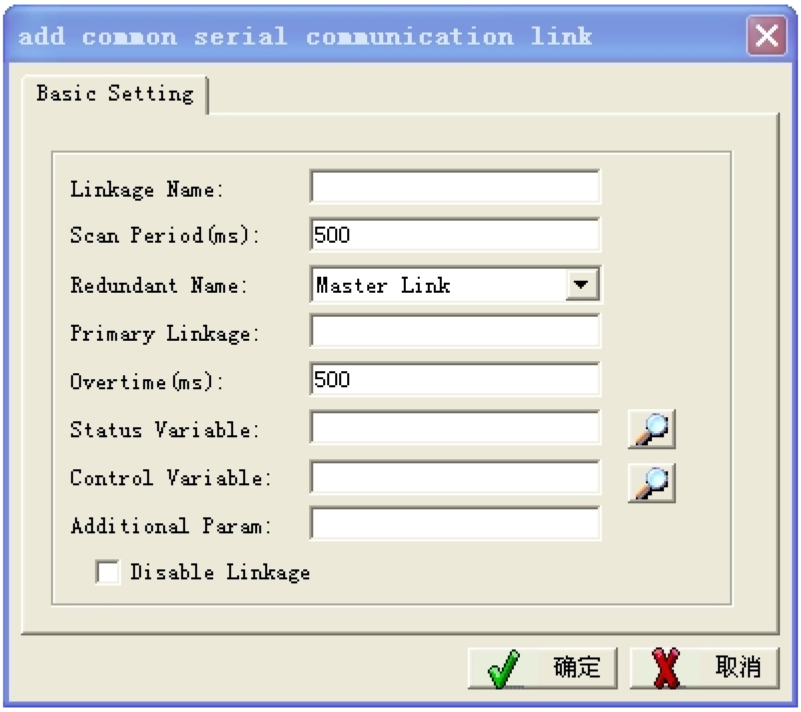
1) Select **Device Configuration** on the left side of the **Project Manager** window and right-click on it.

You will see a right-click menu as shown in Figuur 11.5.



Figuur 11.5

2) Select **Common Link**, and you will see a dialog box as shown in Figuur 11.6.



add common serial communication link

Basic Setting

Linkage Name:

Scan Period(ms): 500

Redundant Name: Master Link

Primary Linkage:

Overtime(ms): 500

Status Variable:

Control Variable:

Additional Param:

□ Disable Linkage

Figuur 11.6

3) Click on the **Basic Link Info** tab, and you can conFiguur the basic link information.

   For details, see section 11.2.1.1 Configuring Basic Link Information. Common links have no characteristic configuration items of their own.

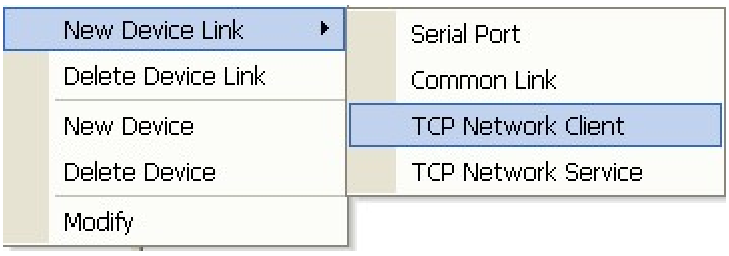
11.2.1.4 Adding a TCP Network Client

The TCP Network Client refers to the link which uses the TCP/IP network for communication and the HMI as the TCP client.

To add a TCP Network Client, do as follows:

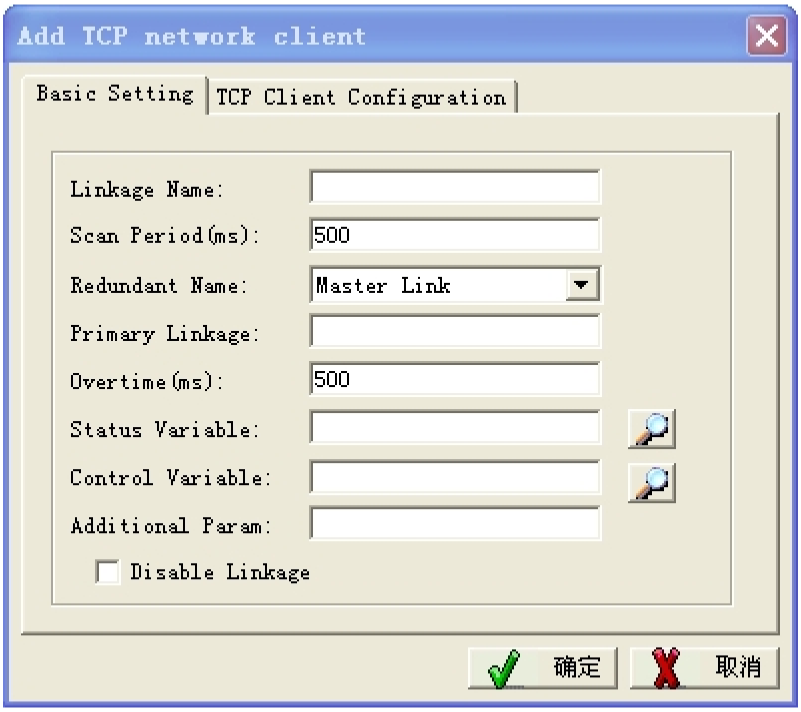
1) Select **Device Configuration** on the left side of the **Project Manager** window and right-click on it.

   You will see a right-click menu as shown in Figuur 11.7.



Figuur 11.7

2) Select **TCP Network Client**, and you will see a dialog box as shown in Figuur 11.8.



Linkage Name:

Scan Period(ms): 500

Redundant Name: Master Link

Primary Linkage:

Overtime(ms): 500

Status Variable:

Control Variable:

Additional Param:

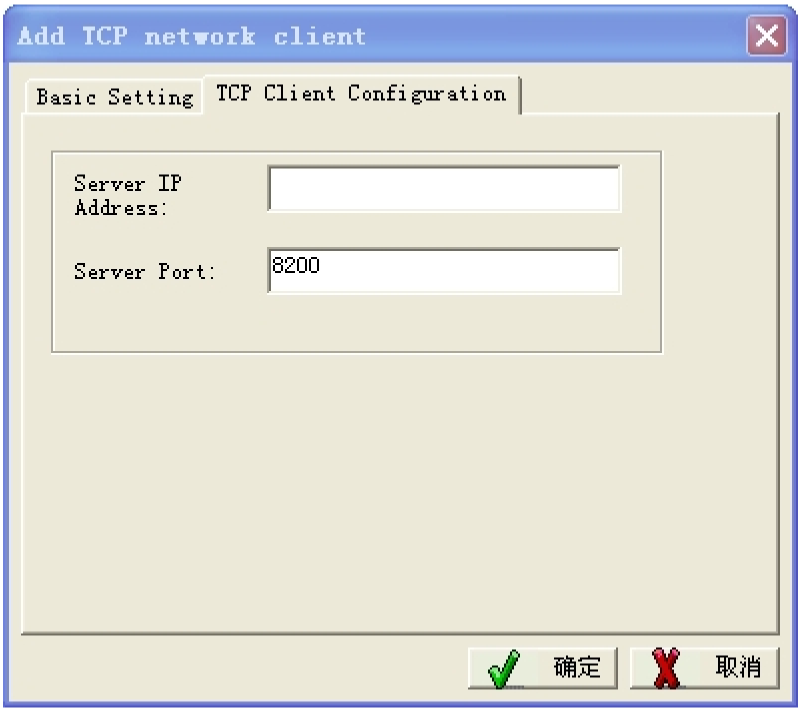
□ Disable Linkage

Figuur 11.8

3) Click on the **Basic Link Info** tab, and you can conFiguur the basic link information.

For details, see section 11.2.1.1 Configuring Basic Link Information.

Click on the **TCP Network Client Configuration** tab, and you can conFiguur communication parameters for the TCP Client, as shown in Figuur 11.9.



Server IP Address:

Server Port: 8200

Figuur 11.9

The parameters in Figuur 11.9 are described as follows:

- **Server IP Adres**: defines the IP adres of the TCP server to which the HMI is to be connected.

- **Server Port Number**: specifies the port number of the TCP server to which the HMI is to be connected.

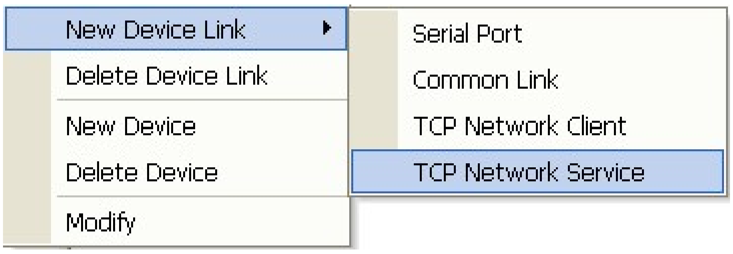
11.2.1.5 Adding a TCP Network Server

The TCP Network Server refers to the link which uses the TCP/IP network for communication and the HMI as the TCP server.

To add a TCP Network Server, do as follows:

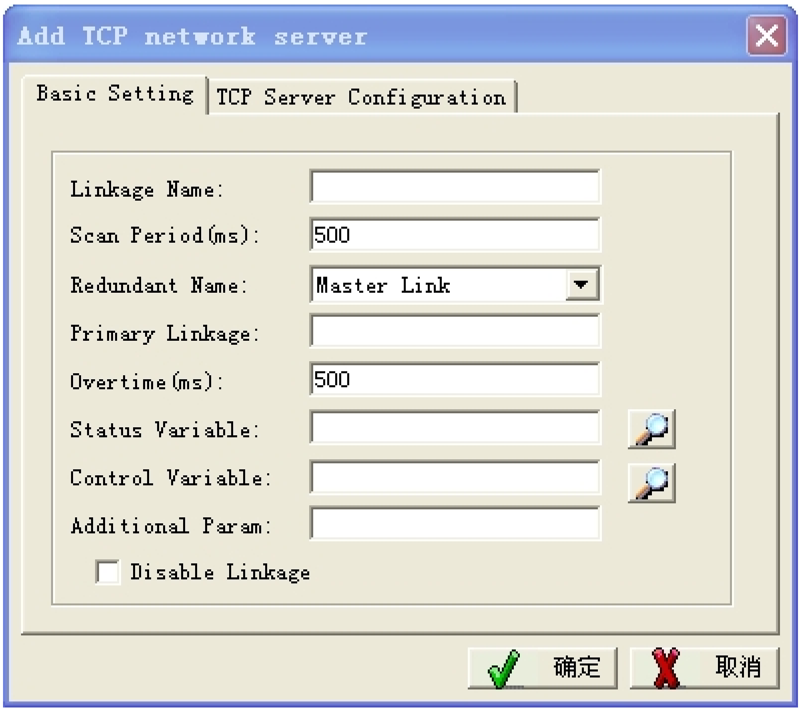
1) Select **Device Configuration** on the left side of the **Project Manager** window and right-click on it.

   You will see a right-click menu as shown in Figuur 11.10.



Figuur 11.10

2) Select **TCP Network Server**, and you will see a dialog box as shown in Figuur 11.11.



Linkage Name:

Scan Period(ms): 500

Redundant Name: Master Link

Primary Linkage:

Overtime(ms): 500

Status Variable:

Control Variable:

Additional Param:

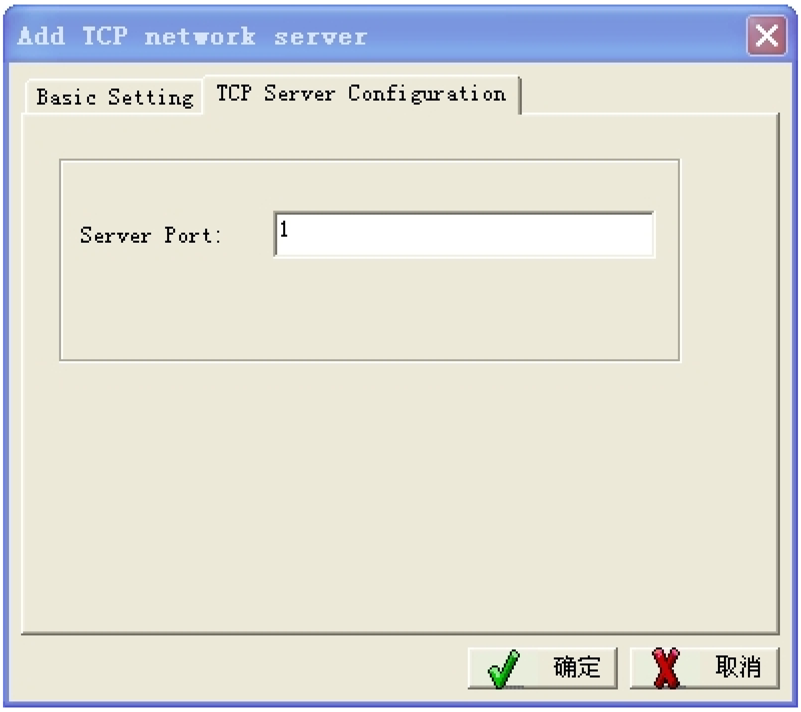
□ Disable Linkage

Figuur 11.11

3) Click on the **Basic Link Info** tab, and you can conFiguur the basic link information.

For details, see section 11.2.1.1 Configuring Basic Link Information.

Click on the **TCP Network Client Configuration** tab, and you can conFiguur communication parameters for the TCP client, as shown in Figuur 11.12.



Server Port: 1

Figuur 11.12

The parameter in Figuur 11.12 is described as follows:

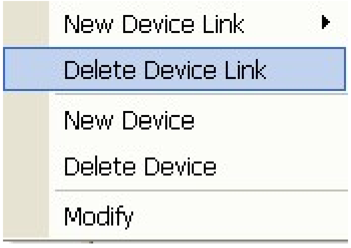
- **Server Port Number**: specifies the port number to be intercepted when the HMI runs as the TCP server.

11.2.1.6 Deleting a Communication Link

To delete an added communication link, do as follows:

1) Select the communication link you want to delete from the left side of the **Project Manager** window and right-click on it.

   You will see a right-click menu as shown in Figuur 11.13.



Figuur 11.13

2) Select **Delete a Communication Link**.

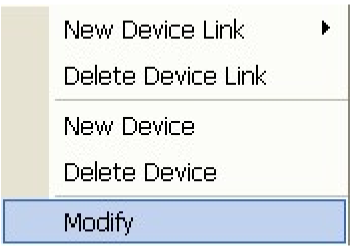
   The selected link will be deleted.

11.2.1.7 Modifying the Link Configuration

To modify an added communication link, do as follows:

1) Select the communication link you want to modify from the left side of the **Project Manager** window and right-click on it.

   You will see a right-click menu as shown in Figuur 11.14.



Figuur 11.14

2) Select **Modify**.

   You can modify the link according to your needs.

11.2.2 Adding Devices

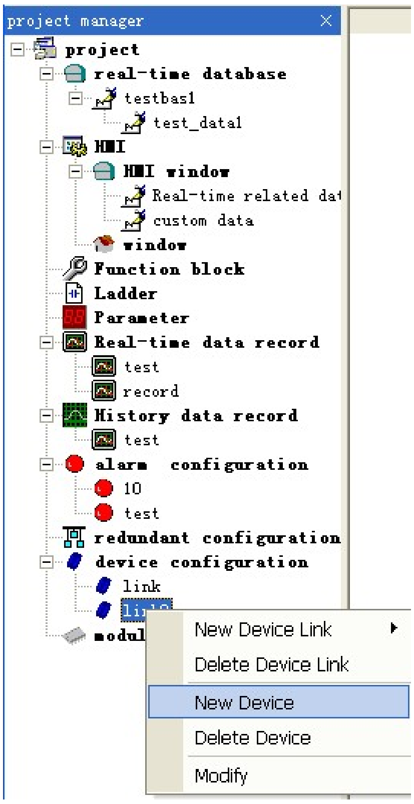
After you define a communication link, you can conFiguur devices for the link. The devices which can be conFiguurd for a link depends on the type of the link. For example, if the communication link is a Common Link, only Virtual IO Devices can be conFiguurd; if the communication link is a TCP Network Client, only three devices can be conFiguurd, which are EASY HMI, Modbus TCP Primary Device, and Siemens S7 300 Series PLC (using the Hilscher Netlink-MPI adaptor).

For details about each device, please see section 11.3 List of Devices. The following part will focus on how to add a device.

11.2.2.1 Adding a Device

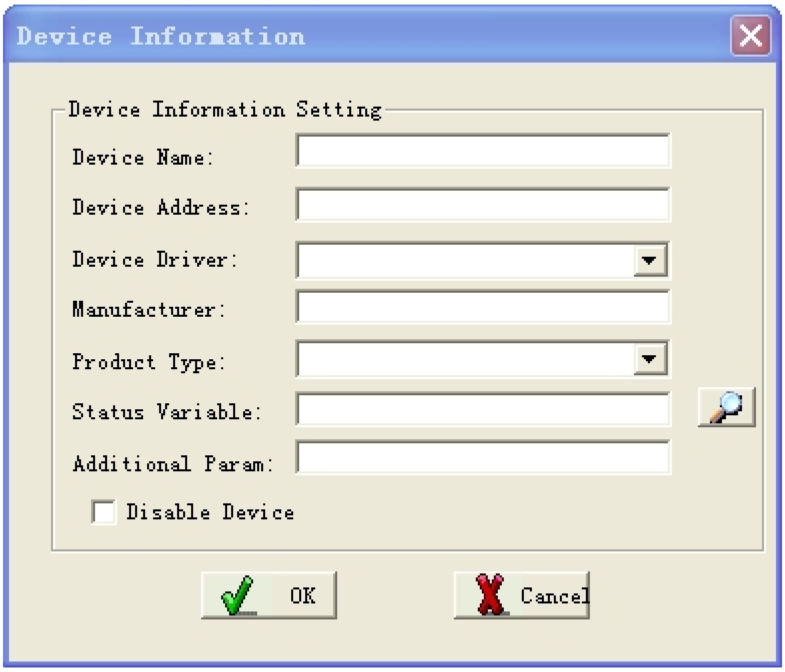
To add a device, do as follows:

1) Select a link for which you want to add a device on the left side of the **Project Manager** window and right-click on it, as shown in Figuur 11.15.



Figuur 11.15

2) Select **Add a Device**, and you will see a dialog box as shown in Figuur 11.16.



Figuur 11.16

The parameters in Figuur 11.16 are described as follows:

- **Device Name**: specifies the name of the device to be added with which the HMI is to communcate.
- **Device Adres**: differentiates various devices on the link. The definition of the

device adres varies according to the protocol used by the device.

- **Device Driver Name**: specifies the name of the driver for the device to be added. The devices on the link use different protocols for communication. EASY provides device drivers for all of the devices based on the different protocols used.
- **Manufacturer**: specifies the name of the device manufacturer. It is allowed to leave it blank.
- **Product Model**: specifies the product model of the device. Considering that even devices of the same manufacturer might use different communication protocols, it is required to specify the product model in addition to the name of the device driver. For example, the modules of the ICP DAS I-7000 series are of different models, such as 4050 or 4117.
- **Status Variable**: must be an int variable. The value of this variable can be one of the following based on the current communication state of the link:
  - -1 -- Device initialization failed
  - 0 -- Device initialization successful

If no status variable is defined, it means that you do not need to get the communication status of the device.

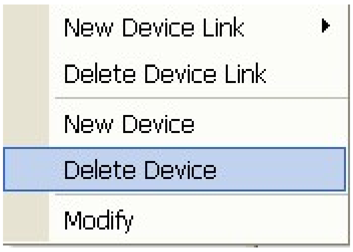
- **Additional Parameter**: defines the additional parameter for the device to be added.
- **Disable the device**: When you check this option, the device will be disabled, which means the system will not load or commnicate with the device.

11.2.2.2 Deleting a Device

To delete an added device, do as follows:

1) Select the device to be deleted on the left side of the **Project Manager** window and right-click on it.

   You will see a right-click menu as shown in Figuur 11.17.



Figuur 11.17

2) Select **Delete a Device**.

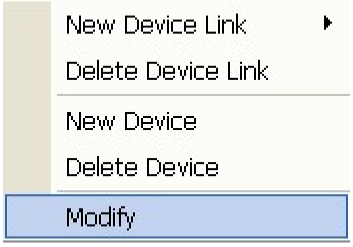
   The selected device will be deleted.

11.2.2.3 Modifying a Device

To modify an added device, do as follows:

1) Select the device to be modified on the left side of the **Project Manager** window and right-click on it.

You will see a right-click menu as shown in Figuur 11.18.



Figuur 11.18

2) Select **Modify**.

You can modify the device according to your needs.

11.2.3 Adding Data

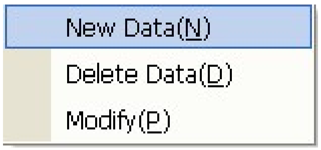
After adding a device, you can add device data for this device. The device data which can be added varies depending on the device itself. For example, for Siemens S7 200 series PLC, only the device data of the **I**, Q, M, and VW types can be added; while for the Delta PLC, only the device data of the X, Y, M, and S types can be added.

11.2.3.1 Adding Data

To add a device data, do as follows:

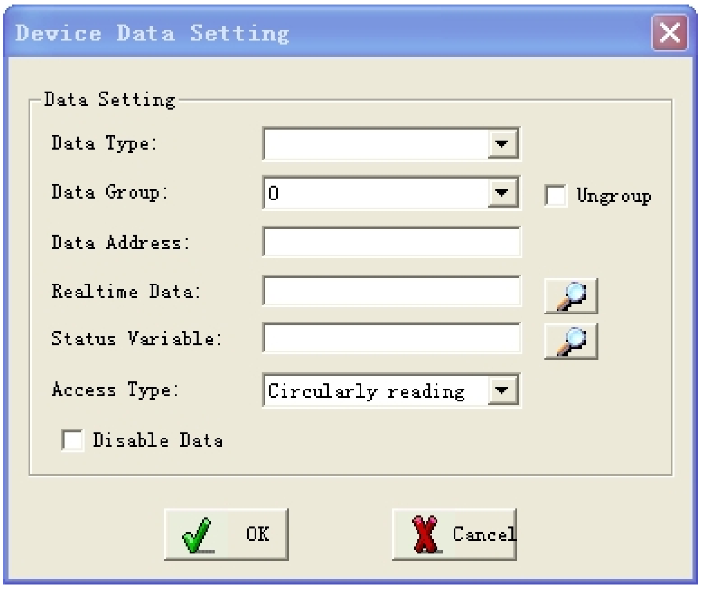
1) Select the device for which you want to add data on the left side of the **Project Manager** window and right-click on it.

You will see a right-click menu as shown in Figuur 11.19.



Figuur 11.19

2) Select **Add Data**, and you will see a dialog box as shown in Figuur 11.20.



Device Data Setting

Data Setting

Data Type:

Data Group: 0 □ Ungroup

Data Address:

Realtime Data:

Status Variable:

Access Type: Circularly reading

□ Disable Data

Figuur 11.20

The parameters in Figuur 11.20 are described as follows:

- **Data Type**: defines the type of the device data to be added. The types of device data which can be added for a device varies depending on the device itself.
- **Data Group**: The concept of data group is adopted by EASY to enhance data transfer efficiency. For the data which can be accessed in the same mode and are of the same data type and data group, EASY allows packing them into one data package during the communication with the device, instead of having one data in one package. However, while grouping data, it is recommended to avoid packing two data which are far apart from each other into the same data group. For example, to access two data, VW0 and VW1000, from Simense S7-200 series PLC. If you pack these two data into the same data group, the system will need to read all the data from VW0 to VW1000 first and then picks out the two target data VW0 and VW1000. However, accessing such huge volume of data at one time might not be allowed by the protocol involved (eah protocol has its own maximum transfer volume), and thus results in communication failure. At present, the system supports 8 data groups in total.
- The **Do Not Group** option: When you check this option, the data will not be grouped, which means the system will access this data in a separate message. As stated above, you can group data so that the system can access multiple data at one time, which helps reduce message transfer and thus enhance communication efficiency. From this sense, it is recommended to group the data as much as possible. However, if you group a data which is too discrete, the returned communication message for the scattered data might be too long, which might cause communication failure. In this case, it is not wise to group the data; it will be preferable if the system accesses it separately.
- **Data Adres**: defines the device adres for differentiating variaous devices on the link. The definition of the device adres varies depending on the protocol used by the device.
- **Real-Time Data Name**: specifies the real-time data in the real-time database

which is associated to the device data, which achieves data exchange between the device and the real-time database.

- **Status Variable Name**: must be an int variable. The value of this variable can be one of the following based on the current communication state of the data:
  - 0 -- Normal
  - -1 -- System error
  - -2 -- Port error
  - -3 -- Communication timeout
  - -4 -- Message error
  - Other -- Other error values defined in the protocol

If no status variable is defined, it means that you do not need to get the communication status of the data.

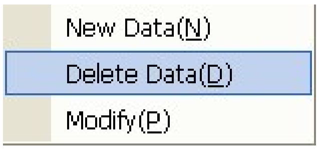
- **Access Mode**: defines how to access the data. At present, the system supports the following access modes:
  - Repeat Read – Repeately reading the value of the device data and then assigning the value to the real-time database data to which the device data is associated.
  - Repeat Write – Repeatedly write the value of the real-time database data to the associated device data
  - Repeat Read and Single Write – Repeatedly reading the value of the device data and then assigning the value to the real-time database data to which the device data is associated, while writing the value to the associated device data only after the value of the real-time database data changes.
  - Single Write – Writing the value to the associated device data only after the value of the real-time database data changes.
- **Disable the data**: When you check this option, the data will be disabled, which means the system will not load or access the data.

11.2.3.2 Deleting Data

To delete an added data, do as follows:

1) Select the device for which you want to delete a data on the left side of the **Project Manager** window, and select the data to be deleted in the list pane on the right side.

2) Right-click on the data, and you will see a right-click menu as shown in Figuur 11.21.



Figuur 11.21

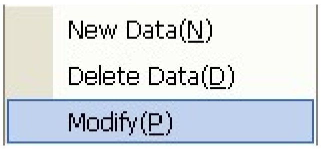
3) Select **Delete Data**.

The selected data will be deleted.

11.2.3.3 Modifying Data

To modify an added data, do as follows:

1) Select the device for which you want to modify a data on the left side of the **Project Manager** window, and select the data to be modified in the list pane on the right side.
2) Right-click on the data, and you will see a right-click menu as shown in Figuur 11.22.



Figuur 11.22

3) Select **Modify**.

You can modify the data according to your needs.

11.3 Device List

This section lists all the devices supported by EASY at present.

These devices can be categorized as follows based on the types of the communication links:

- Serial Ports
- Common Links
- TCP Network Clients
- TCP Network Servers

11.3.1 Serial Ports

11.3.1.1 Modbus RTU Primary Devices

The HMI works as the Modubs primary device and uses the Modbus RTU protocol for communication.

At present, the system supports the following types of data:

Data	Type	Description

Data	Type	Description
DI	bit	Input node. Read-only. In the Modubs protocol, corresponding to: Function number: 02 (Read the discrete input)
DO	Bit	Output node. Read and write. In the Modubs protocol, corresponding to: Function number: 01 (Read the loop) Function number: 15 (Write multiple loops)
AI	Short, ushort	Input register. Read-only. In the Modubs protocol, corresponding to: Function number: 04 (Read the input register)
AO	Short, ushort	Holding register. Read and write. In the Modubs protocol, corresponding to: Function number: 03 (Read the holding register) Function number: 16 (Write multiple registers)
AI_BIT	Bit	To obtain a certain bit of the AI data. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to obtain the third bit of the input register with the adres as 10 (Note: The bit number starts from 0 – the lowest bit), then you will need to select the data type **AI_BIT** and set the data adres as **10.3** while adding the device data.
AO_BIT	Bit	To read and write a certain bit of the AO data. Communiation Mode: Same as that for the AO data, corresponding to Function Nmber 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the third bit of the holding register with the adres as 10 (Note: The bit number starts from 0 – the lowest bit), then you will need to select the data type **AO_BIT** and set the data adres as **10.3** while adding the device data.
AI_LONG 1	long,ulong	To resolute two AI data with continuous adreses as an int value. Resolution Mode: The AI data with the lower adres is resoluted as the **higer 16 bits** of the int value, while the AI data with the higer adres as the **lower 16 bits**. For example, the two AI data, AI10=0x12 and AI11=0x34, will be resoluted as the int value 0x1234. Communiation Mode: Same as that for the AI data, corresponding to Function

Data	Type	Description
		Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the int value saved for the two data , AI10 and AI11, then you will need to select the data type **AI_LONG1** and set the data adres as **10** while adding the device data.
AI_LONG2	long,ulong	To resolute two AI data with continuous adreses as an int value. Resolution Mode: The AI data with the lower adres is resoluted as the **lower 16 bits** of the int value, while the AI data with the higer adres as the **higher 16 bits**. For example, the two AI data, AI10=0x12 and AI11=0x34, will be resoluted as the int value 0x3412. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the int value saved for the two data , AI10 and AI11, then you will need to select the data type **AI_LONG2** and set the data adres as **10** while adding the device data.
AO_LONG1	long,ulong	To resolute two AO data with continuous adreses as an int value. Resolution Mode: The AO data with the lower adres is resoluted as the **higer 16 bits** of the int value, while the AO data with the higer adres as the **lower 16 bits**. For example, the two AO data, AO10=0x12 and AO11=0x34, will be resoluted as the int value 0x1234. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the int value saved for the two data , AO10 and AO11, then you will need to select the data type **AO_LONG1** and set the data adres as **10** while adding the device data.
AO_LONG2	long,ulong	To resolute two AO data with continuous adreses as an int value. Resolution Mode: The AO data with the lower adres is resoluted as the **lower 16 bits** of the int value, while the AO data with the higer adres as the **higer 16 bits**. For example, the two AO data, AO10=0x12 and AO11=0x34, will be resoluted as the int value 0x3412. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the int value saved for the two data , AO10 and AO11, then you will need to select the data type **AO_LONG2** and set the data adres as **10** while adding the device data.

Data	Type	Description
AI_FLOAT1	float	To resolute two AI data with continuous adreses as a float value. Resolution Mode: The AI data with the lower adres is resoluted as the **higer 16 bits** of the float value, while the AI data with the higer adres as the **lower 16 bits**. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the float value saved for the two data, AI10 and AI11, then you will need to select the data type **AI_FLOAT1** and set the data adres as **10** while adding the device data.
AI_FLOAT2	float	To resolute two AI data with continuous adreses as a float value. Resolution Mode: The AI data with the lower adres is resoluted as the **lower 16 bits** of the float value, while the AI data with the higer adres as the **higer 16 bits**. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the float value saved for the two data, AI10 and AI11, then you will need to select the data type **AI_FLOAT2** and set the data adres as **10** while adding the device data.
AO_FLOAT1	float	To resolute two AO data with continuous adreses as a float value. Resolution Mode: The AO data with the lower adres is resoluted as the **higer 16 bits** of the float value, while the AO data with the higer adres as the **lower 16 bits**. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the float value saved for the two data, AO10 and AO11, then you will need to select the data type **AO_FLOAT1** and set the data adres as **10** while adding the device data.
AO_FLOAT2	float	To resolute two AO data with continuous adreses as a float value. Resolution Mode: The AO data with the lower adres is resoluted as the **lower 16 bits** of the float value, while the AO data with the higer adres as the **higer 16 bits**. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the float value saved for the two data, AO10 and AO11, then you will need to select the data type **AO_FLOAT2** and set the data adres as **10** while adding the device data.

11.3.1.2 Modbus RTU Slave Devices

The HMI works as the Modubs slave device and uses the Modbus RTU protocol for communication.

At present, the system supports the following types of data:

Data	Type	Description
DI	bit	Input node. Read-only. In the Modubs protocol, corresponding to: Function number: 02 (Read the discrete input)
DO	bit	Output node. Read and write. In the Modubs protocol, corresponding to: Function number: 01 (Read the loop) Function number: 5 (Write a single loop) Function number: 15 (Write multiple loops)
AI	short, ushort	Input register. Read-only. In the Modubs protocol, corresponding to: Function number: 04 (Read the input register)
AO	short, ushort	Holding register. Read and write. In the Modubs protocol, corresponding to: Function number: 03 (Read the holding register) Function number: 6 (Write a single register) Function number: 16 (Write multiple registers)

Note:

When working as the Modbus slave device, the HMI receives requests from the primary station, carries out operations accordingly, and then responds to the primary station. The operation mode of the Modubs slave device can be described as “Passively Triggered”. Therefore, the parameters **Data Group** and **Access Mode**, and the option **Disable the data** for the data configuration are meaningless, and thus are ignored by the system.

11.3.1.3 Modbus ASCII Primary Devices

The HMI works as the Modubs primary device and uses the Modbus ASCII protocol for communication.

At present, the system supports the following types of data:

Data	Type	Description

Data	Type	Description
DI	bit	Input node. Read-only. In the Modubs protocol, corresponding to: Function number: 02 (Read the discrete input)
DO	bit	Output node. Read and write. In the Modubs protocol, corresponding to: Function number: 01 (Read the loop) Function number: 15 (Write multiple loops)
AI	Short, ushort	Input register. Read-only. In the Modubs protocol, corresponding to: Function number: 04 (Read the input register)
AO	Short, ushort	Holding register. Read and write. In the Modubs protocol, corresponding to: Function number: 03 (Read the holding register) Function number: 16 (Write multiple registers)
AI_BIT	bit	To obtain a certain bit of the AI data. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to obtain the third bit of the input register with the adres as 10 (Note: The bit number starts from 0 – the lowest bit), then you will need to select the data type **AI_BIT** and set the data adres as **10.3** while adding the device data.
AO_BIT	bit	To obtain a certain bit of the AO data. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to obtain the third bit of the holding register with the adres as 10 (Note: The bit number starts from 0 – the lowest bit), then you will need to select the data type **AO_BIT** and set the data adres as **10.3** while adding the device data.
AI_LONG1	long,ulong	To resolute two AI data with continuous adreses as an int value. Resolution Mode: The AI data with the lower adres is resoluted as the **higer 16 bits** of the int value, while the AI data with the higer adres as the **lower 16 bits**. For example, the two AI data, AI10=0x12 and AI11=0x34, will be resoluted as the int value 0x1234. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol.

Data	Type	Description
		Configuration Mode: If you want to read the int value saved for the two data , AI10 and AI11, then you will need to select the data type **AI_LONG1** and set the data adres as **10** while adding the device data.
AI_LONG2	Long,ulong	To resolute two AI data with continuous adreses as an int value. Resolution Mode: The AI data with the lower adres is resoluted as the **lower 16 bits** of the int value, while the AI data with the higer adres as the **higher 16 bits**. For example, the two AI data, AI10=0x12 and AI11=0x34, will be resoluted as the int value 0x3412. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the int value saved for the two data , AI10 and AI11, then you will need to select the data type **AI_LONG2** and set the data adres as **10** while adding the device data.
AO_LONG1	Long,ulong	To resolute two AO data with continuous adreses as an int value. Resolution Mode: The AO data with the lower adres is resoluted as the **higer 16 bits** of the int value, while the AO data with the higer adres as the **lower 16 bits**. For example, the two AO data, AO10=0x12 and AO11=0x34, will be resoluted as the int value 0x1234. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the int value saved for the two data , AO10 and AO11, then you will need to select the data type **AO_LONG1** and set the data adres as **10** while adding the device data.
AO_LONG2	long,ulong	To resolute two AO data with continuous adreses as an int value. Resolution Mode: The AO data with the lower adres is resoluted as the **lower 16 bits** of the int value, while the AO data with the higer adres as the **higer 16 bits**. For example, the two AO data, AO10=0x12 and AO11=0x34, will be resoluted as the int value 0x3412. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the int value saved for the two data , AO10 and AO11, then you will need to select the data type **AO_LONG2** and set the data adres as **10** while adding the device data.
AI_FLOAT1	float	To resolute two AI data with continuous adreses as a float value.

Data	Type	Description
		Resolution Mode: The AI data with the lower adres is resoluted as the **higer 16 bits** of the float value, while the AI data with the higer adres as the **lower 16 bits**. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the float value saved for the two data, AI10 and AI11, then you will need to select the data type **AI_FLOAT1** and set the data adres as **10** while adding the device data.
AI_FLOAT2	float	To resolute two AI data with continuous adreses as a float value. Resolution Mode: The AI data with the lower adres is resoluted as the **lower 16 bits** of the float value, while the AI data with the higer adres as the **higer 16 bits**. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the float value saved for the two data, AI10 and AI11, then you will need to select the data type **AI_FLOAT2** and set the data adres as **10** while adding the device data.
AO_FLOAT1	float	To resolute two AO data with continuous adreses as a float value. Resolution Mode: The AO data with the lower adres is resoluted as the **higer 16 bits** of the float value, while the AO data with the higer adres as the **lower 16 bits**. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the float value saved for the two data, AO10 and AO11, then you will need to select the data type **AO_FLOAT1** and set the data adres as **10** while adding the device data.
AO_FLOAT2	float	To resolute two AO data with continuous adreses as a float value. Resolution Mode: The AO data with the lower adres is resoluted as the **lower 16 bits** of the float value, while the AO data with the higer adres as the **higer 16 bits**. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the float value saved for the two data, AO10 and AO11, then you will need to select the data type **AO_FLOAT2** and set the data adres as **10** while adding the device data.

11.3.1.4 Mitsubishi FX2N Series PLC

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS485
Data Bit	7
Stop Bit	1
Baud Rate	9600
Parity Check	Even
Device Adres	0

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres Format	Description
X	bit	OOO	External input node
Y	bit	OOO	External output node
M	bit	DDD	Internal auxiliary node
S	bit	DDD	Special auxiliary node
T	bit	DDD	Timer node
C	bit	DDD	Couter node
TV	short, ushort	DDD	Timer register
CV	short, ushort	DDD	Counter register
D	short, ushort	DDD	Data register
D_ARRAY	short, ushort data array	DDD~DDD	Data array stored in the data register

Note: D stands for the decimal system, and O stands for the octal system.

11.3.1.5 Mitsubishi Q02H PLC

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS232
Data Bit	8
Stop Bit	1
Baud Rate	115200
Parity Check	Odd

Parameter	Default Value
Device Adres	0

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres Format	Description
X	bit	HHH	External input node
Y	bit	HHH	External output node
M	bit	DDDD	Internal auxiliary node
L	bit	DDDD	Auxiliary node
F	bit	DDDD	Alarm node
V	bit	DDDD	Edge-triggered node
B	bit	HHH	Link register node
TC	bit	DDD	Timer loop
SS	bit	DDD	Holding timer node
SC	bit	DDD	Holding timer loop
CS	bit	DDD	Counter node
CC	bit	DDD	Counter loop
SB	bit	HHH	Special connection register node
S	bit	DDDD	Stepping register
DX	bit	HHH	Direct input node
DY	bit	HHH	Direct output node
TS	bit	DDD	Timer node
W	short, ushort	HHH	Connection register
TN	short, ushort	DDD	Current timer value
SN	short, ushort	DDD	Current value of the holding register
CN	short, ushort	DDD	Current counter value
R	short, ushort	DDDD	File register
SW	short, ushort	HHH	Special connection register
Z	short, ushort	D	Index register
ZR	short, ushort	HHHH	File register
D	short, ushort	DDDD	Data register

Note: D stands for the decimal system, and H stands for the hex system with the range as 0-F.

11.3.1.6 ICP DAS I-7000 Series (ADVANTECH ADAM4000/4100 Series) IO Modules

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres Format	Description
DI	Bit	Decimal	DI input, for DI or mixed modules
DO	Bit		DO output, for DO or mixed modules
AI	Float		AI input, for AI modules and only can be read as project quantities
AO	Float		AO output, for AO modules and only can be written in as project quantities
COUNTER	Long		Counter, for the modules with the counter function and for reading the counter value
CLEARCOUNTER	Bit		For the modules with the counter function. If you set the data value as 1, the counter value will be reset back to 0.

Notes:

1) For the IO modules of this series, you need to specify the specific product model considering the slight difference in the communication protocols used. Regarding the model numbers containing D, for example 7045D, you can put the model number as 7045.
2) The IO modules of this series support two types of parity checks: **checksum enable** and **checksum disable**. While adding a device, if you do not specify the additional parameter for the device, then **checksum enable** is used; to activate **checksum disable**, you will need to enter **checksum=0** in the **Additional Parameter** text box.

11.3.1.7 Siemens S7-200 Series PLC

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS485
Data Bit	8
Stop Bit	1
Baud Rate	9600
Parity Check	Even

Parameter	Default Value
Device Adres	2

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
I	Bit	DDD.O	Digital input
Q	Bit	DDD.O	Digital Output
M	Bit	DDD.O	Mark bit of the internal register
VW	short,ushort	DDDD	Variable of the internal register (word)
AIW	short,ushort	DDDD	Analog input
AQW	short,ushort	DDDD	Analog output
VB	char,uchar	DDDD	Variable of the internal register (byte)
VD	long, ulong	DDDD	Variable of the internal register (double-byte)

Note: D stands for the decimal system, and O stands for the octal system.

Notes:

Siemens 200 series PLC supports two communication modes, **single item** and **multi items**. While adding a device, if you do not specify the additional parameter or the specified additional parameter is **multiitem=0**, then **single item** is used; to activate **multi items**, you will need to enter **multiitem=1** in the **Additional Parameter** text box.

The two communication modes, **single item** and **multi items** are described in details as below:

- **single item**: allows accessing only one type of data in one communication message. For example, to access the four data I0.0, I1.0, Q0.0, and Q1.0 in the **single item** mode, you will need to use two communication messages (one for accessing I0.0 and I1.0, and the other for accessing Q0.0 and Q1.0), because the I data and the Q data are of two different types.
- **multi items**: allows accessing various types of data in one communication message. To access the same four data mentioned above, the **multi items** mode allows accessing all of the four data at one time in one message, separating I0.0 and I1.0 as one item and Q0.0 and Q1.0 as another item.

From the example above, it is obvious that the **multi items** mode provides higher communication efficiency compared with the **single item** mode. However, accessing various types of data in one message might make the message too long to be allowed by the Siemens communication protocol (256 bytes allowed the most). If a message contains data more than 256 bytes, it will end up with communication failure.

In terms of EASY, the **multi items** mode will group the data accessed in the same mode ine one message, which will maximize the communication efficiency. However, it might cause the communication message to be longer than 256 bytes allowed by the protocol involved. Considering all of this, it can be concluded that the **multi items** mode is suitable for the situations with small volume of communication data only.

11.3.1.8 Siemens S7-300 Series PLC (Using the MPI Adaptor)

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS232
Data Bit	8
Stop Bit	1
Baud Rate	19200
Parity Check	Odd
Device Adres	2

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
I	bit	DDD.O	Digital input
Q	bit	DDD.O	Digital Output
M	bit	DDD.O	Mark bit of the internal register
PIB	char,uchar	DDDD	Analog input (byte)
PIW	short,ushort	DDDD	Analog input (word)
PID	long, ulong	DDDD	Analog input (double word)
PQB	char,uchar	DDDD	Analog output (byte)
PQW	short,ushort	DDDD	Analog output (word)
PQD	long, ulong	DDDD	Analog output (double word)
DBX	bit	DDDD.O	Data block (bit)
DBB	char,uchar	DDDD	Data block (byte)
DBW	short,ushort	DDDD	Data block (word)
DBD	long, ulong	DDDD	Data block (double word)

Note: D stands for the decimal system, and O stands for the octal system.

11.3.1.9 Omron PLC

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS232
Data Bit	7
Stop Bit	2
Baud Rate	9600
Parity Check	Even
Device Adres	0

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
IR	bit	DDDBB	I/O and internal relay
HR	bit	DDDBB	Holding relay
AR	bit	DDDBB	Auxiliary relay
LR	bit	DDDBB	Link relay
TC	short,ushort	DDD	Timer/Counter register
DM	short,ushort	DDDD	Data register

Note: D stands for the decimal system, and B stands for the bit coding with the range as 0-15.

11.3.1.10 LG Series PLC (Using the CNET Protocol)

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS232
Data Bit	8
Stop Bit	1
Baud Rate	19200
Parity Check	N/A
Device Adres	0

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
P	bit	DDDH	Input/Output relay
M	bit	DDDH	Internal auxiliary relay
L	bit	DDDH	Link relay
K	bit	DDDH	Holding relay
C	bit	DDDH	Counter relay
T	bit	DDDH	Timer relay
F	bit	DDDH	Special relay
D	short,ushort	DDD	Data register
S	short,ushort	DDDD	Register
CV	short,ushort	DDDD	Current counter value
TV	short,ushort	DDDD	Current timer value

Note: D stands for the decimal system, and H stands for the hex system with the range as 0-F.

11.3.1.11 LG Series PLC (Using the LOAD Protocol)

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS232
Data Bit	8
Stop Bit	1
Baud Rate	38400
Parity Check	N/A
Device Adres	0

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
P	bit	DDDH	Input/Output relay
M	bit	DDDH	Internal auxiliary relay
L	bit	DDDH	Link relay
K	bit	DDDH	Holding relay
C	bit	DDDH	Counter relay
T	bit	DDDH	Timer relay
F	bit	DDDH	Special relay

Data	Type	Adres format	Description
D	short,ushort	DDD	Data register
S	short,ushort	DDDD	Register
CV	short,ushort	DDDD	Current counter value
TV	short,ushort	DDDD	Current timer value

Note: D stands for the decimal system, and H stands for the hex system with the range as 0-F.

11.3.1.12 Delta PLC

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS232
Data Bit	7
Stop Bit	1
Baud Rate	9600
Parity Check	Even
Device Adres	1

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
X	bit	OOO	Digital input relay
Y	bit	OOO	Digital output relay
M	bit	DDD	Internal auxiliary relay
S	bit	DDD	Sequential control relay
T	bit	DDD	Timer relay
C	bit	DDD	Counter relay
D	short,ushort	DDDD	Data register
TV	short,ushort	DDDD	Timer register
CV	short,ushort	DDDD	Counter register (word)
CV2	long, ulong	DDDD	Counter register (double word)

Note: D stands for the decimal system, and O stands for the octal system with the range as 0-7.

11.3.1.13 Panasonic NAIS FP Series PLC

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS232
Data Bit	8
Stop Bit	1
Baud Rate	9600
Parity Check	Odd
Device Adres	1

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
X	bit	DDDH	Digital input relay
Y	bit	DDDH	Digital output relay
R	bit	DDDH	Internal auxiliary relay
L	bit	DDDH	Link control relay
T	bit	DDD	Timer relay
C	bit	DDD	Counter relay
DT	short,ushort	DDD	Data register
SV	short,ushort	DDD	Preset timer/counter value register
EV	short,ushort	DDD	Actual timer/counter value register

Note: D stands for the decimal system, and H stands for the octal system with the range as 0-F.

11.3.1.14 Emerson Series PLC (Using the Modbus RTU protocol)

The default values of the related communication parameters are listed as follows:

Parameter	Default Value
Communication Port Type	RS485
Data Bit	8
Stop Bit	1
Baud Rate	19200
Parity Check	Even
Device Adres	1

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres format	Description
X	bit	OOO	Digital input relay
Y	bit	OOO	Digital output relay
M	bit	DDDD	Auxiliary relay
SM	bit	DDD	Special auxiliary relay
S	bit	DDD	Stepping register
T	bit	DDD	Timer loop
C	bit	DDD	Counter loop
D	short,ushort	DDDD	Data register
SD	short,ushort	DDD	Special data register
Z	short,ushort	DDD	Index register
TV	short,ushort	DDD	Timer
CV	short,ushort	DDD	Counter
D_D	long,ulong	DDDD	Data register (double word)
CV_D	long,ulong	DDD	Counter (double word)

Note: D stands for the decimal system, and H stands for the octal system with the range as 0-F.

11.3.2 Communication Links

11.3.2.1 Virtual IO Devices

EASY provides internal virtual IO devices. They can be accessed and operated in almost the same mode as the actual devices. The only difference is that reading and writing the virtual devices do not involve the accessing of the serial port or the TCP/IP network. Besides, you can use the virtual IO devices for tesing purpose.

At present, the system supports the following types of data, as listed in the table below:

Data Symbol	Data Type	Function
DI	bit	Read-only DI bits
DO	bit	Write-only DO bits
DIO	bit	Read and write digitals

Data Symbol	Data Type	Function
AI	short,ushort	Read-only AI values
AO	short,ushort	Write-only AO values
AIO	short,ushort	Read and write analogs
INC	bit,char,uchar,short,ushort,long,ulong	Auto-increment int variable; 1 added each time; read and write
INCF	float,double	Auto-increment float variable; 1 added each time; read and write
DEC	bit,char,uchar,short ushort,long,ulong	Auto-decrement int variable; 1 added each time; read and write
DECF	float,double	Auto-decrement float variable; 1 added each time; read and write
RAND	bit,char,uchar,short ushort,long,ulong	Random int variable, one random value generated each time
COMERR	bit,char,uchar,short ushort,long,ulong	To set the communication status variable, with the value range as below: • 1: Link timeout • 2: System error • 0: Normal

11.3.3 TCP Network Clients

11.3.3.1 EASY HMIs

TCP network clients are used for the mutual communication between EASY HMIs. You can add EASY HMI devices, and the HMIs can mutually access the real-time data from the real-time database of each other.

While adding EASY HMI devices, you must set the server port number to 8200.

11.3.3.2 Modbus TCP Primary Devices

The HMI works as the Modubs primary device and uses the Modbus TCP/IP protocol for communication.

According to the Modbus TCP/IP protocal, the default port number is 502.

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Function Description
DI	bit	Input node. Read-only. In the Modubs protocol, corresponding to: Function number: 02 (Read the discrete input)
DO	bit	Output node. Read and write. In the Modubs protocol, corresponding to: Function number: 01 (Read the loop) Function number: 15 (Write multiple loops)
AI	short, ushort	Input register. Read-only. In the Modubs protocol, corresponding to: Function number: 04 (Read the input register)
AO	short, ushort	Holding register. Read and write. In the Modubs protocol, corresponding to: Function number: 03 (Read the holding register) Function number: 16 (Write multiple registers)
AI_BIT	bit	To obtain a certain bit of the AI data. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to obtain the third bit of the input register with the adres as 10 (Note: The bit number starts from 0 – the lowest bit), then you will need to select the data type **AI_BIT** and set the data adres as **10.3** while adding the device data.
AO_BIT	bit	To read and write a certain bit of the AO data. Communiation Mode: Same as that for the AO data, corresponding to Function Nmber 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the third bit of the holding register with the adres as 10 (Note: The bit number starts from 0 – the lowest bit), then you will need to select the data type **AO_BIT** and set the data adres as **10.3** while adding the device data.
AI_LONG1	long,ulong	To resolute two AI data with continuous adreses as an int value. Resolution Mode: The AI data with the lower adres is resoluted as the **higer 16 bits** of the int value, while the AI data with the higer adres as the **lower 16 bits**. For example,

Data	Type	Function Description
		the two AI data, AI10=0x12 and AI11=0x34, will be resoluted as the int value 0x1234. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the int value saved for the two data , AI10 and AI11, then you will need to select the data type **AI_LONG1** and set the data adres as **10** while adding the device data.
AI_LONG2	long,ulong	To resolute two AI data with continuous adreses as an int value. Resolution Mode: The AI data with the lower adres is resoluted as the **lower 16 bits** of the int value, while the AI data with the higer adres as the **higher 16 bits**. For example, the two AI data, AI10=0x12 and AI11=0x34, will be resoluted as the int value 0x3412. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the int value saved for the two data , AI10 and AI11, then you will need to select the data type **AI_LONG2** and set the data adres as **10** while adding the device data.
AO_LONG1	long,ulong	To resolute two AO data with continuous adreses as an int value. Resolution Mode: The AO data with the lower adres is resoluted as the **higer 16 bits** of the int value, while the AO data with the higer adres as the **lower 16 bits**. For example, the two AO data, AO10=0x12 and AO11=0x34, will be resoluted as the int value 0x1234. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the int value saved for the two data , AO10 and AO11, then you will need to select the data type **AO_LONG1** and set the data adres as **10** while adding the device data.
AO_LONG2	long,ulong	To resolute two AO data with continuous adreses as an int value. Resolution Mode: The AO data with the lower adres is resoluted as the **lower 16 bits** of the int value, while the AO data with the higer adres as the **higer 16 bits**. For example, the two AO data, AO10=0x12 and AO11=0x34, will be resoluted as the int value 0x3412. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the int value saved for the two data , AO10 and AO11, then you will need to select the data type **AO_LONG2** and set the

Data	Type	Function Description
		data adres as **10** while adding the device data.
AI_FLOAT1	float	To resolute two AI data with continuous adreses as a float value. Resolution Mode: The AI data with the lower adres is resoluted as the **higer 16 bits** of the float value, while the AI data with the higer adres as the **lower 16 bits**. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the float value saved for the two data, AI10 and AI11, then you will need to select the data type **AI_FLOAT1** and set the data adres as **10** while adding the device data.
AI_FLOAT2	float	To resolute two AI data with continuous adreses as a float value. Resolution Mode: The AI data with the lower adres is resoluted as the **lower 16 bits** of the float value, while the AI data with the higer adres as the **higer 16 bits**. Communiation Mode: Same as that for the AI data, corresponding to Function Number 04 (Read the input register) in the Modubs protocol. Configuration Mode: If you want to read the float value saved for the two data, AI10 and AI11, then you will need to select the data type **AI_FLOAT2** and set the data adres as **10** while adding the device data.
AO_FLOAT1	float	To resolute two AO data with continuous adreses as a float value. Resolution Mode: The AO data with the lower adres is resoluted as the **higer 16 bits** of the float value, while the AO data with the higer adres as the **lower 16 bits**. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the float value saved for the two data, AO10 and AO11, then you will need to select the data type **AO_FLOAT1** and set the data adres as **10** while adding the device data.
AO_FLOAT2	float	To resolute two AO data with continuous adreses as a float value. Resolution Mode: The AO data with the lower adres is resoluted as the **lower 16 bits** of the float value, while the AO data with the higer adres as the **higer 16 bits**. Communiation Mode: Same as that for the AO data, corresponding to Function Number 03 (Read the holding register) and Function Number 16 (Write multiple registers) in the Modubs protocol. Configuration Mode: If you want to read and write the float value saved for the two data, AO10 and AO11, then you will need to select the data type **AO_FLOAT2** and set the data adres as **10** while adding the device data.

11.3.3.3 Siemens S7 300 Series PLC (Using the Hilscher Netlink - MPI Adaptor)

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Adres Format	Description
I	bit	DDD.O	Digital input
Q	bit	DDD.O	Digital output
M	bit	DDD.O	Mark bit of the internal register
IB	char,uchar	DDDD	Digital input (byte)
QB	char,uchar	DDDD	Digital output (byte)
MB	char,uchar	DDDD	Mark bit of the internal register (byte)
C	short,ushort	DDDD	Counter register
T	short,ushort	DDDD	Timer register
DBX	bit	DDDD.O	Data block (bit)
DBB	char,uchar	DDDD	Data block (byte)
DBW	short,ushort	DDDD	Data block (word)
DBD	long, ulong,float	DDDD	Data block (double word)

Note: D stands for the decimal system, and O stands for the octal system.

11.3.4 TCP Network Servers

11.3.4.1 Modbus TCP Slave Devices

The HMI works as the Modubs slave device and uses the Modbus TCP/IP protocol for communication. According to the Modbus TCP/IP protocol, the default port is 502.

At present, the system supports the following types of data, as listed in the table below:

Data	Type	Function Description
DI	bit	Input node. Read-only. In the Modubs protocol, corresponding to: Function number: 02 (Read the discrete input)
DO	bit	Output node. Read and write. In the Modubs protocol, corresponding to: Function number: 01 (Read the loop) Function number: 5 (Write a single loop) Function number: 15 (Write multiple loops)
AI	short, ushort	Input register. Read-only.

Data	Type	Function Description
		In the Modubs protocol, corresponding to: Function number: 04 (Read the input register)
AO	short, ushort	Holding register. Read and write. In the Modubs protocol, corresponding to: Function number: 03 (Read the holding register) Function number: 6 (Write a single register) Function number: 16 (Write multiple registers)

Note:

When working as the Modbus slave device, the HMI receives requests from the primary station, carries out operations accordingly, and then responds to the primary station. The operation mode of the Modubs slave device can be described as “Passively Triggered”. Therefore, the parameters **Data Group** and **Access Mode**, and the option **Disable the data** for the data configuration are meaningless, and thus are ignored by the system.

11.4 System Variables for Device Configuration

Database Name	Variable Name	Data Type	Default Value	Description
system	IoEnable	bit	1	The available values are described as follows: • 1: Device management function enabled • 0: Device management function disabled
hmi_system_set	link_timeout_wnd_on	bit	1	The available values are described as follows: • 1: The system automatically displays the communication timeout prompt window when communication timeout occurs. • 0: The system does not display the timeout prompt window when communiation timeout occurs.
	link_timeout_wnd_x	short	-1	Defines the value of the X-axis in the top left corner of the communication timeout prompt window.

	link_timeout_wnd_y	Short	-1	Defines the value of the Y-axis in the top left corner of the communication timeout prompt window.

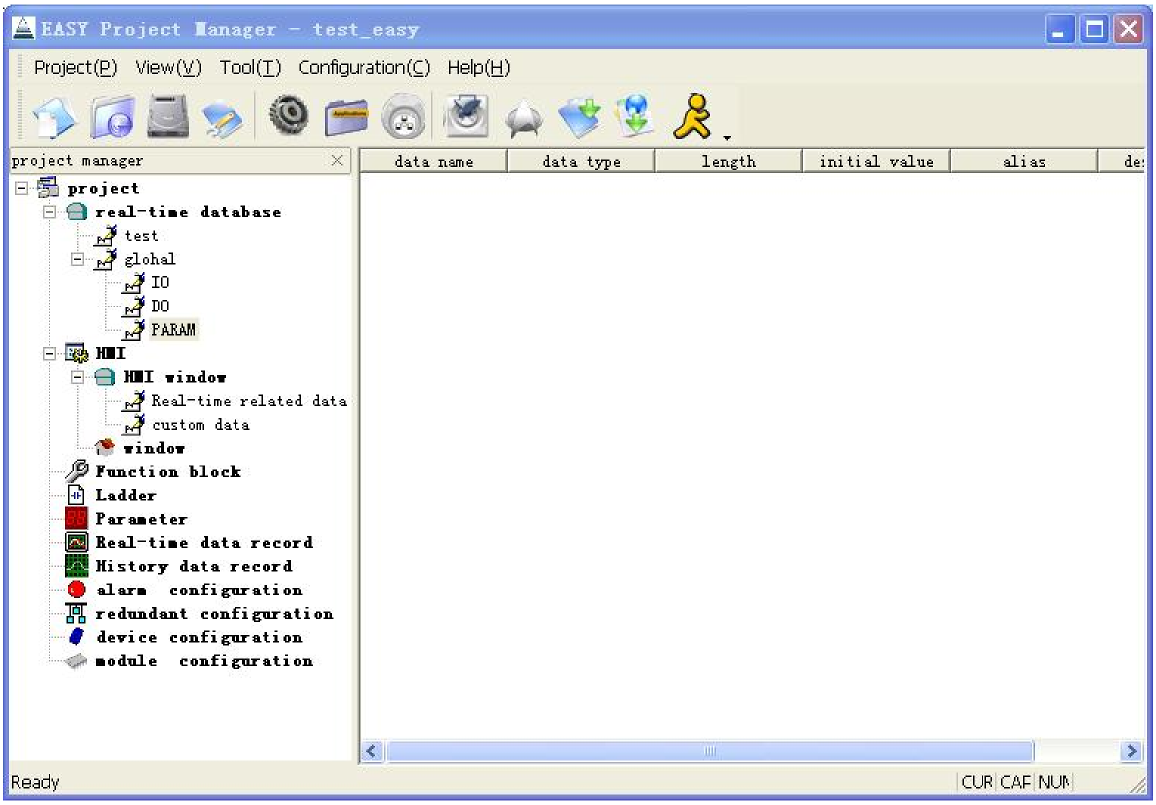
11.5 Example for Device Configuration

Take the Siemens S7-200 series PLC for example to explain how to conFiguur a device.

Suppose that the S7-200 PLC application involoves the following variables: I0.0, I0.1, I0.2, I0.3, Q0.0, Q0.1, Q0.2, Q0.3, M10.1, M10.2, M10.3, and M10.4, and that all these data requires data exchange with the HMI.

In this case, do the device configuration as follows:

1. Under the **Real-Time Database** node in the **Project Manager** window, create a database named **global**, and then create three data groups **IO**, **DO**, and **PARAM** under the **global** node, as shown in Figuur 11.23.



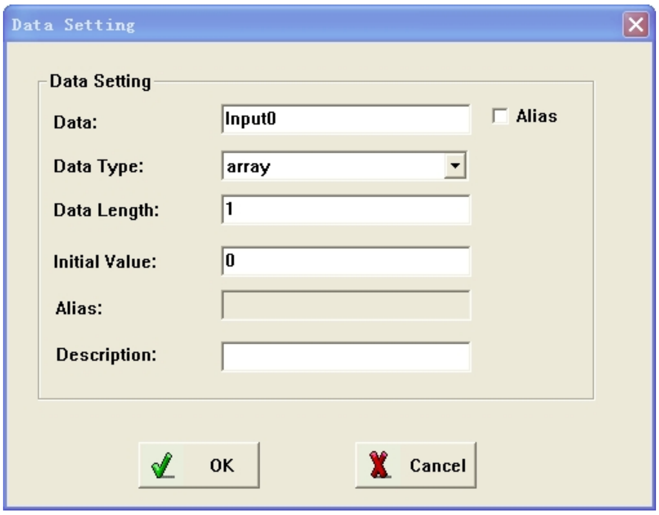
Figuur 11.23

Under the three added data groups, add real-time database data, as described below:

5) For the **IO** data group, add the data **Input0**, **Input1**, **Input2**, and **Input3**.

Take **Input0** for example. Do the data configuration as shown in Figuur 11.24 and

Figuur 11.25.


Data Setting

Data Setting

Data: Input0 □ Alias

Data Type: array

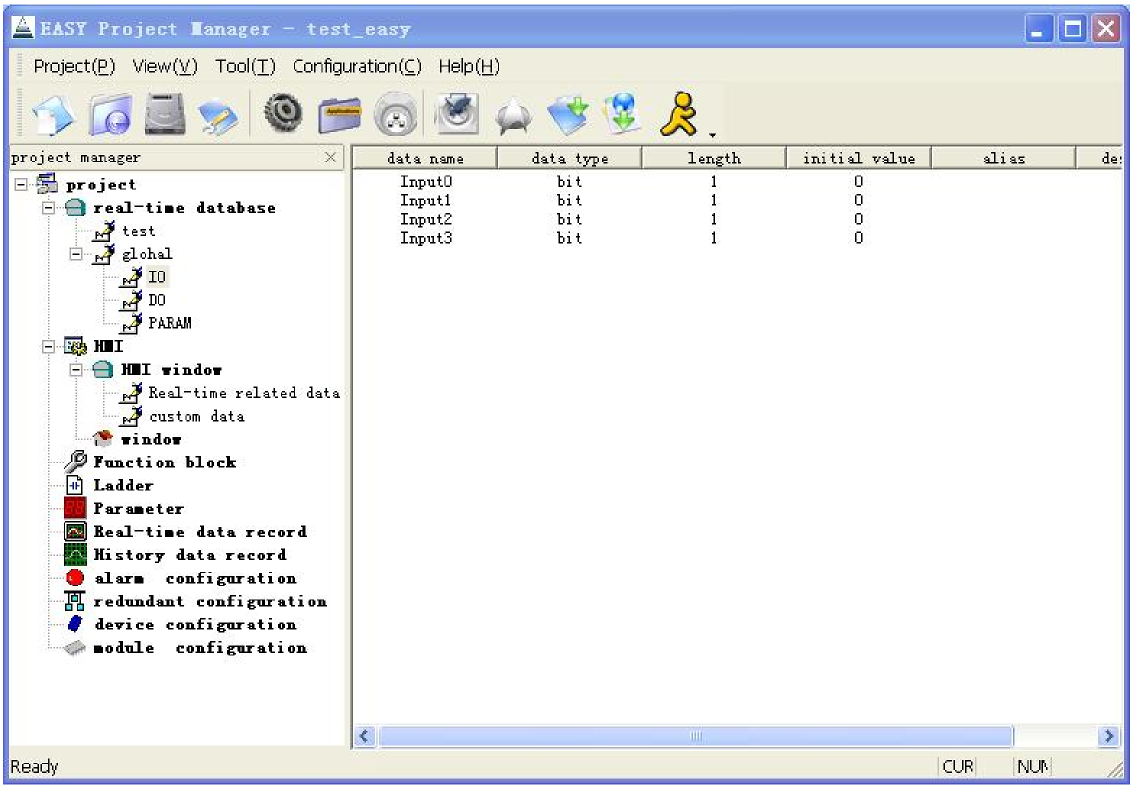
Data Length: 1

Initial Value: 0

Alias:

Description:

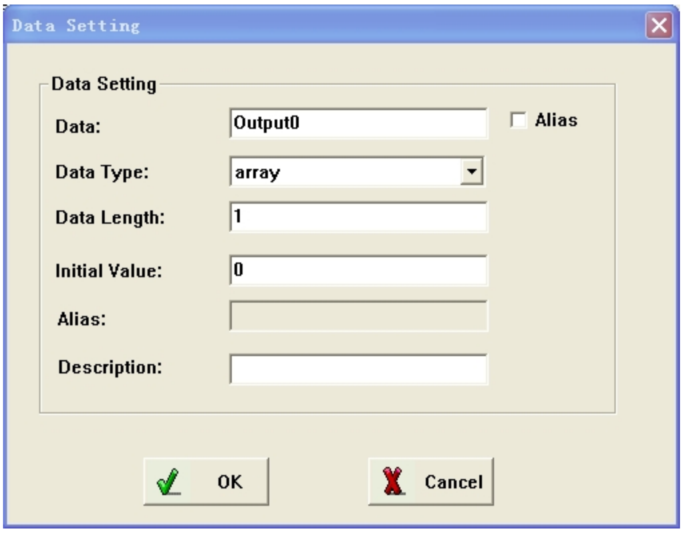
Figuur 11.24


EASY Project Manager - test_easy

data name	data type	length	initial value	alias	de:
Input0	bit	1	0		
Input1	bit	1	0		
Input2	bit	1	0		
Input3	bit	1	0		

Figuur 11.25

6) For the **DO** data group, add the data **Output0**, **Output1**, **Output2**, and **Output3**. Take **Output0** for example. Do the data configuration as shown in Figuur 11.26 and Figuur 11.27.



Data Setting

Data: Output0 ☐ Alias

Data Type: array

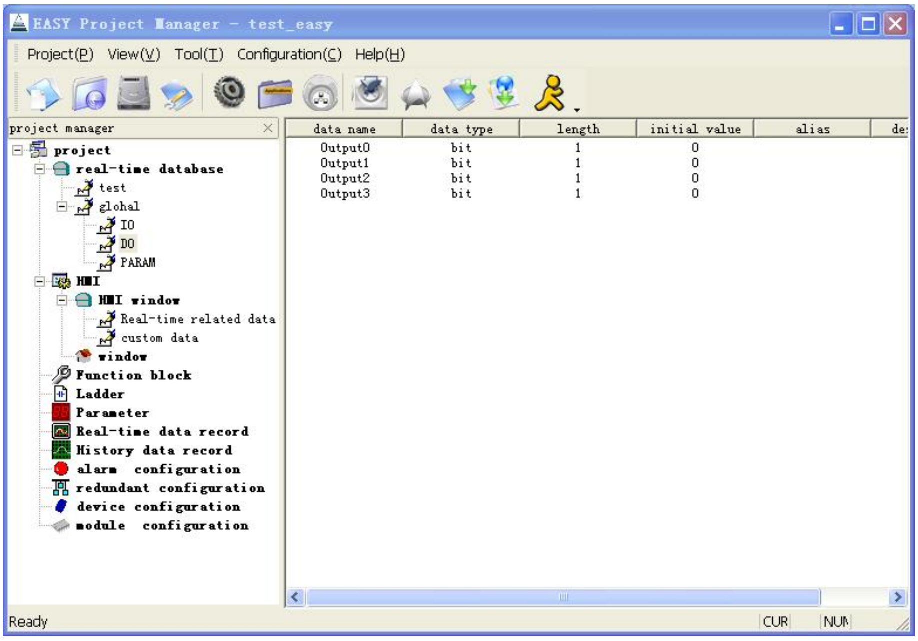
Data Length: 1

Initial Value: 0

Alias:

Description:

Figuur 11.26

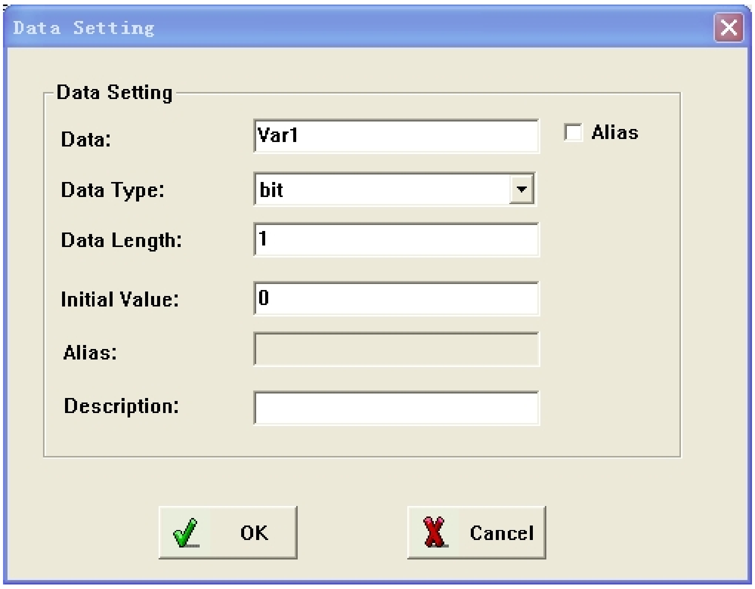

EASY Project Manager - test_easy
Project(P) View(V) Tool(T) Configuration(C) Help(H)
project manager
Ready
CUR
NUM

- project
  - real-time database
    - test
    - global
      - IO
      - DO
      - PARAM
  - HMI
    - HMI window
      - Real-time related data
      - custom data
    - window
  - Function block
  - Ladder
  - Parameter
  - Real-time data record
  - History data record
  - alarm configuration
  - redundant configuration
  - device configuration
  - module configuration

data name	data type	length	initial value	alias	de:
Output0	bit	1	0		
Output1	bit	1	0		
Output2	bit	1	0		
Output3	bit	1	0		

Figuur 11.27

7) For the **PARAM** data group, add the data **Var1**, **Var2**, **Var3**, and **Var4**.
   Take **Var1** for example. Do the data configuration as shown in Figuur 11.28 and Figuur 11.29.



Data Setting

Data: Var1 □ Alias

Data Type: bit

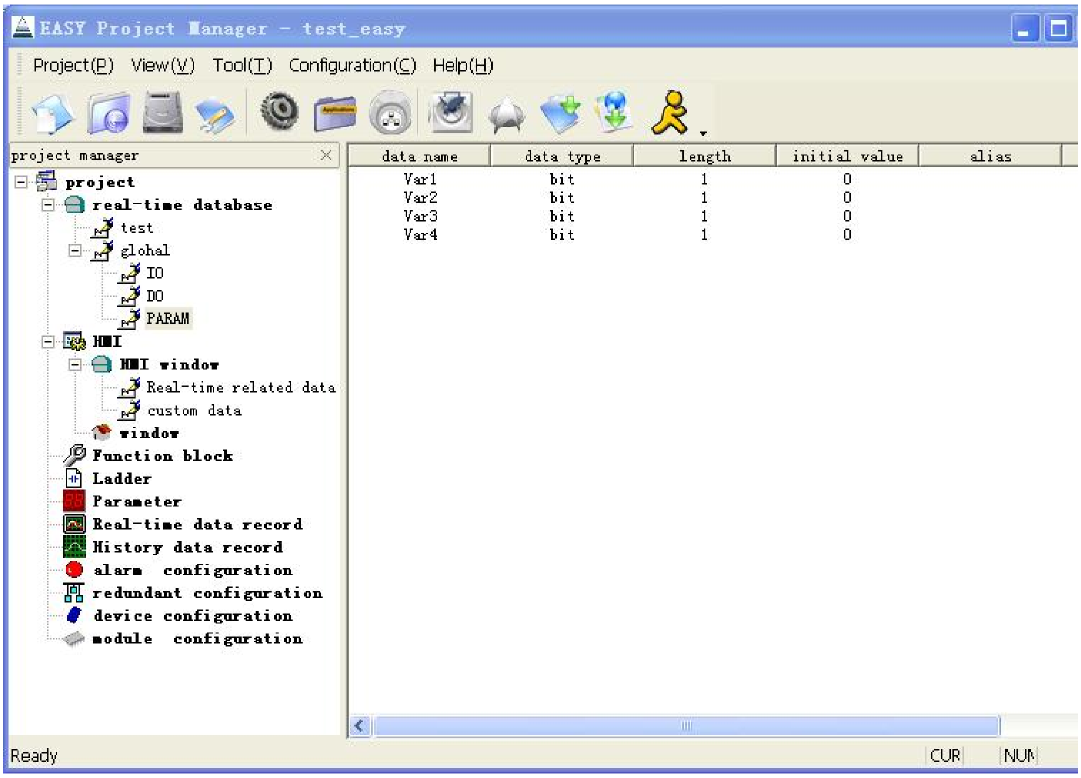
Data Length: 1

Initial Value: 0

Alias:

Description:

Figuur 11.28



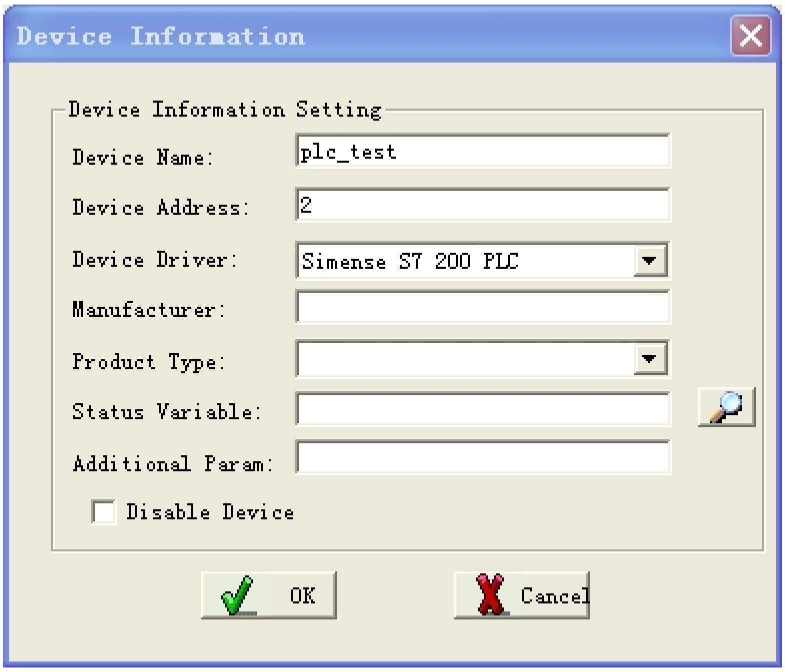
data name	data type	length	initial value	alias
Var1	bit	1	0	
Var2	bit	1	0	
Var3	bit	1	0	
Var4	bit	1	0	

Figuur 11.29

2. Under the **Device Configuration** node, add a communication link **link1** of the **Serial Port** type. ConFiguur the link parameters as follows:

Baud Rate: 9600; Data Bit: 8; Stop Bit: 1; Parity Check: Even

After configuring the link, add a device for the link, as shown in Figuur 11.30.



Device Information

Device Information Setting

Device Name: plc_test

Device Address: 2

Device Driver: Simense S7 200 PLC

Manufacturer:

Product Type:

Status Variable:

Additional Param:

□ Disable Device

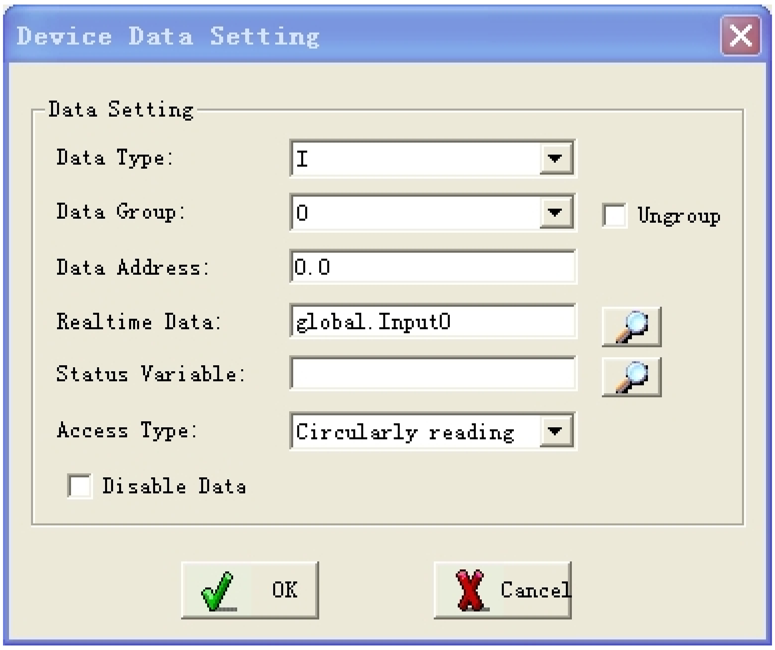
Figuur 11.30

The parameters for device configuration in Figuur 11.30 are described as follows:

- **Device Name**: **plc_test** (The device name can be defined to anything of your choice.)
- **Device Adres**: S7-200 (The default device adres is 2.)
- **Device Driver Application**: Select **Siemens S7 200 Series PLC**. This setting is very crucial. Make sure that all the settings satisfy your needs. However, considering that the S7-200 series all use the PPI communication protocol, **Siemens S7 200 Series PLC** is selected for all S7-200 PLCs as the driver application.

3. Add data for device communication. The actual types of data which might exist in the PLC can all be associated to the variables in the real-time database.

1) **PLC input register I**: Associate the data from I0.0 to I0.3 individually to the real-time database data from global.Input0 to global.Input3. Take I0.0 for example. ConFiguur thee data as shown in Figuur 11.31.



Device Data Setting

Data Setting

Data Type: I

Data Group: 0 □ Ungroup

Data Address: 0.0

Realtime Data: global.Input0

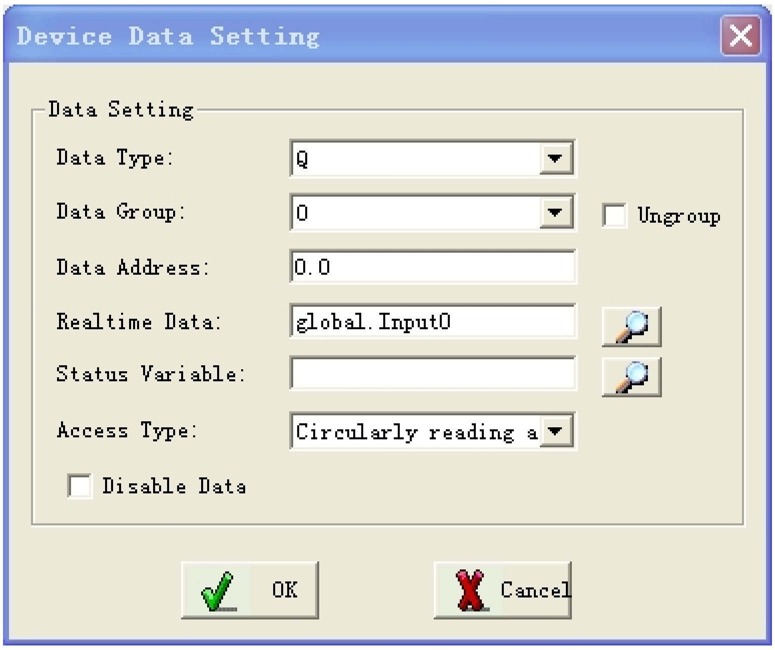
Status Variable:

Access Type: Circularly reading

□ Disable Data

Figuur 11.31

2) **PLC input register Q**: Associate the data from Q0.0 to Q0.3 individually to the real-time database data from global.Output0 to global.Output3. Take Q0.0 for example. ConFiguur the data as shown in Figuur 11.32.



Device Data Setting

Data Setting

Data Type: Q

Data Group: 0 □ Ungroup

Data Address: 0.0

Realtime Data: global.Input0

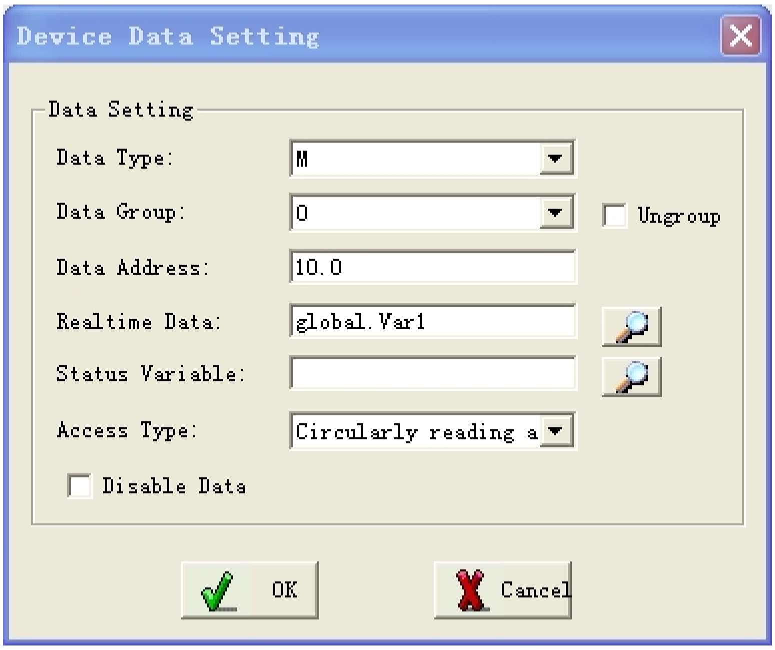
Status Variable:

Access Type: Circularly reading a

□ Disable Data

Figuur 11.32

3) **PLC bit register M**: Associate the data from M10.1 to M10.4 individually to the real-time database data from global.Var1 to global.Var4. Take M10.1 for example. ConFiguur the data as shown in Figuur 11.33.



Data Type: M

Data Group: 0 □ Ungroup

Data Address: 10.0

Realtime Data: global.Var1

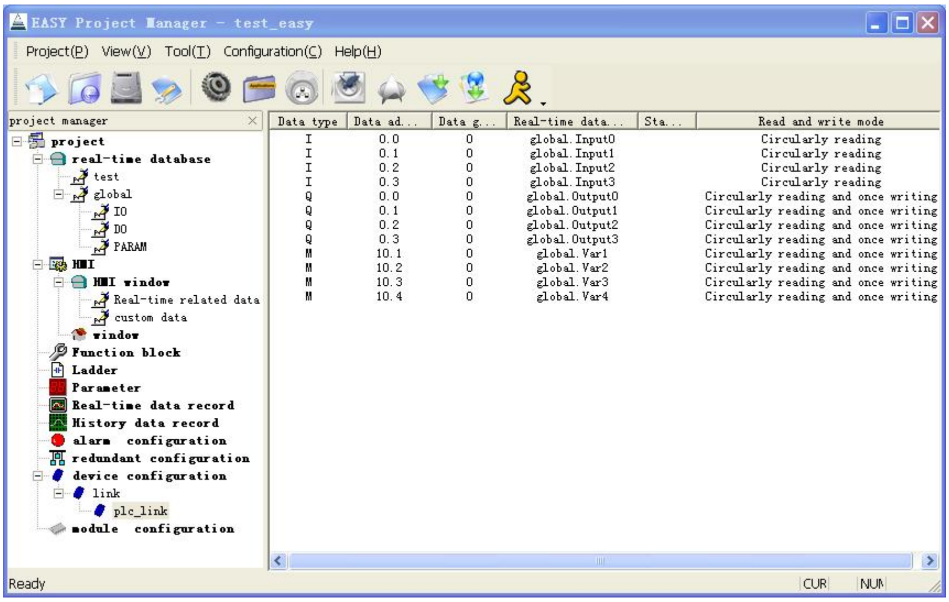
Status Variable:

Access Type: Circularly reading a

□ Disable Data

Figuur 11.33

The conFiguurd communication data will be listed as shown in Figuur 11.34.



Data type	Data ad...	Data g...	Real-time data...	Sta...	Read and write mode
I	0.0	0	global.Input0		Circularly reading
I	0.1	0	global.Input1		Circularly reading
I	0.2	0	global.Input2		Circularly reading
I	0.3	0	global.Input3		Circularly reading
Q	0.0	0	global.Output0		Circularly reading and once writing
Q	0.1	0	global.Output1		Circularly reading and once writing
Q	0.2	0	global.Output2		Circularly reading and once writing
Q	0.3	0	global.Output3		Circularly reading and once writing
M	10.1	0	global.Var1		Circularly reading and once writing
M	10.2	0	global.Var2		Circularly reading and once writing
M	10.3	0	global.Var3		Circularly reading and once writing
M	10.4	0	global.Var4		Circularly reading and once writing

Figuur 11.34

Chapter 12 Development of Control Function Blocks

12.1 Overview

For the monitoring system, the monitoring hardware devices are compulsory. These devices can be PLCs, DCSs, intelligent instruments or PC-based industrial computers (referred to as PC-Based devices hereinafter). They can also be the currently popular Fieldbus Systems. In the control system, these devices take the leading roles for control; the input and output of process signals can only be transferred to the field devices through these hardware devices.

For the existing control systems, there are two methods of implementing control policies, as described below:

- Method 1

  The PLCs, DCSs, and intelligent instruments all have internal ready-to-use control algorithms. The preset control solutions and policies can be implemented after some configuration is done.

  However, this method has its disadvantages, as listed below:

  - Firstly, the internal control policies of these control devices are hard to be modified.

    Some control policies are even not allowed to be modified during the system operation.
  - Secondly, the control capabilities of these control devices are very limited. They are only capable of implementing some simple and routine control.

    For example, the logic operations of DCSs are of low speed, and the control algorithms of PLCs are of limited varieties. These disadvantages seriously restrict the device performance to be brought into full play.
- Method 2

  These control devices can communicate conveniently with PCs, and use the various algorithms provided by some function blocks of the configuration software on the PC. This makes up for the inabilities of the computing and control capabilities of these control devices.

At present, none of the HMIs out there in the market are capable of solving the issues mentioned above. However, EASY manages to embed complicated control policies into interfaces of the HMI, and thus easily implement complicated control functions.

12.2 Basic Concepts

In the EASY applications, the main control functions are implemented through scripts or ladder diagrams. Besides, EASY provides various control function blocks. Each function block stands for an operation, algorighm, or variable (which are the basic execution elements of policies), similar to an integrated circuit block which has multiple inputs and outputs. Each input or output pin has a unique name, and the meaning and value range of each pin vary according to the type of the function block.

12.3 Architecture

Basic function blocks can be called repeatedly and are assigned with a name with each call. The sequence in which basic function blocks are executed is decided by how they are sorted under the **Function Block** node. In general, they are executed from top to bottom.

There are five categories of basic function blocks:

- Variable Function Blocks: provide variable links for the other function blocks.
- Mathematical Operation Function Blocks: carry out mathematical operations between various variables.
- Programming Control Function Blocks: carry out jumps between policies.
- Logic Function Blocks: carry out logic control and logic operations.
- Control Algorithm Function Blocks: carry out operations and controls according to standard control algorithms.

All input and output data for function blocks come from the real-time database. All you need to do is just to associate the input and output variables of each function block to the real-time database data. The function blocks are executed periodically, which ensures the control reliability.

A complete function block is mainly composed of the following parts:

- Function Block Name: specifies the name of the function block; can be user-defined.
- Function Block Type: specifies the type of the function block, such as Arithmetics, Comparison, and Logic.
- Function Block Sub-Type: specifies the sub-type of the function block; for example, the arithmetic function block has four sub-types, Addition, Subtraction, Multiplication, and Division.

- Function Block Allow Variable: refers to a bit data in the real-time database. When the value of the variable is 1, the function block will be executed; when the value is 0, the function block will not be executed.
- Function block input and output variables: A function block may have multiple input and output variables. The function block is executed as follows: The data is obtained from the input variable and operated according to the type of the function block, and the calculated result is then saved to the output variable.

  For example, the arithmetic addition function block has two input variables IN1 and IN2, and one output variable OUT. The function block is executed as this: OUT=IN1+IN2, where IN1 and IN2 each can be associated with a real-time data, and OUT can be associated with another real-time data. After the function block is executed, the value of the real-time data associated with OUT equals to the sum of the two real-time data associated with IN1 and IN2.

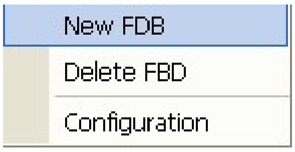
12.4 Operational Instructions

12.4.1 Adding a Function Block

To add a control block, do as follows:

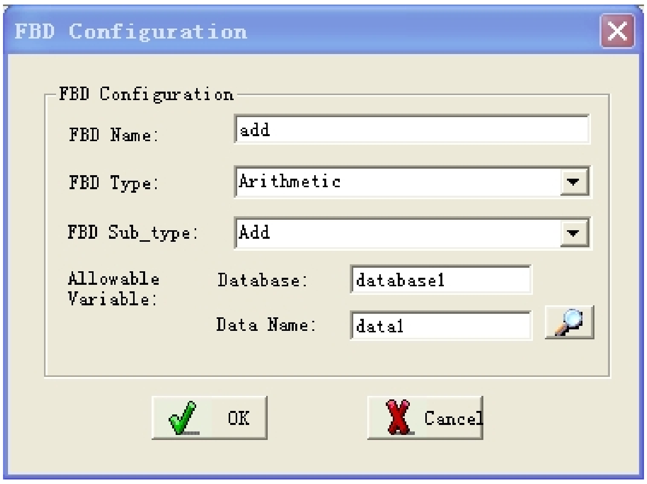
1) Select **Function Block** on the left side of the **Project Manager** window and right-click on it.

   You will see a right-click menu as shown in Figuur 12.1.



Figuur 12.1

2) Select **Add a Function Block**, and you will see a dialog box as shown in Figuur 12.2.



FBD Configuration

FBD Name: add

FBD Type: Arithmetic

FBD Sub_type: Add

Allowable Variable: Database: database1

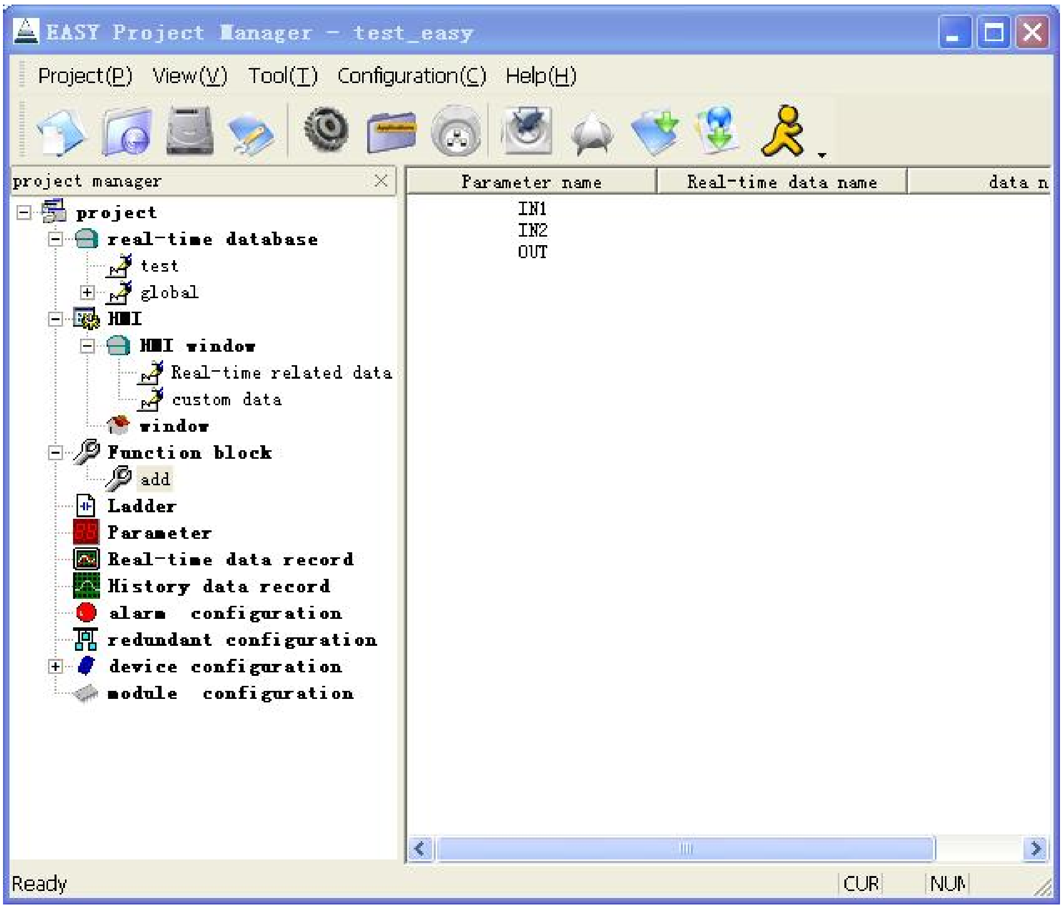
Data Name: data1

Figuur 12.2

The configuration parameters in Figuur 12.2 are described as follows:

- **Function Block Name**: defines the name of the function block.
- **Function Block Type**: defines the specific type of the function block, such as Arithmetic, Comparison, and Logic.
- **Function Block Sub-Type**: defines the sub-type of the function block; for example, the Arithmetic Function Block has four sub-types, which are Addition, Subtraction, Multiplication, and Division.
- **Allow Variable**: refers to a bit data in the real-time database. When the value of this variable is 1, the function block will be executed; when the value is 0, the function block will not be executed.

After adding a function block, select it in the navigator on the left side of the **Project Manager** window, and you will see a list of data displayed on the right side of the window, as shown in Figuur 12.3.



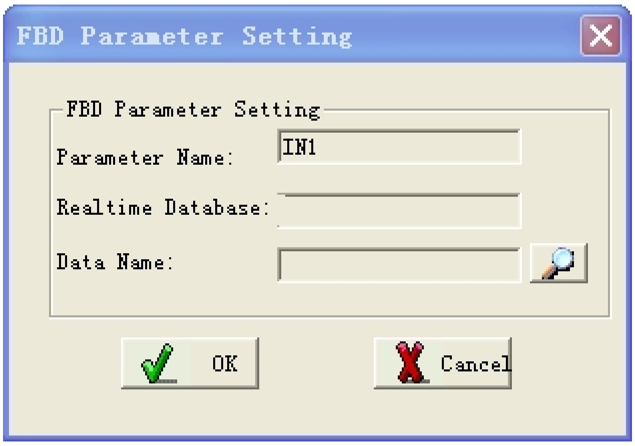
Parameter name	Real-time data name	data n
IN1		
IN2		
OUT		

Figuur 12.3

Right-click on the parameter **IN1**, and you will see a right-click menu as shown in Figuur 12.4.

Figuur 12.4

Select **Modify** from the right-click menu or double-click on the parameter **IN1**, and you will see a dialog box as shown in Figuur 12.5.



FBD Parameter Setting

Parameter Name: IN1

Realtime Database:

Data Name:

Figuur 12.5

Click on the icon 🔍 on the right side, select the associated data fromm the real-time database, ad then click on **OK** to save the configuration.

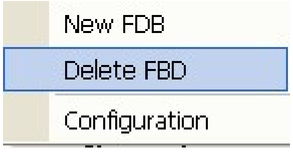
You can conFiguur the parameters **IN2** and **OUT** in the same way as the parameter **IN1**.

12.4.2 Deleting a Function Block

To delete a function block, do as follows:

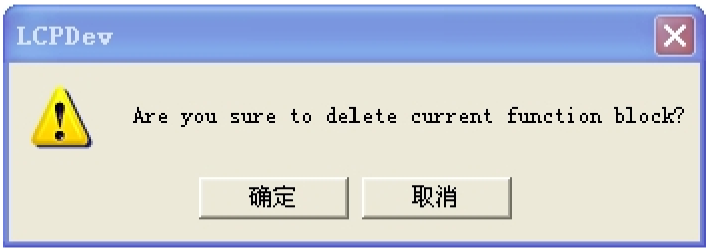
1) Select the added function block on the left side of the **Project Manager** window, and right-click on it.

   You will see a right-click menu as shown in Figuur 12.6.



Figuur 12.6

2) Select **Delete a Function Block**, and you will see a dialog box as shwon in Figuur 12.7.


LCPDev

Are you sure to delete current function block?

Figuur 12.7

3) Click on **OK**.

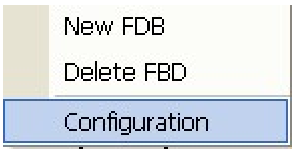
   The selected function block will be deleted.

12.4.3 Configuring a Function Block

To conFiguur a function block, do as follows:

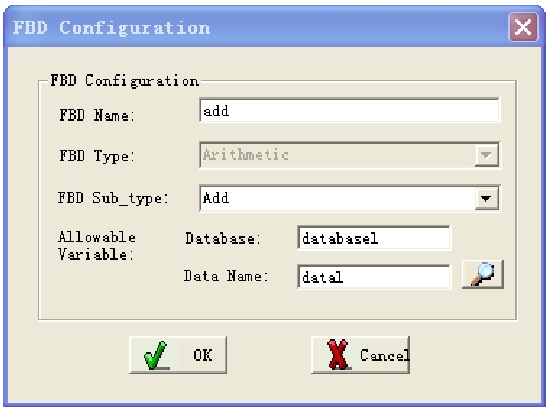
1) Select the added function block on the left side of the **Project Manager** window, and right-click on it.

   You will see a right-click menu as shown in Figuur 12.8.



Figuur 12.8

2) Select **ConFiguur**, and you will see a dialog box as shown in Figuur 12.9.



FBD Configuration

FBD Configuration

FBD Name: add

FBD Type: Arithmetic

FBD Sub_type: Add

Allowable Variable: Database: database1

Data Name: data1

Figuur 12.9

3) After you do all the configuration, click on **OK** to save the settings.

12.5 Input and Output Instructions

12.5.1 Arithmetic Function Blocks

12.5.1.1 Addition

Function: To add two operands; for example, OUT = IN1 + IN2.

Input:

Input	Data Type	Description
IN1	All data types except string	Summand of addition
IN2	All data types except string	Addend of addition

Output:

Output	Data Type	Description
OUT	All data types except string	Sum of two operands

12.5.1.2 Subtraction

Function: To subtract one one operand from aother; for example, OUT = IN1 - IN2.

Input:

Input	Data Type	Description
IN1	All data types except string	Minuend of subtraction
IN2	All data types except string	Subtrahend of subtraction

Output:

Output	Data Type	Description
OUT	All data types except string	Difference between two operands

12.5.1.3 Multiplication

Function: To multiply one operand by another; for examle, OUT = IN1 × IN2.

Input:

Input	Data Type	Description
IN1	All data types except string	Multiplicand of multiplication
IN2	All data types except string	Multiplicator of multiplication

Output:

Output	Data Type	Description
OUT	All data types except string	Product of two operands

12.5.1.4 Division

Function: To divide one operand by another; for example, OUT = IN1 / IN2.

Input:

Input	Data Type	Description
IN1	All data types except string	Dividend of division
IN2	All data types except string	Divisor of division

Output:

Output	Data Type	Description
OUT	All data types except string	Quotient of two operands

12.5.2 Comparison Function Blocks

12.5.2.1 Greater Than

Function: To carry out the Greater Than operation between two operands.

Input:

Input	Data Type	Description
IN1	All data types except string	Comparand 1
IN2	All data types except string	Comparand 2

Output:

Output	Data Type	Description
OUT	Bit data only	If IN1 is greater than IN2, then OUT = 1; otherwise, OUT = 0.

12.5.2.2 Equal To Or Greater Than

Function: To carry out the Equal To Or Greater Than operation between two operands.

Input:

Input	Data Type	Description
IN1	All data types except string	Comparand 1
IN2	All data types except string	Comparand 2

Output:

Output	Data Type	Description
OUT	Bit data only	If IN1 is equal to or greater than IN2, then OUT = 1; otherwise, OUT = 0.

12.5.2.3 Less Than

Function: To carry out the Less Than operation between two operands.

Input:

Input	Data Type	Description
IN1	All data types except string	Comparand 1
IN2	All data types except string	Comparand 2

Output:

Output	Data Type	Description
OUT	Bit data only	If IN1 is less than IN2, then OUT = 1; otherwise, OUT = 0.

12.5.2.4 Equal To Or Less Than

Function: To carry out the Equal To Or Less Than operation between two operands.

Input:

Input	Data Type	Description
IN1	All data types except string	Comparand 1
IN2	All data types except string	Comparand 2

Output:

Output	Data Type	Description
OUT	Bit data only	If IN1 is equal to or less than IN2, then OUT = 1; otherwise, OUT = 0.

12.5.2.5 Equal To

Function: To carry out the Equal To operation between two operands.

Input:

Input	Data Type	Description
IN1	All data types except string	Comparand 1
IN2	All data types except string	Comparand 2

Output:

Output	Data Type	Description
OUT	Bit data only	If IN1is equal to IN2, then OUT = 1; otherwise, OUT = 0.

12.5.2.6 Not Equal To

Function: To carry out the Not Equal To operation between two operands.

Input:

Input	Data Type	Description
IN1	All data types except string	Comparand 1
IN2	All data types except string	Comparand 2

Output:

Output	Data Type	Description
OUT	Bit data only	If IN1 is not equal to IN2, then OUT = 1; otherwise, OUT = 0.

12.5.3 Type Conversion Function Blocks

Function: OUT = IN

If the OUT and IN data are of the same type, the data value will be copied directly; otherwise, the data type conversion will be required between the OUT and IN data. For example, to convert the int data to the float data.

Input:

Input	Data Type	Description

Input	Data Type	Description
IN1	All data types	

Output:

Output	Data Type	Description
OUT	All data types	

Note: The data type conversion might affect the data accuracy; for example, when a float data is converted to an int data. For the string data, copying among the string data is allowed. However, it is not allowed to convert the string data to other types of data; for example, a string data cannot be converted to an int or float data.

12.5.4 Linear Conversion Function Blocks

Function: To implement the linear conversion between the input IN and the output OUT. The data range for IN is MININ - MAXIN, and the corresponding data range for OUT is MINOUT – MAXOUT.

Input:

Input	Data Type	Description
IN	All data types except string	
MININ	All data types except string	Minimum value allowed for input
MAXIN	All data types except string	Maximum value allowed for input
MINOUT	All data types except string	Minimum value allowed for output
MAXOUT	All data types except string	Maximum value allowed for output

Output:

Output	Data Type	Description
OUT	All data types except string	OUT = (IN – MININ)/ (MAXIN-MININ)*(MAXOUT-MINOUT) + MINOUT

12.5.5 Logic Function Blocks

12.5.5.1 Logical AND

Function: To carry out the Logical AND operation between two operands.

Input:

Input	Data Type	Description
IN1	Bit data only	
IN2	Bit data only	

Output:

Output	Data Type	Description
OUT	Bit data only	Result of the Logical AND operation

12.5.5.2 Logical OR

Function: To carry out the Logical OR operation between two operands.

Input:

Input	Data Type	Description
IN1	Bit data only	
IN2	Bit data only	

Output:

Output	Data Type	Description
OUT	Bit data only	Result of the Logical OR operation

12.5.5.3 Logical Exclusive OR

Function: To carry out the Logical Exclsive OR operation between two operands.

Input:

Input	Data Type	Description
IN1	Bit data only	
IN2	Bit data only	

Output:

Output	Data Type	Description
OUT	Bit data only	Result of the Logical Exclusive OR operation

12.5.5.4 Logical NOT

Function: To carry out the Logical NOT operation between two operands.

Input:

Input	Data Type	Description
IN1	Bit data only	

Output:

Output	Data Type	Description
OUT	Bit data only	Result of the Logical NOT operation

12.5.6 PID Function Blocks

Regarding process control, the PID controller (also known as PID regulators), which controls according to the percentage of deviation (P), Integral (I) and differential (D) control, is the most widely used as a automatic controller. It has the following advantages:

- Simple working principles
- Easy to realize
- Widely applicable
- Control parameters independent of each other
- Selection of relatively simple parameters

Besides, in theory the PID controller is proved the best control system for the typical process control objects, First-Order Plus Dead-Time and Second-Order Plus Dead-Time. During the past 20 plus years, there came out some complicated control algorithms which can only be realized on the computer. However, at present, the PID controller is still the most widely used control algorithm even in process computer control.

Input:

Input	Data Type	Description
SP	All data types except string	Setpoint
PV	All data types except string	Process variable
AV	All data types except string	Output value of the PID algorithm
MAXOUT	All data types except string	Maximum value allowed for the output
MINOUT	All data types except string	Minimum value allowed for the output
MV	All data types except string	Output value of the manipulated variable. When AM=1, then AV=MV.
KP	All data types except string	Proportional coefficient
TI	All data types except string	Integral time constant (unit: s)
TD	All data types except string	Derivative time constant (unit: s)
IS	All data types except string	Deviation integral value (namely, cumulative deviation)
AM	Bit variables	Manual manipulation flag. • When AM=1, the manipulation will be manual. • When AM=0, the manipulation will be automatic.

PN:	Bit variables	• When PN=1, it indicates the positive action. • When PN=0, it indicates the negative action.

Output:

Output	Data Type	Description
AV	All data types except string	Output value
IS	All data types except string	Deviation integral value (namely, cumulative deviation)

Execution Function: To execute the PID algorithm.

Note: The PID algorithm provides bumpless switch between the automatic and manual PID controls. Some variables, such as AV and MV, are not only input variables but also output variables.

The PID controller carries out the PID algorithm through the setpoint (SP) and the process variable. The PID control loop works in two modes, MAN and AUT. In the MAN mode, the PID control loop works as a manual regulator; in the AUT mode, the PID control loop carries out the PID algorithm automatically. The SP can be defined by the operator.

When the PID control loop works in the MAN mode, the SP picks up the automatic tracking function equal to PV, facilitating the bumpless switch from the MAN mode to the AUT mode.

1. Calculate the PID control output

Proportional Term = Proportion * (Current Deviation – Last Deviation)

Integral Term = Proportion * Deviation * Collection Cycle / integral time constant

Differential Term = Proportion * differentiating time constant * (Current Deviation - 2*Last Deviation + Last Two Deviations)/Collection Cycle.

- If it is a positive action, then Output = Last Output + Proportional Term + Integral Term + Differential Term.
- If it is a negative action, then Output = Last Output - Proportional Term - Integral Term - Differential Term.

Judge whether the output and deviation exceed the preset limit as follows. If so, process it accordingly.

$$\Delta U_i = K_p\left[e_i - e_{i-1} + \frac{T}{T_i}e_i + \frac{T_d}{T}\left(e_i - 2e_{i-1} + e_{i-2}\right)\right]$$

2. Select PID parameters

The setting of the parameters for the digital PID regulator is similar to that for the analog PID regulator. Decide what parameters to select for the regulator based on the requirements of process on the control performance. The impact of each parameter on the system performance is briefly described below:

- Impact of Proportional Coefficient (P) on the System Performance

  With the increase of the proportional coefficient, the system reacts more sensitively and with faster speed and less steady-state error. However, too much increase of the P value will result in more number of vibrations and longer regulating time.

  If the P value is too big, the system will become unstable; if it is too small, the system performance will become slow.

  The P value can be negative, mainly determined by the actuator, sensor, and the characteristics of the conrol object. If the P value has an incorrect symbol, the status of the object (indicated by the PV value) will become further and further away from the target control state (indicated by the SP value). In this case, you will need to change the symbol of the P value to the opposite.
- Impact of Integral Control on the System Performance

  The integral control will reduce the system stability. The smaller the I value is, the stronger the integral control, and then the unstabler the system. However, it eliminates the steady-state error and enhances the system cotrol accuracy.
- Impact of Differential Control on the System Performance

  The differential control improves the dynamic characteristics. However, the bigger the D value is, the bigger the overshoot, and thus the shorter the regulating time; the smaller the D value, the bigger the overshoot as well, and the longer the regulating time. Only when the D value is appropriate, the overshoot is small, and thus reducing the regulating tme.

12.5.7 First-Order Model Function Blocks

Function: To conFiguur a model object for the first-order system.

Input:

Input	Data Type	Description

Input	Data Type	Description
IN	All data types except string	Input value
KP	All data types except string	Amplification coefficient
T	All data types except string	Time constant (unit: s)
MAXOUT	All data types except string	Maximum output value
MINOUT	All data types except string	Minimum output value

Output:

Output	Data Type	Description
OUT	All data types except string	Output value

12.5.8 Differential Function Blocks

Function: To realize the differential function. The differential of OUT equals to that of IN.

Input:

Input	Data Type	Description
IN	All data types except string	Input variable

Output:

Output	Data Type	Description
OUT	All data types except string	Output value

12.5.9 Integral Function Blocks

Function: To realize the integral function. The integral of OUT equals to that of IN.

Input:

Input	Data Type	Description
IN	All data types except string	Input variable
SW	Int variable	Defines how the integral function works, as follows: • When SW=0, the integral result is reset, namely, OUT=0. • When SW=1, the integral becomes cumulative. • When SW=2, the integral function stops while the integral result is stored.

Output:

Output	Data Type	Description
OUT	All data types except string	Output value

12.6 System Variables for Function Blocks

Database Name	Variable Name	Data Type	Default Value	Description
system	FbdEnable	bit	1	• When the variable value is 1, the function block is enabled. • When the variable value is 0, the function block is disabled. In other words, none of the conFiguurd function blocks will be executed.
	FbdCycleTime	ulong	100	Defines the cycle in which the function blocks are executed (unit: ms).
	FbdHeartbeat	bit		Defines the heartbeat of the function blocks during the operation. The value of this variable

Database Name	Variable Name	Data Type	Default Value	Description
				switches between 0 and 1 repeatedly during the operation of the function blocks. The value switches once all function blocks are executed.

Chapter 13 Ladder Diagram Programming

13.1 Overview

To better satisfy the needs of industrial control, the EASY HMI integrates as well the soft PLC function besides the cofiguration functions provided by the commonly-used configuration software. With the soft PLC function, the EASY HMI is capable of realizing some functions which can only be implemented on the PLC. Therefore, in some cases, the EASY HMI can replace the PLC; for example, you can directly connect to IO modules and implement the control over these modules completely from the HMI.

The soft PLC function of the EASY HMI is implemented through the ladder diagram programming, which is widely-used in the PLC industry at present. Because of this, if you are an engineer or technician familiar with the PLC, you can start working with the EASY HMI right away without any special technical training.

During the ladder diagram programming, you can not only use directly the various ladder diagram components provided by the system, but also access all the data in the real-tiime database. Therefore, it can be concluded that the EASY HMI is more preferable compared with the traditional PLC regarding rapidly configuring powerful control projects.

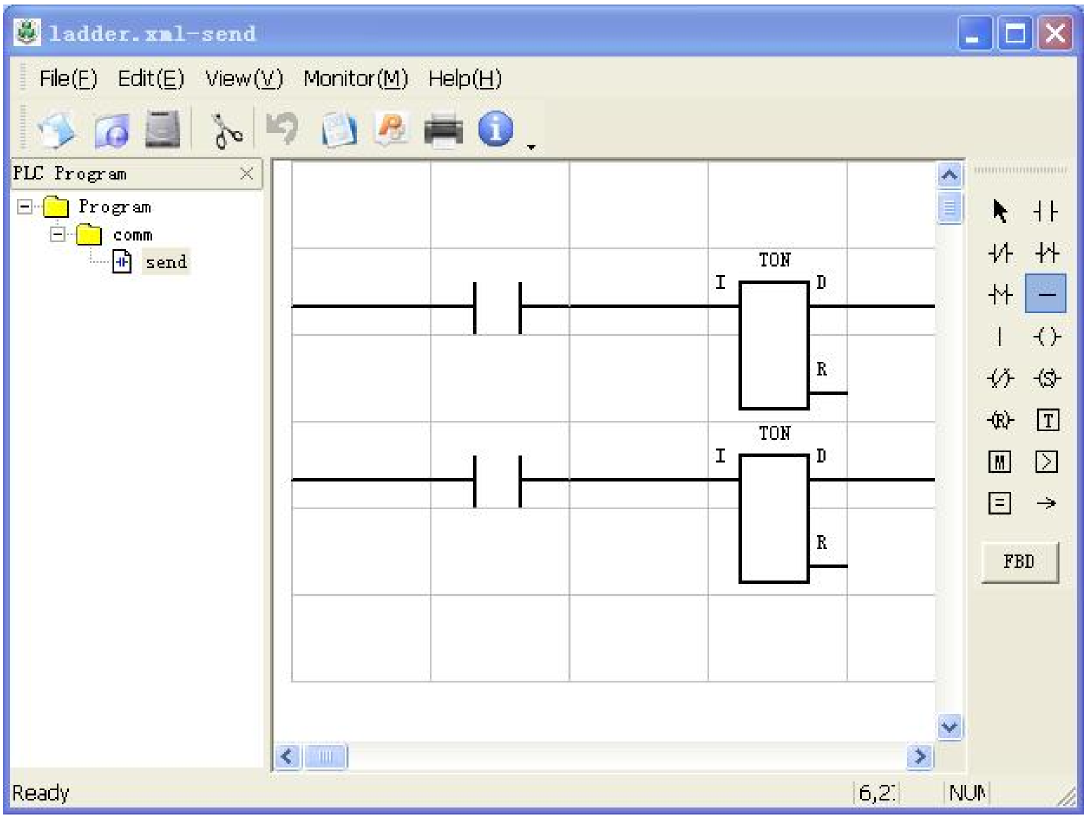
13.2 Creating a Ladder Diagram Program

13.2.1 Overview

To create a ladder diagram program, do as follows:

Double-click on the **Ladder Diagram** node in the navigator on the left side of the **Project Manager** window.

You will see the **Ladder Diagram Editor** window as shown in Figuur 13.1.



Figuur 13.1

The ladder diagram programming is organized hierachically in two levels, program segments and program blocks. A ladder diagram program can be composed of one or more program segments, and a program segment can be composed of one or more program blocks. You can realize the ladder diagram logic in each program block.

The ladder diagram is designed to be composed of several program segments, because in this way it allows you to specify an operational variable for each program segment to control whether to run this program segment.

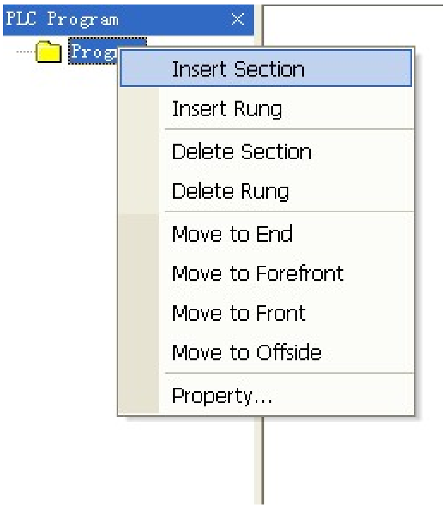
The following sections describe in details how to create program segments and program blocks.

13.2.2 Create a Program Segment

To create a program segment, do as follows:

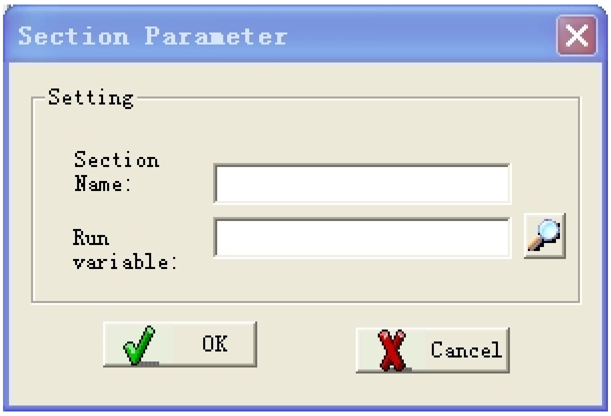
1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select **Program** on the left side and right-click on it.

You will see a right-click menu as shown in Figuur 13.2.



Figuur 13.2

2) Select **Create a Program Segment**, and you will see a dialog box as shown in Figuur 13.3.



Figuur 13.3

The parameters in Figuur 13.3 are described as follows:

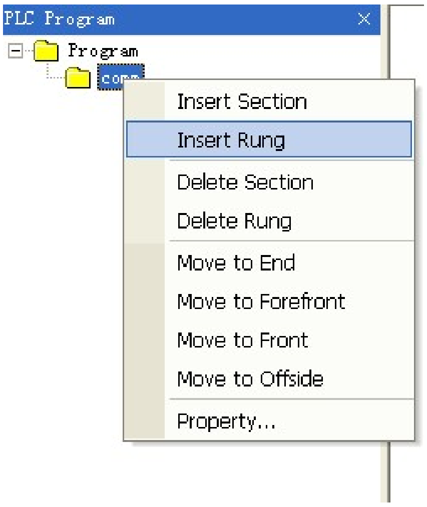
- **Program Segment Name**: defines the name of the program segment to be created.
- **Operational Variable**: defines whether to run this program segment.
  This operational variable is an int variable. When the value is 0, the program segment will not run; when the value is not 0, the program segment will run. If no operational variable is defined, then the program segment will run by default.

13.2.3 Creating a Program Block

After you create a program segment, you can add program blocks for it as follows:

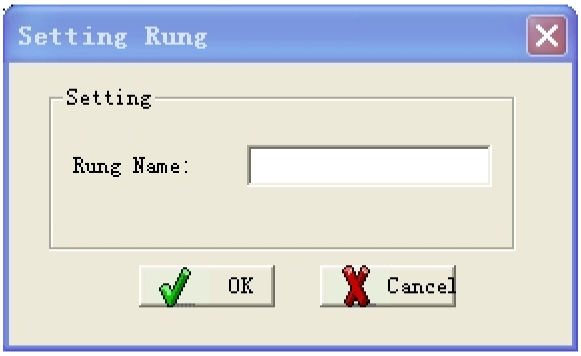
1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select the program segment for which you want to add a program block, and right-click on it.

You will see a right-click men as shown in Figuur 13.4.



Figuur 13.4

2) Select **Add a Program Block**, and you will see a dialog box as shown in Figuur 13.5.



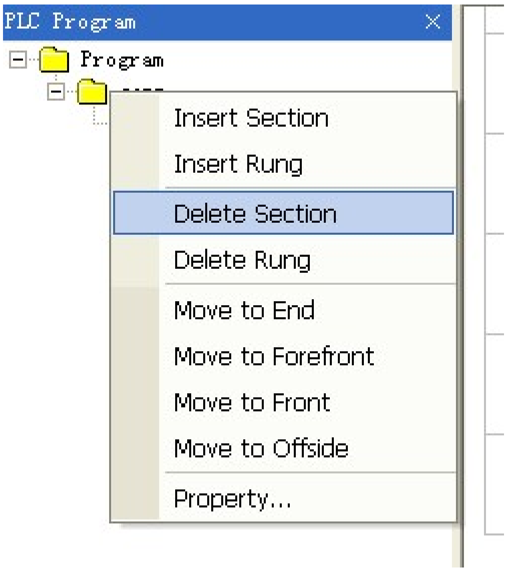
Figuur 13.5

3) Set the name for the program block, and click on **OK**.

13.2.4 Deleting a Program Segment

To delete an existing program segment, do as follows:

1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select the program segment to be deleted on the left side of the window, and right-click on it.
You will see a right-click menu as shown in Figuur 13.6.



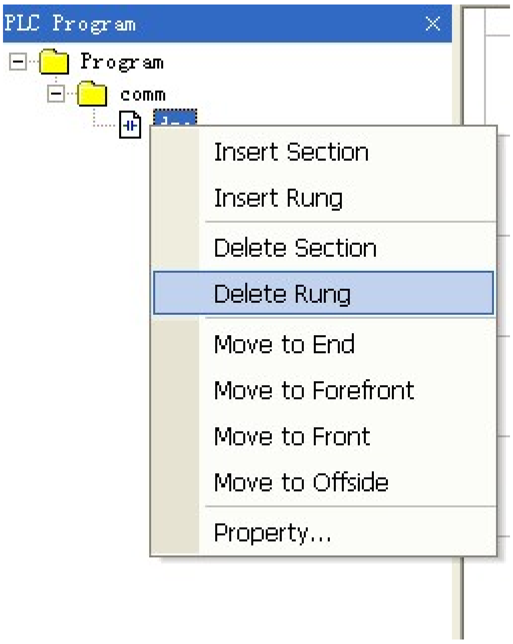
Figuur 13.6

2) Select **Delete a Program Segment**, and the selected program segment will be deleted.

13.2.5 Deleting a Program Block

To delete an existing program block, do as follows:

1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select the program block to be deleted on the left side of the window, and right-click on it.
You will see a right-click menu as shown in Figuur 13.7.



Figuur 13.7

2) Select **Delete a Program Block**, and the selected program block will be

deleted.

13.2.6 Moving a Program Block

Program blocks in the ladder diagram are executed from top to bottom in the same sequence as they are sorted under the **Program Segment** node. You can change the execution sequence by dragging the program block to a different location under the **Program Segment** node.

13.2.6.1 To the End

You can move a program block to the most bottom of the list under the **Program Segment** node.

Do as follows:

1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select the program block to be moved on the left side of the window, and right-click on it.
2) On the right-click menu displayed, select **Move to The End**.

13.2.6.2 To the Beginning

You can move a program block to the beginning of the list under the **Program Segment** node.

Do as follows:

1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select the program block to be moved on the left side of the window, and right-click on it.
2) On the right-click menu displayed, select **Move to The Beginning**.

13.2.6.3 To the Previous Layer

You can move a program block to the previous layer within the same **Program Segment** node.

Do as follows:

1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select the program block to be moved on the left side of the window, and right-click on it.
2) On the right-click menu displayed, select **Move to The Previous Layer**.

13.2.6.4 To the Next Layer

You can move a program block to the next layer within the same **Program Segment** node.

Do as follows:

1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, select the program block to be moved on the left side of the window, and right-click on it.
2) On the right-click menu displayed, select **Move to The Next Layer**.

13.2.7 Editting a Ladder Diagram Program

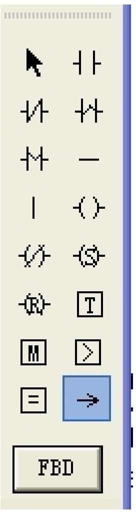
You can edit the ladder dialgram programming for a defined program block.

Do as follows:

1) In the **Ladder Diagram Editor** window as shown in Figuur 13.1, double-click on the program block to be editted on the left side of the window.
   You can see a rectanglar grid on the right side of the window.
2) Drag and put ladder diagram components inside the grid according to your needs.
   Each component takes only one column. Some components can spread across a few rows according to the needs.

13.2.7.1 Adding a Component

On the right side of the **Ladder Diagram Editor** window as shown in Figuur 13.1 is located a tool set, which lists all ladder diagram components supported by EASY, as shown in Figuur 13.8.



Figuur 13.8

To add a component, do as follows:

1) Select the component control from the tool set.
2) Drag and put this component in the appropriate grid.

   Depending on the function of the component, some components can only be placed in the last column of the grid (for example, open loops -( )-), while some others are not allowed to be placed in the last column of the grid (for example, normally open contacts -| |-).

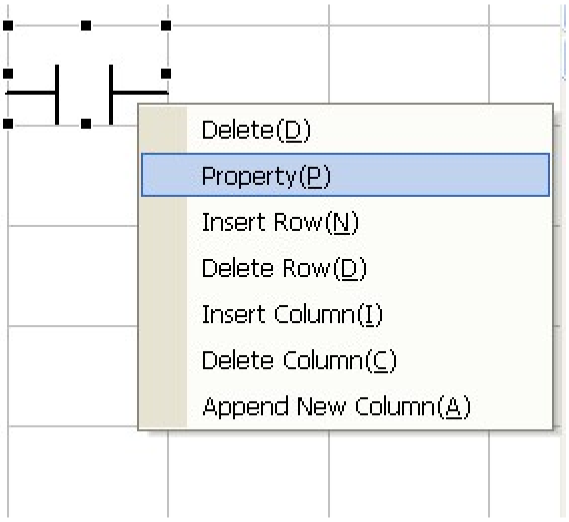
Note: After adding a component, you must conFiguur the properties of this component before using it.

13.2.7.2 Editting a Component

After adding a component, you need to edit the properties of this component.

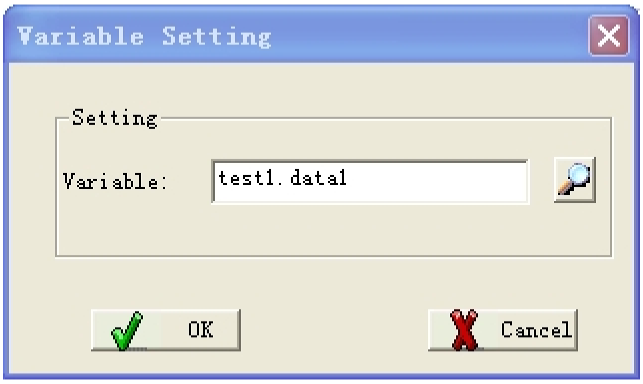
Do as follows:

Double-click on the component for which you want to edit the properties or right-click on the component, and then select **Properties**, as shown in Figuur 13.9.



Figuur 13.9

The **Property Setting** dialog box displayed varies from component to component. For normally open contacts, the **Property Setting** dialog box is as shown in Figuur 13.10.



Variable Setting

Setting

Variable: test1.data1

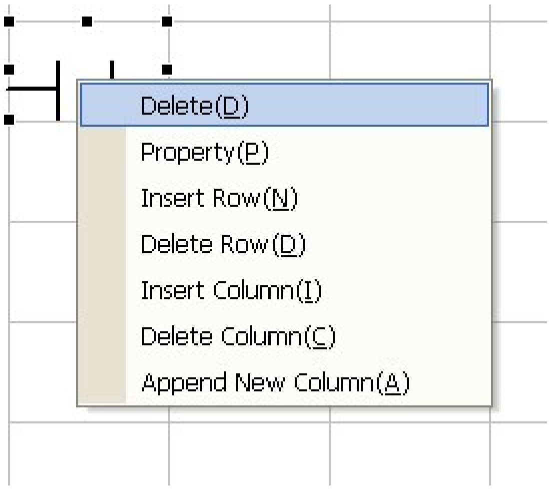
Figuur 13.10

13.2.7.3 Deleting a Component

To delete a component, do as follows:

1) Select the component to be deleted.

2) Press the **DEL** on the keyboard, or right-click on the component and select **Delete**, as shown in Figuur 13.11.



Figuur 13.11

13.2.7.4 Inserting a Row

You can insert a new row before a selected row.

Do as follows:

1) Select a row and then right-click on it.

2) Select **Insert a Row** on the right-click menu.

To insert a row before the last row, you need to select **Add a Row**.

13.2.7.5 Deleting a Row

You can select a row and delete it, on the condition that the selected row does not have any components attached.

To delete a row, do as follows:

1) Select a row and right-click on it.
2) Select **Delete a Row** on the right-click menu.

13.2.7.6 Inserting a Column

You can insert a column before the selected column.

Do as follows:

1) Select a column ad right-click on it.
2) Select **Insert a Column**.

13.2.7.7 Deleting a Column

You can select a column and delete it, on the condition that the selected column does not have components attached.

Do as follows:

1) Select a column and right-click on it.
2) Select **Delete a Column** on the right-click menu.

13.2.7.8 Adding a Row at the End

You can add a row before the last row.

Do as follows:

1) Right-click anywhere in the grid.
2) Select **Add a Row** on the right-click menu.

13.3 Components in the Ladder Diagram

13.3.1 Connecting Line

The connecting line is for connecting components of the ladder diagram.

At present, the system has three types of connecting lines, as described below:

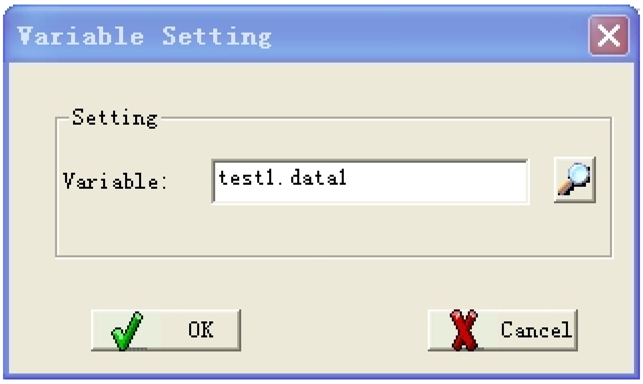
- Horizontal Lines ═: Horizontally connecting the components. These lines occupy only one cell.

- Vertical Lines |: Vertically connecting the components.
- Horizontal Connect-To Lines →: Horizontally connecting the components. These lines may occupy multiple cells, namely, all the cells between the current cursor location and the next component.

13.3.2 Digital Input

Digital input components include four types, Normally Open Contacts ┤├, Normally Closed Contacts ┤/├, Rising Edges ┤↑├, Falling Edges ┤↓├. None of these four types of components are allowed to be placed in the last column of the grid.

After you add a digital input component in the grid, double-click on the component to conFiguur the associated variables in the dialog box as shown in 13.12.



Variable: test1.data1

Figuur 13.12

The functions of and instructions for these four types of components are described in details in the table below.

Component	Function	Special Instruction
Normally Open Contact ┤├	• When the value of the variable to which this component is associated is 0, the contact state is 0. • When the value of the associated variable is non-0, the contact state is 1, namely, OUT=IN & VAL.	• If no associated variable is specified, the output is always equal to the input, namely, OUT=IN. • If you set the associated variable to the constant 1, the output is always equal to the input, namely, OUT=IN. • If you set the associated variable to the constant 0, then the output is always 0, namely, OUT=0.
Normally	• When the value of the variable	• If no associated variable is specified,

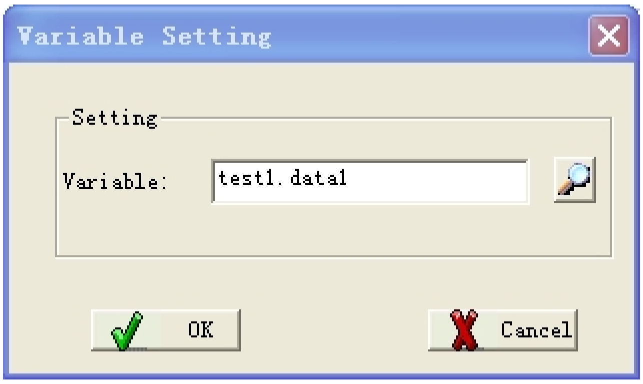
Closed Contact -|/|-	to which this component is associated is 0, the contact state is 1. - When the value of the associated variable is non-0, the contact state is 0, namely, OUT=IN & !VAL.	the output is always equal to the opposite of the input, namely, OUT=!IN. - If you set the associated variable to the constant 1, the output is always 0, namely, OUT=0. - If you set the associated variable to the constant 0, the output is always equal to the input, namely, OUT=IN.
Rising Edge -|↑|-	When the value of the associated variable changes from 0 to 1, the contact state is 1; otherwise, the contact state is 0, namely, OUT=IN & VAL.	- If no associated variable is specified, the output is 1 when the input changes from 0 to 1, namely, OUT=IN. - If the associated variable is set to the constant 1 or 0, the output is always 0, namely, OUT=0.
Falling Edge -|↓|-	When the value of the associated variable changes from 1 to 0, the contact state is 1; otherwise, the contact state is 0, namely, OUT=IN & VAL	- If no associated variable is specified, the output is 1 when the input changes from 1 to 0, namely, OUT=IN. - If the associated variable is set to the constant 1 or 0, the output is always 0, namely, OUT=0.

(Note: In the table above, IN stands for the component input, VAL for the value of the associated variable, and OUT for the component output.)

13.3.3 Digital Output

Digital output components include four types, Open Loops -( )-, Closed Loops -(/)-, Setting -(S)-, and Reset -(R)-. These four types of components are only allowed to be placed in the last column of the grid.

After you add a digital output component in the grid, double-click on the component to conFiguur the associated variables in the dialog box as shown in Figuur 13.13.



Variable Setting

Setting

Variable: test1.data1

OK

Cancel

Figuur 13.13

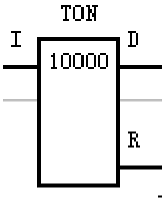
The functions of and instructions for these four types of components are described in details in the table below.

Component	Function
Open Loop -( )-	• When the loop input is 0, the value of the varialbe to which the loop is associated is 0. • When the loop input is non-0, the value of the associated variable is 1.
Closed Loop -(/)-	• When the loop input is 0, the value of the associated variable is 1. • When the loop input is non-0, the value of the associated variable is 0.
Setting -(S)-	• When the loop input is non-0, the value of the associated variable is 1. • When the loop input is 0, the value of the associated variable remains the same.
Reset -(R)-	• When the loop input is non-0, the value of the associated variable is 0. • When the loop input is 0, the value of the associated variable remains the same.

13.3.4 Timer

The Timer component T is not allowed to be placed in the last column of the grid.

After you add a Timer component, the Timer component will be displayed in a cell of the grid, as shown in Figuur 13.14.



Figuur 13.14

I stands for Input. When the value of **I** is 1, the timer starts running; when the value of **I** is 0, the timer stops.

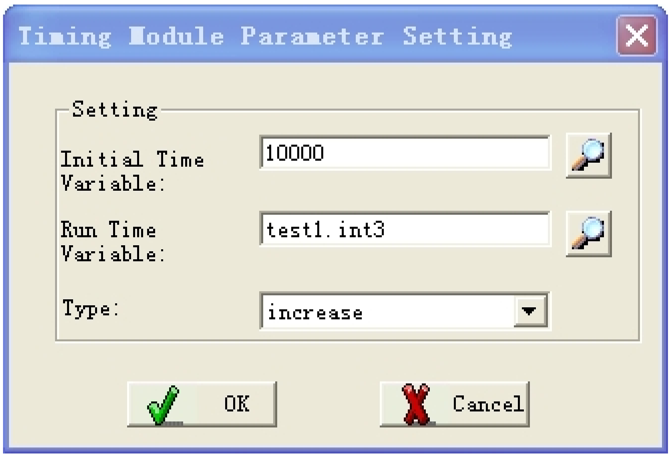
D stands for Done, which means the preset time for starting the timer is reached.

R stands for Running, which means the timer is running.

The functions of the timer are described as follows:

- When the value of **I** is 1, the timer starts running. During the timing period, the value of **D** is 0, and the value of **R** is 1. Once the timing period runs out, the value of **D** becomes 1, and the value of **R** becomes 0.
- When the value of **I** is 0, the timer stops running. In this case, the status of the timer, whether it is **Not Started**, **Started**, or **Stopped**, makes no difference. The values of both **D** and **R** are 0 as long as the value of **I** is 0.

Double-click on the Timer component, and you can conFiguur its properties in the dialog box as shown in Figuur 13.15.



Timing Module Parameter Setting

Setting

Initial Time Variable: 10000

Run Time Variable: test1.int3

Type: increase

Figuur 13.15

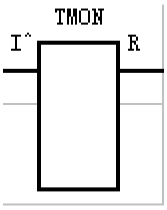
The configuration parameters in Figuur 13.15 are described as follows:

- **Initial Time Variable**: defines the time when the timer starts running.
  You can either enter an int constant, or associate it to a data in the real-time database.
- **Runtime Variable**: reflects the running of the timer.
  - ➢ If the timer is an incremental timer, you can observe that the value of this variable increases gradually from 0 to the value defined for **Initial Time Variable** during the running of the timer. The timer stops running once the value defined for **Initial Time Variable** is reached.
  - ➢ If the timer is a decremental timer, you can observe that the value of this variable decreases gradually from the value defined for **Initial Time Variable** to 0. The timer stops running once the value reaches 0. You do not need to specify this variable if you do not need to know the running status of the timer.
- **Type**: Two types are available for selection, **Incremental** and **Decremental**. For more details, see the description for **Running Time Variable**.

13.3.5 Single-Shot Trigger

Single-shot triggers M are not allowed to be placed in the last column of the grid.

After you add a single-shot trigger component, the component will be shown in a cell of the grid, as shown in Figuur 13.16.



Figuur 13.16

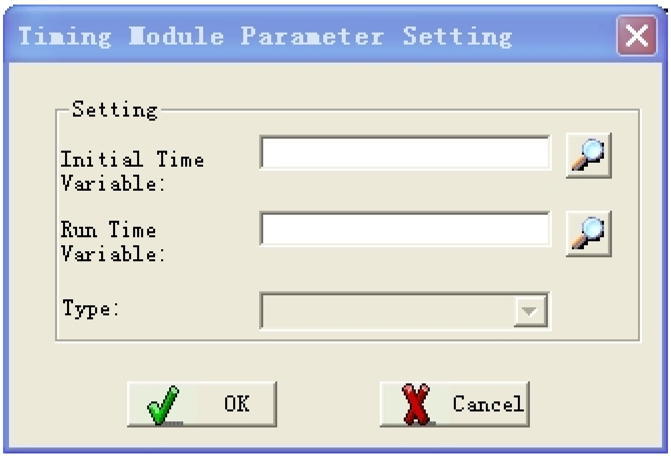
I stands for the input of the single-shot trigger. When the value of **I** is 1, the single-shot trigger starts running; when the value is 0, the trigger stops.

R stands for Running, which means the single-shot trigger is running.

The functions of the single-shot trigger are described as follows:

1. At the beginning of the operation of the ladder diagram, the single-shot trigger has not started running yet, and thus the value of **R** is 0.
2. During the operation of the ladder diagram, the single-shot trigger starts running when a rising edge (from 0 to 1) occurs to **I**, and the value of **R** becomes 1. The trigger will keep running until the preset running time runs out, regardless of the changes for **I**. Once the preset running time runs out, the value of **R** becomes 0.
3. When the operation of the single-shot trigger is complete, the single-shot trigger will start again when a rising edge (from 0 to 1) occurs to **I**, and step 2 will be repeated.

Double-click on the single-shot trigger component, and you can conFiguur its properties in the dialog box as shown in Figuur 13.17.



Figuur 13.17

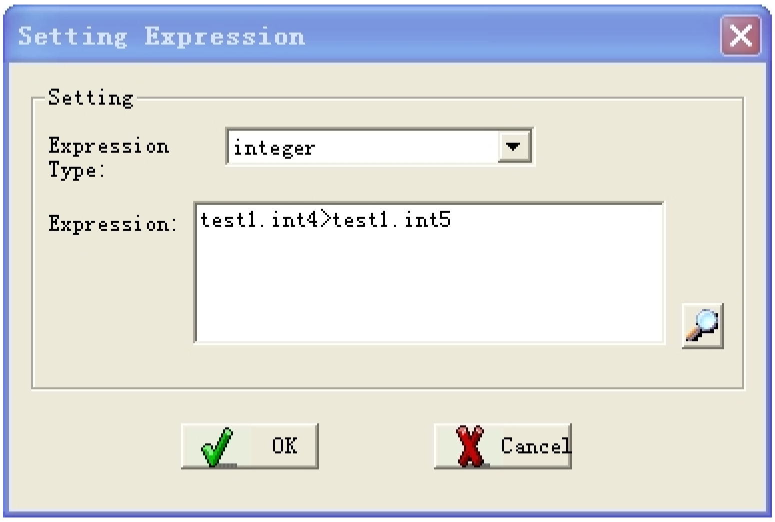
The configuration parameters in Figuur 13.17 are described as follows:

- **Initial Time Variable**: defines the time when the single-shot trigger starts running. You can either enter an Int constant, or associate it to a data in the real-time database.
- **Runtime Variable**: reflects the running of the single-shot trigger.
  During the running of the single-shot trigger, you can observe that the value of this variable increases gradually from 0 to the value defined for **Initial Time Variable**. The trigger stops running once the value defined for **Initial Time Variable** is reached.
  You do not need to specify this variable if you do not need to know the running status of the single-shot trigger.
- **Type**: This parameter is meaningless to the single-shot trigger.

13.3.6 Comparison Components

Comparison components ▷ are not allowed to be placed in the last column of the grid.

After you add a comparison component, double-click on it, and you can conFiguur its properties in the dialog box as shown in Figuur 13.18.



Expression Type: integer

Expression: test1.int4>test1.int5

Figuur 13.18

The parameters in Figuur 13.18 are described as follows:

- **Expression Type**: Two types of expressions are available, Int and Float.
- **Expression**: You can enter the expression for comparison or judgement.
  Follow the rules below when forming an expression:
  1. Use the following operators:
     - >: for More Than
     - <: for Less Than

- ◆ = or ==: for Equal To
- ◆ <>: for Not Equal To
- ◆ (and): for brackets

2. Do not use spaces in the expression.
3. You can include the variables from the real-time database in the expression, following the reference rule **Database Name.Real-Time Data Name**. Please be noted that the symbol $ is not included.

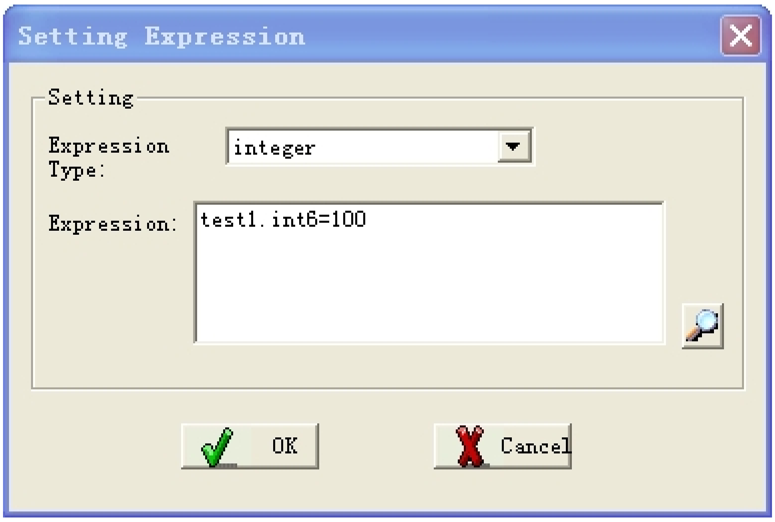
An example of the expression can be **test1.int4<>((test1.int5+5)*3)**.

For comparison components, the output is 0 when the input is 0. When the input is 1, the system will carry out the calculation according to the expression: if the result is TRUE, the output is 1; otherwise, the output is 0.

13.3.7 Assignment Components

Assignment components ⊟ are only allowed to be placed in the last column of the grid. You can define the assignment operation for these components.

After you add an assignment component in the grid, double-click on it, and you can conFiguur its properties in the dialog box as shown in Figuur 13.19.



Setting Expression

Setting

Expression Type: integer

Expression: test1.int6=100

Figuur 13.19

The configuration parameters in Figuur 13.19 are described as follows:

- **Expression Type**: Two types of expressions are available, Int and Float.
- **Expression**: defines the expression for assignment operation.
  Follow the rules below when forming an expression:
  1. Do not use spaces in the expression.
  2. You can include the variables from the real-time database in the expression, following the reference rule **Database Name.Real-Time Data Name**. Please be noted that the symbol $ is not included.

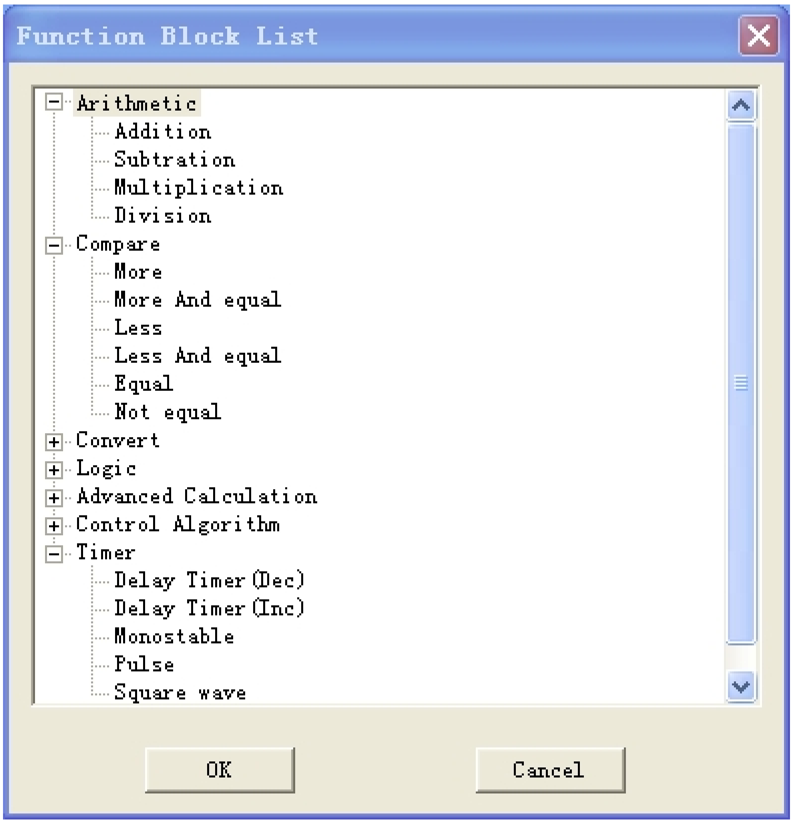
An example of the expression can be **test1.int6=100**.

For assignment components, when the input is 0, the assignment operation will not be carried out; when the input is 1, the assignment operation is carried out according to the expression.

13.3.8 Function Block Components

Besides the basic ladder diagram components, EASY provides function block components as well.

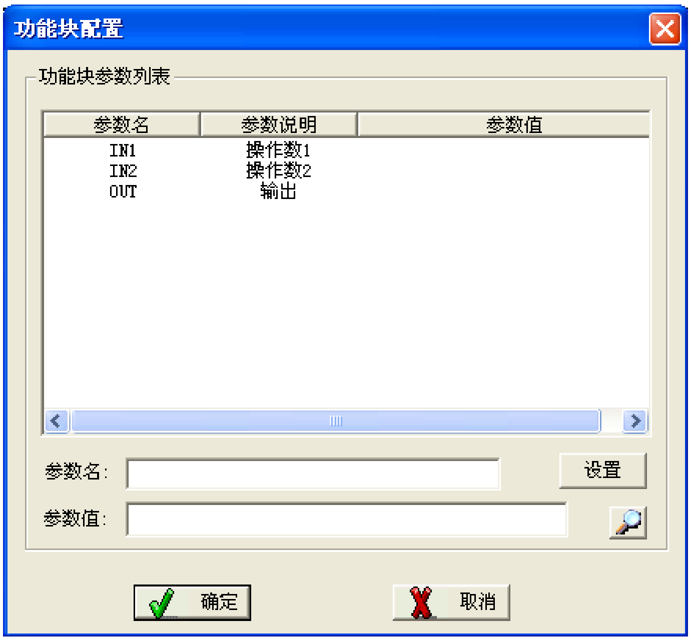
On the right side of the **Ladder Diagram Editor** window as shown in Figuur 13.1, click on the **Function Block** button in the tool set, and you can see a **Function Block List** dialog box as shown in Figuur 13.20.



Figuur 13.20

As dislayed in Figuur 13.20, function block components fall into seven categories according to functions, including Arithmetics, Comparison, Conversion, Logical Qubit Operation, Advanced Computing, Control Algorithm, and Timer. Each category is composed of several components; for example, the Arithmetics category has components such as Addition and Subtraction. To select a component, you just need to click on it and then click on **OK**. And then you can drag and place it in your target cell of the grid.

After you add a function block component, you need to conFiguur its properties. Take the Addition component as an example. Double-click on the component, and you can see a dialog box as shown in Figuur 12.21.



功能块配置

功能块参数列表

参数名	参数说明	参数值
IN1	操作数1	
IN2	操作数2	
OUT	输出	

Figuur 13.21

To conFiguur the parameters for the component properties, do as follows:

1) Select a parameter from the **Function Block Parameter List** in Figuur 13.21. The name of the selected parameter will be shown in the text box behind **Parameter Name**.

2) In the **Parameter Value** text box, enter your specified value, or click on the icon 🔍 on the right side to select a data from the real-time database.

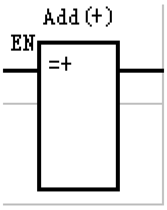
3) Click on the **Settings** button, and complete all the related settings.

4) Click on **OK** to save all the parameter settings.

13.3.8.1 Arithmetics

Arithmetics function blocks are composed of four types of components, including Addition, Subtraction, Multiplication, and Division, which carry out specifically the addition, subtraction, multiplication, and division operations. These components are only allowed to be placed in the last column of the grid.

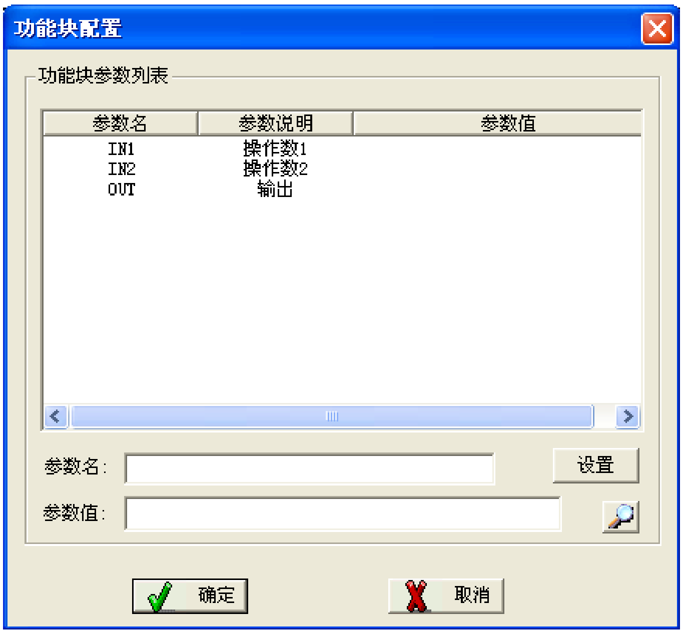
Take the Addition component for example. After you add an Addition component, the component will be displayed in the grid as shown in Figuur 13.22.



Figuur 13.22

EN stands for Enable. Only when the value of **EN** is 1, the arithmetic operation of the component will be executed; when the value of **EN** is 0, the operation will not be executed.

To conFiguur the properties of the component, double-click on the component, and you will see a dialog box as shown in Figuur 13.23.



参数名	参数说明	参数值
IN1	操作数1	
IN2	操作数2	
OUT	输出	

参数名：

参数值：

Figuur 13.23

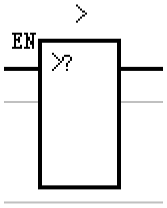
The parameters listed in Figuur 13.23 are described as follows:

- **IN1**: the first operand of the arithmetic operation. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database
- **IN2**: the second operand of the arithmetic operation. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database
- **OUT**: the result of the arithmetic operation. It must be a data in the real-time database. When the value of **EN** is 1, the arithmetic operation will be executed, and the result will be saved into a data in the real-time database.

13.3.8.2 Comparison

Comparision function blocks are composed of six types of components, including Greater Than, Equal To Or Greater Than, Less Than, Equal To Or Less Than, Equal To, and Not Equal To. These components are not allowed to be placed in the last column of the grid.

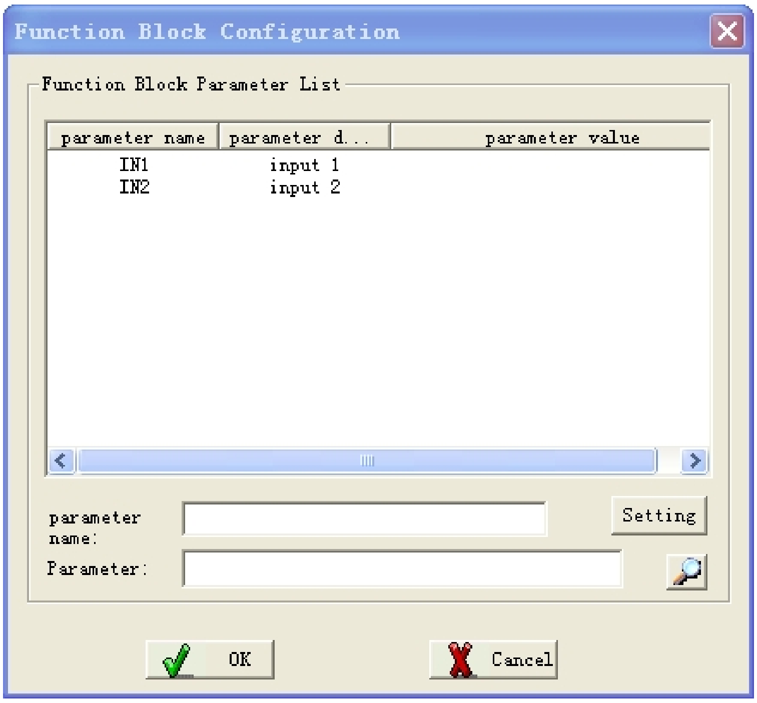
Take the Greater Than component for example. After you add a Greater Than component, the component will be displayed in the grid, as shown in Figuur 13.24.



Figuur 13.24

EN stands for Enable. Only when the value of **EN** is 1, the comparison operation of the component will be executed, and the output is set based on the comparison result. When the value of **EN** is 0, the comparison operation will not be executed, and the output is always 0.

After you add a comparison component, double-click on it, and you can conFiguur its properties in the dialog box as shown in Figuur 13.25.


Function Block Configuration

Function Block Parameter List

parameter name	parameter d...	parameter value
IN1	input 1	
IN2	input 2	

parameter name:

Parameter:

Figuur 13.25

The confguration parameters in Figuur 13.25 are described as follows:

- **IN1**: the first operand of the comparison operation. The value of this parameter can be one of the following:

- Int constant
- Float constant
- Data in the real-time database

- **IN2**: the second operand of the comparison operation. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database

13.3.8.3 Conversion

Conversion function blocks are composed of two types of components, Type Conversion (Assignment) and Linear Conversion. These components are only allowed to be placed in the last column of the grid.

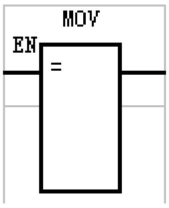
13.3.8.3.1 Type Conversion (Assignment)

The Type Conversion (Assignment) component has two parameters, the input parameter **IN** and the output parameter **OUT**. It works to convert the type of **IN** to the same as that of **OUT**. If **IN** and **OUT** are of the same data type, then the data value can be copies directly; otherwise, the data type conversion will be carried out between **OUT** and **IN**, for example, type conversion from Int to Float.

For type conversion, pay attention to the following aspects:

- Type conversion might affect data accurary; for example, when converting a Float data to an Int data.
- For String data, it is copying from the IN string to the OUT string.
- Type conversion is not allowed between a string data and a non-string data.

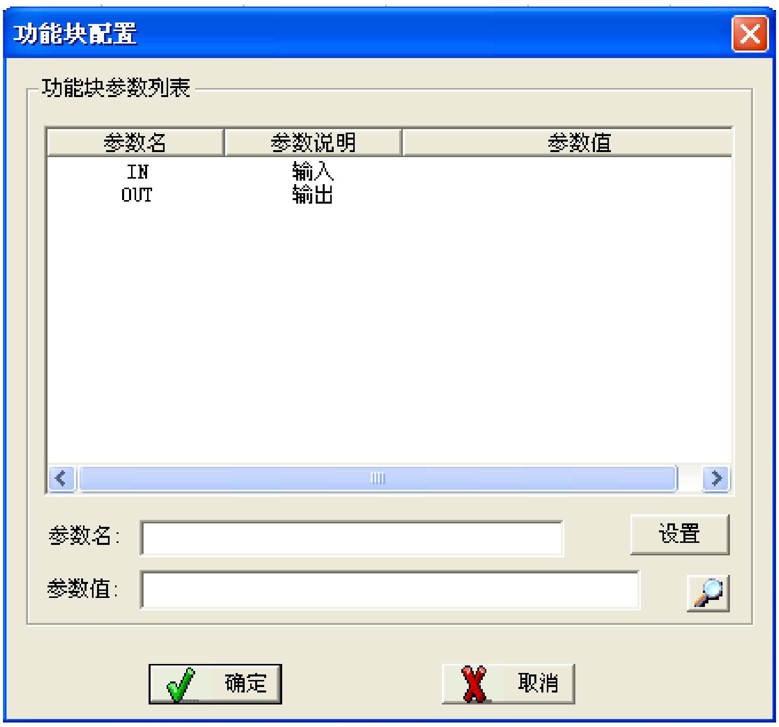
After you add a type conversion (assignment) component, the component will be displayed in the grid as shown in Figuur 13.26.



Figuur 13.26

EN stands for Enable. Only when the value of **EN** is 1, the type conversion operation will be executed; when the value of **EN** is 0, the type conversiion operation will not be executed.

To conFiguur the properties of the component, double-click on it, and you will see a dialog box as shown in Figuur 13.27.



功能块配置

功能块参数列表

参数名	参数说明	参数值
IN	输入	
OUT	输出	

参数名：

参数值：

Figuur 13.27

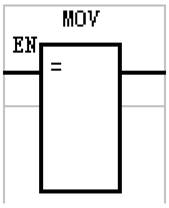
The configuration parameters are described as follows:

- **IN1**: the data whose type is to be converted. The value of the parameter can be of any data type.
- **OUT**: the result after type conversion. The value of this parameter must be a data in the real-time database.

13.3.8.3.2 Linear Conversion

The Linear Conversion component has six parameters, which are **IN**, **MININ**, MAXIN, MINOUT, MAXOUT, and OUT. It works to implement the linear conversion from **IN** to **OUT** (the value range of **IN** is MININ to MAXIN, and the value range of **OUT** is **MINOUT** to **MAXOUT**.

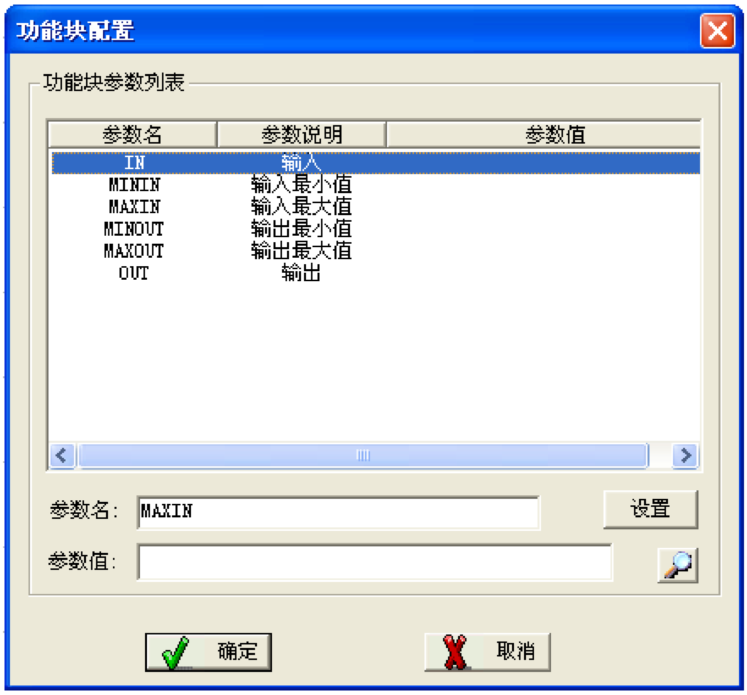
After you add a linear converstion component, the component will be displayed in the grid as shown in Figuur 13.28.



Figuur 13.28

EN stands for Enable. Only when the value of **EN** is 1, the linear conversion operation will be executed; when the value of **EN** is 0, the linear converstion operation will not be executed.

To conFiguur the properties of the component, double-click on it and you will see a dialog box as shown in Figuur 13.29.



参数名	参数说明	参数值
IN	输入	
MININ	输入最小值	
MAXIN	输入最大值	
MINOUT	输出最小值	
MAXOUT	输出最大值	
OUT	输出	

参数名: MAXIN

参数值:

Figuur 13.29

The configuration parameters in Figuur 13.29 are described as follows:

- **IN**: the data for which linear conversion is to be implemented. The value of this parameter can be on of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- **MININ**: the minimum value of the input range. The value of this parameter can be one of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- **MAXIN**: the maximum value of the input range. The value of this parameter can be one of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- **MINOUT**: the minimum value of the output range. The value of this parameter can be one of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- **MAXOUT**: the maximum value of the output range. The value of this parameter

can be one of the following:

- ➢ Int constant
- ➢ Float constant
- ➢ Data in the real-time database

- **OUT**: the result of the linear conversion. The value of this parameter must be a data in the real-time database.

13.3.8.4 Logical Qubit Operation

Logical Qubit Operation function blocks are composed of four types of components, including Logical AND, Logical OR, Logical Exclusive OR, and Logical NOT. They correspond individually to four operations in the C language, which are Bitwise And, Bitwise Or, Bitwise Xor, and Bitwise Not. See below for more details.

- Bitwise And (&): returns the bitwise and of the binary numbers of the two operands involved in the operation.

  You can get a 1 only when both binary digits at the same location of the binary number are 1; otherwise, you will get a 0.

  For example, “9&5” can be converted to “00001001 (the binary number of 9)&00000101 (the binary number of 5)”, and the operation result is “00000001 (the binary number of 1)”, namely, “9&5=1”.

- Bitwise Or (|): returns the bitwise or of the binary numbers of the two operands involved in the operation.

  You can get a 1 as long as one of the binary digits at the same location of the two binary numbers is 1; otherwise, you will get a 0.

  For example, “9|5” can be converted to “00001001 (the binary number of 9)|00000101 (the binary number of 5)”, and the operation result is “00001101 (the binary number of 13)”, namely, “9|5=13”.

- Bitwise Xor (^): returns the bitwise Xor of the binary numbers of the two operands involved in the operation.

  You can get a 1 when the two binary digits at the same location of the binary number are different from each other; otherwise, you will get a 0.

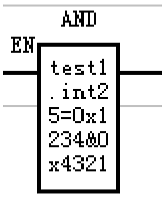
  For example, “9^5” can be converted to “00001001 (the binary number of 9)^00000101 (the binary number of 5)”, and the operation result is “00001100 (the binary number of 12)”, namely, “9^5=12”.

- Bitwise Not (~): returns the bitwise not of each binary digit of the binary number of the operand involved in the operation. In other words, the binary digits which are on will be turned off and those which are off will be turned on.

For example, “~9” can be converted to “~00001001 (the binary number of 9)”, and the operation result is “11110110 (the binary number of 246)”, namely, “~9=246”.

These four types of components are only allowed to be placed in the last column of the grid.

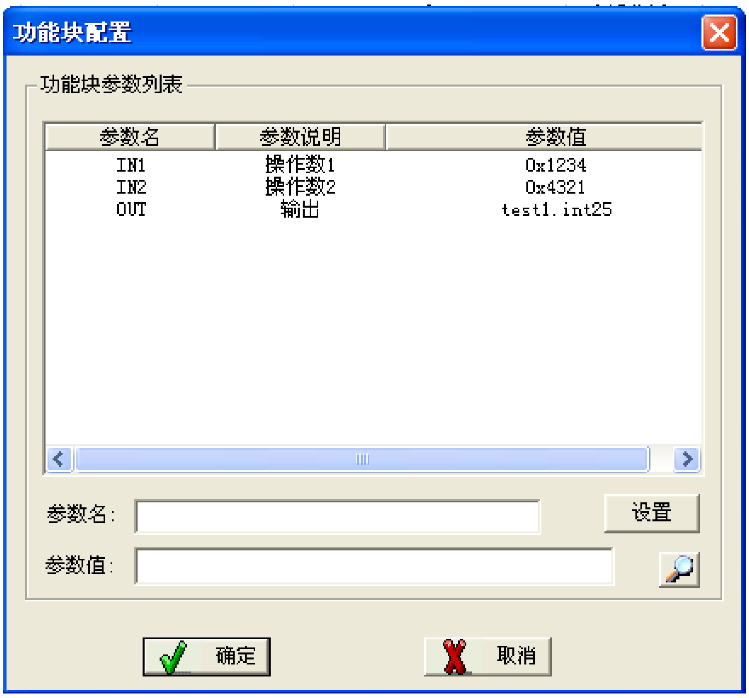
Take the Logical And component for example. After you add a logical AND component, the component will be displayed in the grid as shown in Figuur 13.30.



Figuur 13.30

EN stands for Enable. Only when the value of **EN** is 1, the logical operation of the component will be executed; when the value of **EN** is 0, the operation will not be executed.

To conFiguur the properties of the component, double-click on it, and you will see a dialog box as shown in Figuur 13.31.



功能块配置

功能块参数列表

参数名	参数说明	参数值
IN1	操作数1	0x1234
IN2	操作数2	0x4321
OUT	输出	test1.int25

参数名:

参数值:

Figuur 13.31

The configuration parameters in Figuur 13.31 are described as follows:

- **IN1**: the first operand of the logical operation. The value of this parameter can be one of the following:
  - Int constant (If starting with **0x**, it means the input is a hex data; for example, 0x1234.)
  - Data in the real-time database
- **IN2**: the second operand of the logical operation. This parameter is invalid for the Logical Not component. The value of this parameter can be one of the following:

    - Int constant (If starting with **0x**, it means the input is a hex data; for example, 0x1234.)
    - Data in the real-time database
- **OUT**: the result of the logical operation. It must be a data in the real-time database. When the value of **EN** is 1, the logical operation will be executed, and the result is saved into a data in the real-time database.

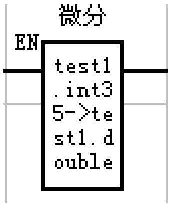
13.3.8.5 Advanced Computing

Advanced Computing function blocks are composed of two types of components, including Differential and Integral components. These components are only allowed to be placed in the last column of the grid.

13.3.8.5.1 Differential Components

The differential component has two parameters, including the input parameter **IN** and the output parameter **OUT**. It works to implement the differential function, namely, to make the differential of **OUT** equals to that of **IN**.

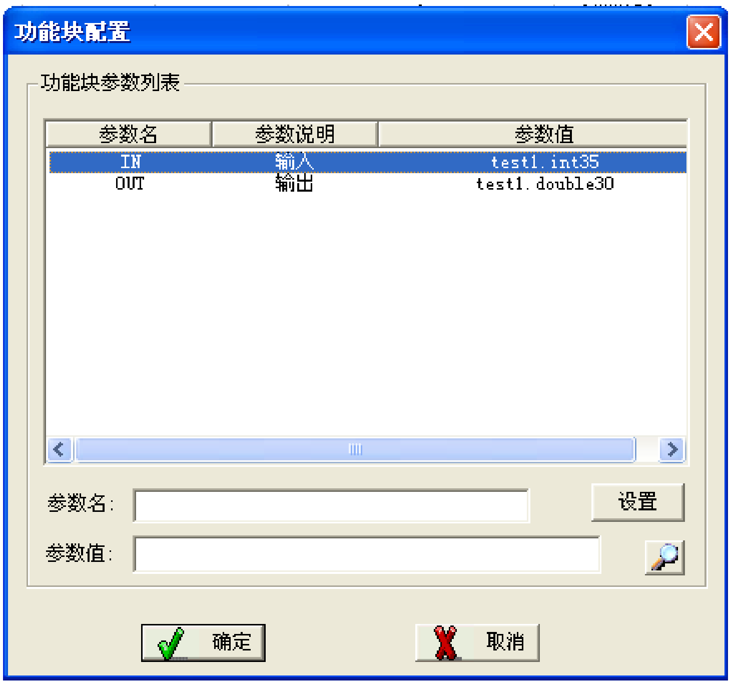
After you add a differential component, the component will be displayed in the grid as shown in Figuur 13.32.



Figuur 13.32

EN stands for Enable. Only when the value of **EN** is 1, the differential operation will be executed; when the value of **EN** is 0, the differential operation will not be executed.

To conFiguur the properties of the differential component, double-click on it and you will see a dialog box as shown in Figuur 13.33.



功能块配置

功能块参数列表

参数名	参数说明	参数值
IN	输入	test1.int35
OUT	输出	test1.double30

参数名：

参数值：

Figuur　13.33

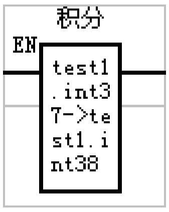
The configuration parameters in Figuur 13.33 are described as follows:

- **IN**: the data for which the differential operation is to be implemented. The value of this parameter must be a data in the real-time database.
- **OUT**: the result of the differential operation. The value of this parameter must be a data in the real-time database.

13.3.8.5.2 Integral Components

The Integral component has three parameters, which are **IN**, **OUT**, and **SW**. It works to implement the integral function, namely, to make the integral of **OUT** equal to that of **IN**.

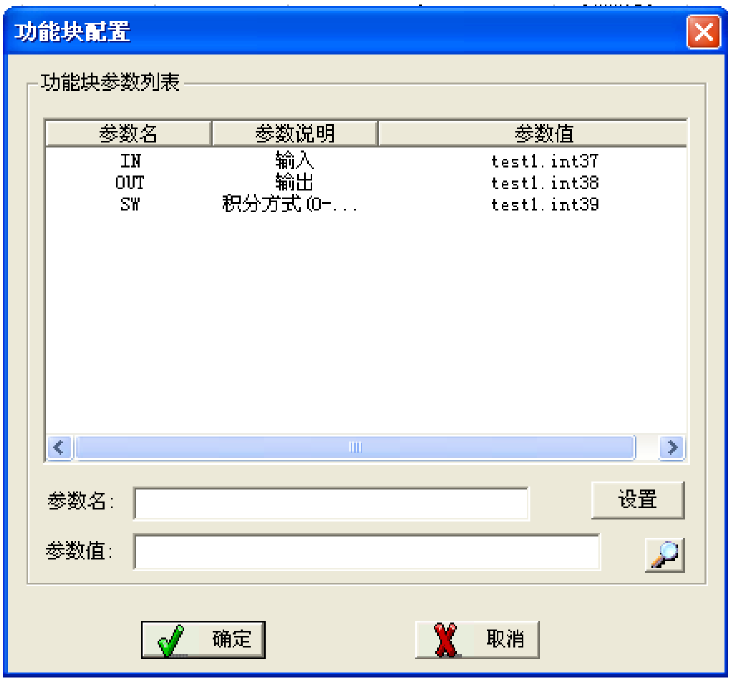
After you add an integral component, the component will be displayed in the grid as shown in Figuur 13.34.



Figuur　13.34

EN stands for Enable. Only when the value of **EN** is 1, the integral operation will be executed; when the value of **EN** is 0, the integral operation will not be executed.

To conFiguur the properties of the component, double-click on it, and you will see a dialog box as shown in Figuur 13.35.



功能块配置

功能块参数列表

参数名	参数说明	参数值
IN	输入	test1.int37
OUT	输出	test1.int38
SW	积分方式(0-...	test1.int39

参数名:

参数值:

Figuur 13.35

The configuration parameters in Figuur 13.35 are described as follows:

- **IN**: the data for which the integral operation is to be implemented. The value of this parameter must be a data in the real-time database.
- **OUT**: the result of the integral operation. The value of this parameter must be a data in the real-time database.
- **SW**: the integral mode. When SW=0, it means to reset the integral result, namely, OUT=0. When SW=1, it means the integral becomes cumulative. When SW=2, the integral function stops and the integral result is saved. The value of this parameter can be one of the following:
  - Int constant (0, 1, or 2)
  - Data in the real-time database

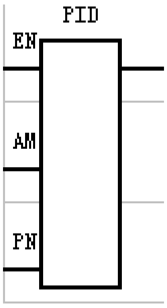
13.3.8.6 Control Algorithm

Control Algorithm function blocks are composed of two type of components, including PID Algorithm components and First-Order Model components. These components are only allowed to be placed in the last column of the grid.

13.3.8.6.1 PID Algorithm Components

PID Algorithm components are used to implement the PID regulating function. For details, see section 12.5.6 PID Function Blocks.

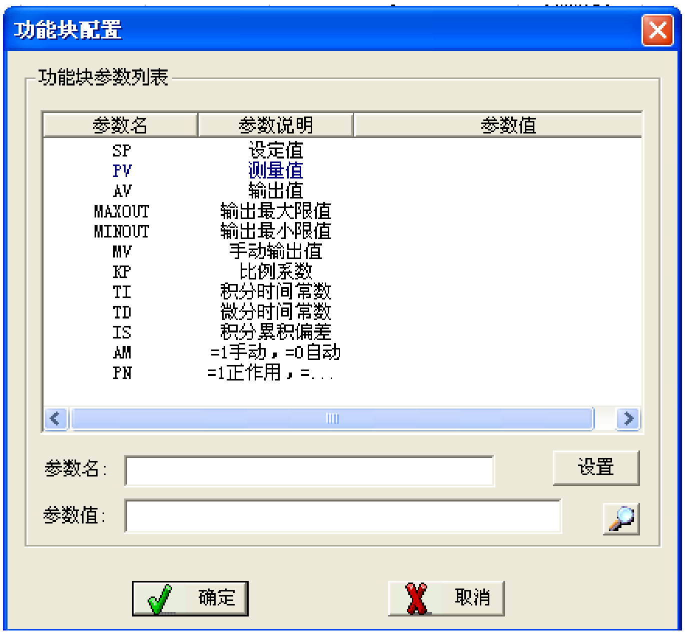
After you add a PID algorithm component, the component will be displayed in the grid as shown in Figuur 13.36.



Figuur 13.36

EN stands for Enable. Only when the value of **EN** is 1, the PID regulating algorithm will be executed; when the value of **EN** is 0, the PID regulating algorithm will not be executed.

To conFiguur the properties of the component, double-click on it, and you will see a dialog box as shown in Figuur 13.37.



参数名	参数说明	参数值
SP	设定值	
PV	测量值	
AV	输出值	
MAXOUT	输出最大限值	
MINOUT	输出最小限值	
MV	手动输出值	
KP	比例系数	
TI	积分时间常数	
TD	微分时间常数	
IS	积分累积偏差	
AM	=1手动，=0自动	
PN	=1正作用，=...	

参数名:

参数值:

Figuur 13.37

The configuration parameters in Figuur 13.37 are described as follows:

- **SP**: the setpoint. The value of this parameter must be the data in the real-time database.
- **PV**: the process variable to be measured. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database
- **AV**: the output value. The value of this parameter must be the data in the real-time database.
- **MAXOUT**: the maximum value of the output. The value of this parameter can be

one of the following:

  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database

- **MINOUT**: the minimum value of the output. The value of this parameter can be one of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- **MV**: the manual output value. The value of this parameter must be the data in the real-time database.
- **KP**: the proportional coefficient. The value of this parameter can be one of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- TI: the integral time constant. The value of this parameter can be one of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- TD: the differential time costant. The value of this parameter can be one of the following:
  - ➢ Int constant
  - ➢ Float constant
  - ➢ Data in the real-time database
- **IS**: the cumulative integral deviation. The value of this parameter must be the data in the real-time database.
- **AM**: AM=1 indicates Manual, while AM=0 indictes Automatic. The value of this parameter can either directly come from the input of the connecting line of the ladder diagram, or can be conFiguurd in the **Function Block Configuration** dialog box. If the connecting line in the ladder diagram has an input corresponding to **AM** and the **AM** parameter is conFiguurd in the **Function Block Configuration** dialog box as well, the system will take the input of the connecting line in the ladder diagram as the value of **AM**. To conFiguur the **AM** parameter in the **Function Block Configuration** dialog box, the value of the **AM** parameter can be one of the following:
  - ➢ Int constant (0 or 1)
  - ➢ Data in the real-time database
- **PN**: PN=1 indicates the positive action, while PN=0 indicates the negative action. The value of this parameter can either directly come from the input of the connecting line in the ladder diagram, or can be conFiguurd in the **Function Block Configuration** dialog box. If the connecting line in the ladder diagram has an input corresponding to **PN** and the **PN** parameter is conFiguurd in the

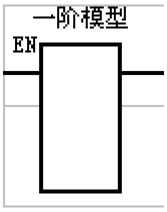
Function Block Configuration dialog box as well, the system will take the input of the connecting line in the ladder diagram as the value of **PN**. To conFiguur the **PN** parameter in the **Function Block Configuration** dialog box, the value of the **PN** parameter can be one of the following:

- Int constant (0 or 1)
- Data in the real-time database

13.3.8.6.2 First-Order Model Components

Function: to realize model objects of the first-order system.

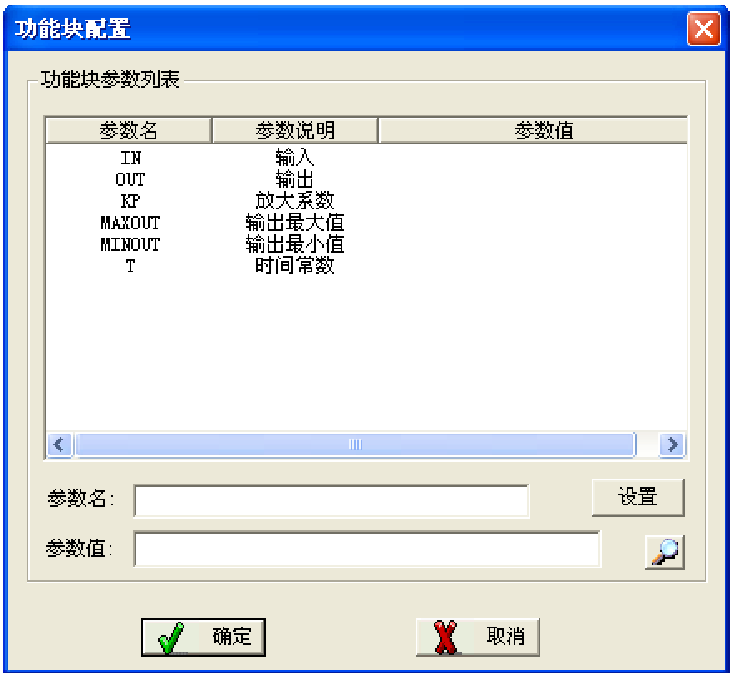
After you add an iintegral component, the component will be displayed in the grid as shown in Figuur 13.38.



Figuur 13.38

EN stands for Enable. Only when the value of **EN** is 1, the operation defined for the first-order model object will be executed; when the value of **EN** is 0, the operation will not be executed.

To conFiguur the properties of the component, double-click on it, and you will see a dialog box as shown in Figuur 13.39.



参数名	参数说明	参数值
IN	输入	
OUT	输出	
KP	放大系数	
MAXOUT	输出最大值	
MINOUT	输出最小值	
T	时间常数	

参数名：

参数值：

Figuur 13.39

The configuration parameters in Figuur 13.39 are described as follows:

- **IN**: the input. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database
- **OUT**: the output. The value of this parameter must be a data in the real-time database
- **KP**: the amplification coefficient. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database
- **MAXOUT**: the maximum value of the output. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database
- **MINOUT**: the minimum value of the output. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database
- **T**: the time constant. The value of this parameter can be one of the following:
  - Int constant
  - Float constant
  - Data in the real-time database

13.3.8.7 Timer

Timer function blocks are composed of five types of components, including Delay Timer (Decremental), Delay Timer (Incremental), Single-Shot Trigger Timer, Cycle Timer (Impulsive), and Cycle Timer (Square Wave). These components are not allowed to be placed in the last column of the grid.

13.3.8.7.1 Delay Timer (Decremental)

See the description about the dcremental timer in section 13.3.4 Timer.

13.3.8.7.2 Delay Timer (Incremental)

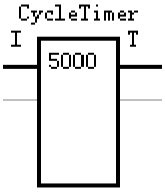
See the description about the incremental timer in section 13.3.4 Timer.

13.3.8.7.3 Single-Shot Trigger Timer

Same as section 13.3.5 Single-Shot Trigger.

13.3.8.7.4 Cycle Timer (Impulsive)

After you add a Cycle Timer (Impulsive) component, the component will be displayed in the grid as shown in Figuur 13.40.



Figuur 13.40

I stands for the input of the timer. When the value of **I** is 1, the timer will be started; when the value of **I** is 0, the timer stops running.

The difference between Cycle Timer and Delay Timer is that the execution of Delay Timer is only one time while Cycle Timer runs at interval, as described below:

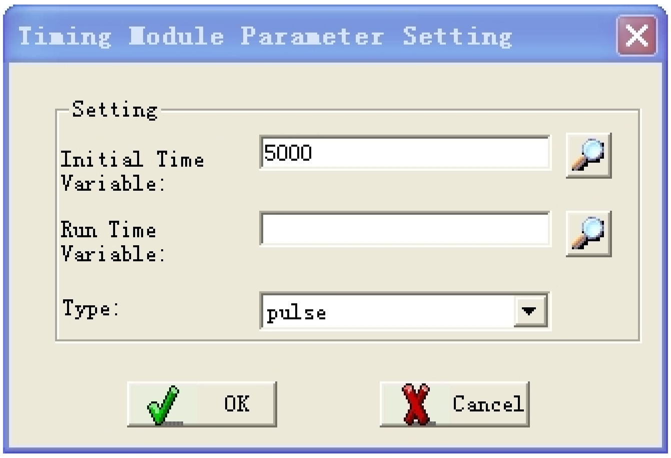
- Delay Timer: starts running when the value of **I** becomes 1; however, it stops immediately when the preset running time runs out, even though the value still remains 1. To have Delay Timer start running again, you must set the vaue of **I** to 0 first and then change it to 1.
- Cycle Timer (Impulsive): is executed repeatedly at intervals. It starts running when the value of **I** is 1. When the preset running time runs out, the system returns the output 1 for the **T** end, which triggers the timer to restart counting and resets the value of **T** to 0.

Therefore, it can be understood that the system generates an impulse for Cycle Timer (Impulsive) at a specified interval. Thus, Cycle Timer (Impulsive) is suitable for the operations to be implemented at a certain interval.

Cycle Timer (Impulsive) works as follows:

1. When the value of **I** is 1, the timer starts running. During the counting of the timer, the value of **T** remains 0. When the preset running time of the timer runs out, the value of **T** becomes 1, which triggers the timer to restart counting immediately and resets the value of **T** to 0.
2. When the value of **I** becomes 0, the timer stops running. The value of **T** will be 0 as long as the value of **I** is 0, regardless the status of the timer.

To conFiguur the properties of the component, double-click on it, and you will see a dialog box as shown in Figuur 13.41.



Timing Module Parameter Setting

Setting

Initial Time Variable: 5000

Run Time Variable:

Type: pulse

OK Cancel

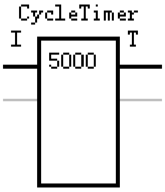
Figuur 13.41

The configuration parameters in Figuur 13.41 are described as follows:

- **Initial Time Variable**: defines when Cycle Timer starts running.
  You can either directly enter an Int constant for this parameter, or associate it to a data in the real-time database.
- **Runtime Variable**: reflects the running of the timer.
  You do not need to specify this variable if it is unnecessary to observe the running status of the timer.
- **Type**: For Cycle Timer (Impulsive), you need to set this parameter to **Impulsive**.

13.3.8.7.5 Cycle Timer (Square Wave)

After you add a Cycle Timer (Sqaure Wave) component, the component is displayed in the grid as shown in Figuur 13.42.



Figuur 13.42

I stands for the input of the timer. When the value of **I** is 1, the timer starts running; when the value of **I** is 0, the timer stops running.

Same as Cycle Timer (Impulsive), Cycle Timer (Square Wave) is also executed at intervals. However, there is difference in how they work, as described below:

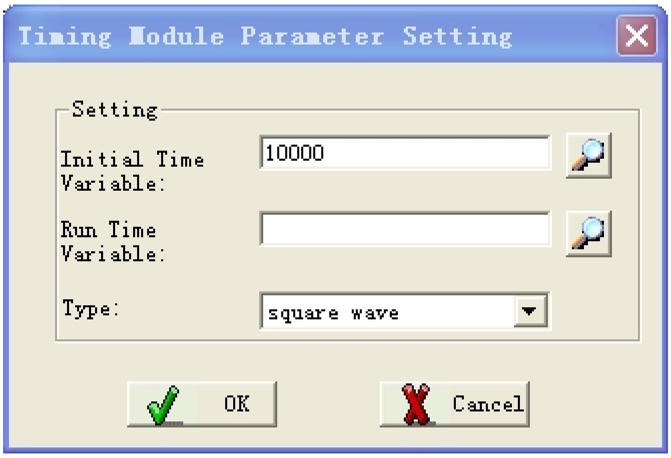
- For Cycle Timer (Impulsive), the value of **T** becomes 1 when the preset running time runs out, and then the timer restarts counting and the value of **T** is reset to 0. Considering this, the output 1 of **T** is only a transient state. It can be understood that an impulse with the value as 1 is generated for **T** at a specified interval.
- For Cycle Timer (Square Wave), the value of **T** also becomes 1 when the preset running time runs out, and then the timer restarts counting. However, the value of

T remains 1 (instead of being reset to 0 immediately) until the preset running time runs out again when its value is then reset to 0. During the next cycle of running, the value of **T** remains 0; when the preset running time runs out again, the value of **T** becomes 1. Therefore, it can be concluded that the value of **T** for Cycle Timer (Square Wave) switches repeatedly between 0 and 1 at a specified interval, which looks like a square wave.

Cycle Timer (Square Wave) works as follows:

1. At the beginning of the operation of the ladder diagram, the value of **T** is 0 since Cycle Timer (Square Wave) has not started running yet.
2. When the value of **I** is 1, the timer starts running. The value of **T** remains 0 during the first running of the timer. When the preset running time runs out, the value of **T** beomes 1, and the timer restarts counting while the value of **T** remains 1.
3. When the preset running time runs out for the second time, the value of **T** switches back to 0, and the timer restarts counting while the value of **T** remains 0.
4. As long as the value of **I** is 1, the output value of **T** switches repeatedly between 0 and 1 at the specified interval.
5. When the value of **I** is 0, the timer will stop running. The output value of **T** will be 0 as long as the value of **I** is 0, regardless the status of the timer.

To conFiguur the properties of the component, double-click on it, and you will see a dialog box as shown in Figuur 13.43.



Timing Module Parameter Setting

Setting

Initial Time Variable: 10000

Run Time Variable:

Type: square wave

Figuur 13.43

The configuration parameters in Figuur 13.43 are described as follows:

- **Initial Time Variable**: defines when the timer starts running.
  You can either enter an Int constant for this parameter, or associate it to a data in the real-time database.
- **Runtime Variable**: reflects the running of the timer.
  You do not need to specify this variable if it is unnecessary to observe the

running status of the timer.

- **Type**: For Cycle Timer (Square Wave), you need to set this parameter to **Square Wave**.

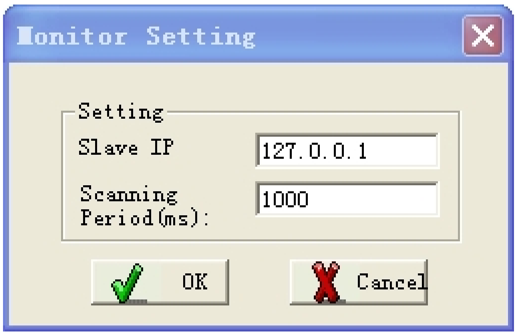
13.4 Monitoring the Ladder Diagram

EASY provides the ladder diagram monitoring function, which facilitates you to easily monitor the running status of the ladder diagram.

13.4.1 Monitoring Configuration

Before monitoring the ladder diagram, you need to conFiguur the monitoring parameters. Do as follows:

In the **Ladder Diagram Editor** window as shown in Figuur 13.1, click on the **Monitoring** menu and then the sub-menu **Monitoring Configuration**, and you will see a dialog box as shown in Figuur 13.44.



Monitor Setting

Setting

Slave IP: 127.0.0.1

Scanning Period(ms): 1000

Figuur 13.44

The configuration parameters in Figuur 13.44 are described as follows:

- **IP Adres of Slave Device**: specifies the IP adres of the computer on which the ladder diagram application is running.
- **Scanning Cycle (ms)**: defines the monitoring interval (unit: ms).

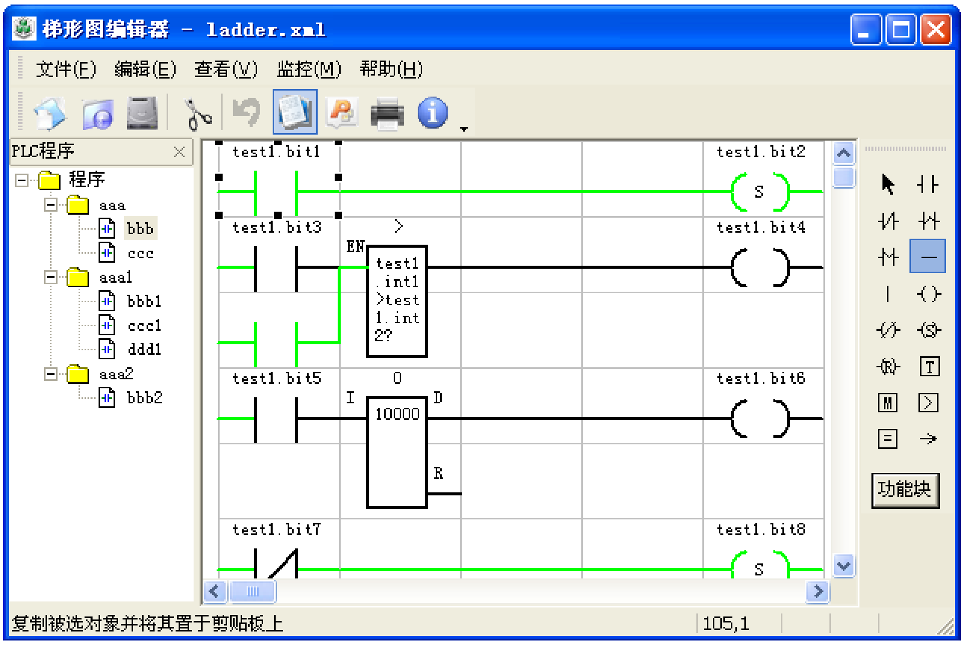
13.4.2 Starting Monitoring

After you conFiguur the monitoring parameters, click on on the **Monitoring** menu and then the sub-menu **Start Monitoring** in the **Ladder Diagram Editor** window as shown in Figuur 13.1, and the monitoring of the ladder diagram is started. Once the monitoring starts, it is not allowed to do any modification on the ladder diagram.

With the monitoring function, you can do the following operations.

13.4.2.1 Monitoring the Operation Process of the Ladder Diagram

After the monitoring function is started, the operation process of the ladder diagram will be displayed in the ladder diagram editing area on the right side of the **Ladder Diagram Editor** window, as shown in Figuur 13.45.



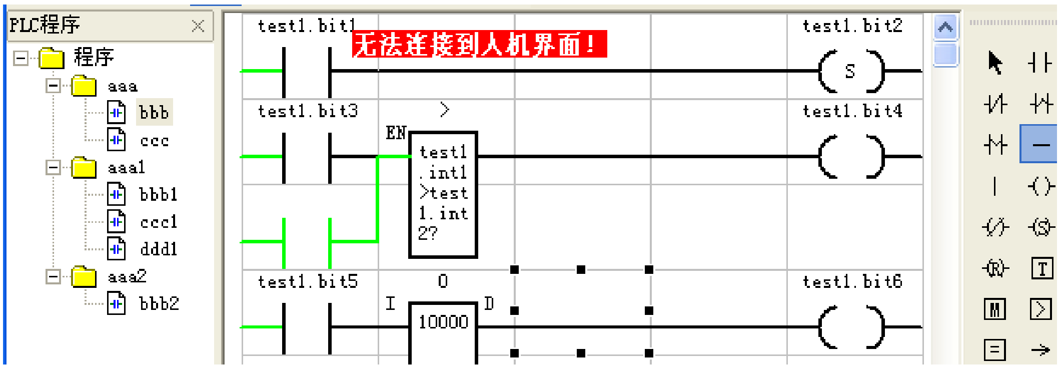
复制被选对象并将其置于剪贴板上

Figuur 13.45

As shown in the Figuur above, if the ladder diagram component is in green, it means that the current value of the component is 1; otherwise, the current value is 0. If the color of the connecting line is green, it means that the connecting line is in the Open state; otherwise, it is in the Closed state.

You need to select the program block for monitoring. To select a program block, double-click on it on the left side of the **Ladder Diagram Editor** window.

If the monitoring fails, the failure error will be displayed in the editing area on the right side of the **Ladder Diagram Editor** window. For example, if communication exception occurs, an error message as shown in Figuur 13.46 will be displayed.



无法连接到人机界面！

Figuur 13.46

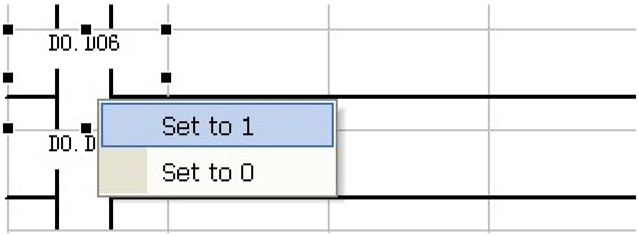
The possible error messages are listed in the table below:

Errror Message	Possible Cause
Unable to connect to the HMI!	Communication exceptions.
Communication error!	
System error!	System processing exceptions.
The program segment or block does not exist!	The project you opened is inconsistent with that currently running in the HMI.
The program block is not running!	The current value of the operational variable associated with the program segment where the program block belongs to is 0, which causes the system to read that the program block is not running.

13.4.2.2 Forcibly Changing the Digital Input Settings

After the monitoring function is started, you can forcibly control the digital input components directly in the editing area as shown in Figuur 13.45.

Right-click on a digital input component, and you will see a right-click menu as shown in Figuur 13.47.

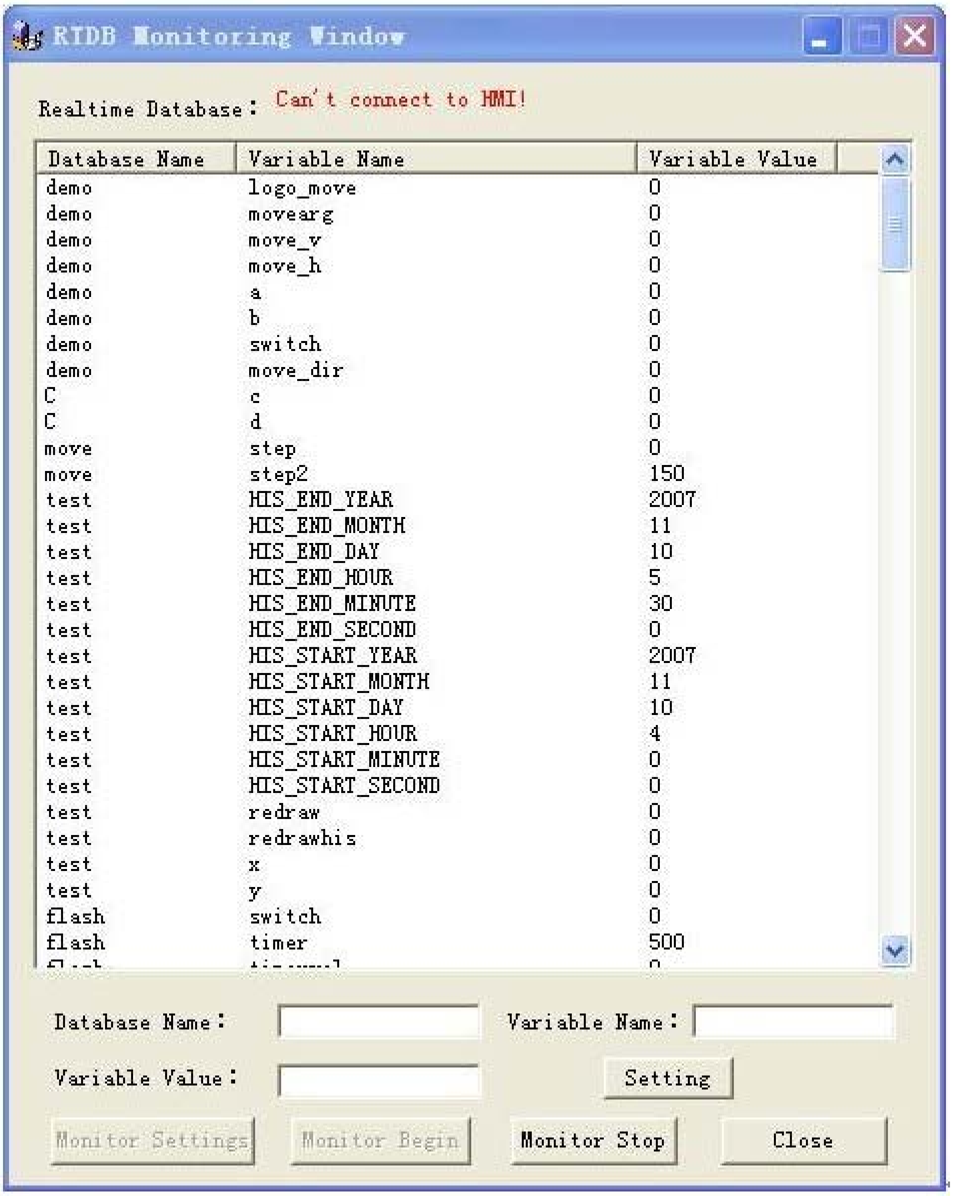


Figuur 13.47

To forcibly change the value of the digital input to 1, click on **Set as 1** on the right-click menu; to forcibly change the input value to 0, click on **Set as 0**.

13.4.2.3 Monitoring the Data in the Real-Time Database

In the **Ladder Diagram Editor** window as shown in Figuur 13.1, click on the **Monitoring** menu and then the sub-menu **Monitoring Real-Time Data**, and the **EASY Real-Time Data Monitoring** tool will be started, as shown in Figuur 13.48.



RTDB Monitoring Window

Realtime Database: Can't connect to HMI!

Database Name	Variable Name	Variable Value
demo	logo_move	0
demo	movearg	0
demo	move_v	0
demo	move_h	0
demo	a	0
demo	b	0
demo	switch	0
demo	move_dir	0
C	c	0
C	d	0
move	step	0
move	step2	150
test	HIS_END_YEAR	2007
test	HIS_END_MONTH	11
test	HIS_END_DAY	10
test	HIS_END_HOUR	5
test	HIS_END_MINUTE	30
test	HIS_END_SECOND	0
test	HIS_START_YEAR	2007
test	HIS_START_MONTH	11
test	HIS_START_DAY	10
test	HIS_START_HOUR	4
test	HIS_START_MINUTE	0
test	HIS_START_SECOND	0
test	redraw	0
test	redrawhis	0
test	x	0
test	y	0
flash	switch	0
flash	timer	500

Database Name:

Variable Name:

Variable Value:

Figuur 13.48

With this tool, you can monitor or modify the data in the real-time database.

13.4.3 Stopping Monitoring

In the **Ladder Diagram Editor** window as shown in Figuur 13.1, click on **Monitoring** and then **Stop Monitoring**, and the monitoring function will be stopped.

13.5 System Variables for the Ladder Diagram

Database Name	Variable Name	Data Type	Default Value	Description
system	PlcEnable	bit	1	• When the variable value is

Database Name	Variable Name	Data Type	Default Value	Description
				1, the ladder diagram function is enabled. • When the variable value is 0, the ladder diagram function is disabled.
	PlcCycleTime	ulong	100	This variable defines the cycle for executing the ladder diagram (unit: ms).
	PlcHeartbeat	bit		This variable defines the heartbeat of the ladder diagram during the operation. As long as the ladder diagram is running, the value of this variable will switch repeatedly between 0 and 1. The value switch happens every time the ladder diagram is executed.

Chapter 14 Expansion Module Programming

14.1 Overview

EASY HMI allows you to add expansion modules. The expansion modules can be added as dynamic databases into the HMI system and run as threads.

The expansion modules can be used for implementing special control algorithms or dedicated communication protocols.

The expansion modules must be programmed in the standard C programming language.

14.2 Module Export Functions

All expansion modules must be able to support two functions (external and non-static functions) which are described in the rest of this section.

14.2.1 module_init

Original Function: int **module_init**(**char** **params*)

Function Description: To initialize the module. EASY calls this function automatically at the system startup. You can program the module initialization code with this function.

Return Values: 0 Failed
1 Successful

Parameters: *params*: Initialization parameters.

14.2.2 module_exit

Original Function: void **module_exit**()

Function Description: To exit the module. EASY calls this function automatically at the system shutdown. You can program the module exit code with this function.

Return Values: None.

Parameters: *None.*

Note: The above two functions must be exported from the dynamic database of the module.

14.3 User IO Driver Module

The expansion module can also be used to realize specific IO driver. The driver will be registered during the module initialization, so that the system can initialize and then execute the driver automatically during the system operation.

The data architecture of the IO driver module is as follows:

```
typedef struct _ctrl_io_driver_t
  {
      struct _ctrl_io_driver_t *next;
      char *name;
      int (*init)();
      int (*run)();
      void (*release)();
  }ctrl_io_driver_t;
```

In the data architecture above, **name** refers to the name of the driver, the **init** function pointer refers to the initial code of the driver, the **run** function pointer refers to the run function of the driver, the **release** function pointer refers to the release code of the driver, and the **next** function pointer is for system internal use only – for linking multiple drivers. Among all these functions, the **run** function runs in a separate thread, while the other functions run in the main thread of the HMI.

14.3.1 control_io_register_dirver

Original Function: int **control_io_register_driver**(**ctrl_io_driver_t** **io_driver*)

Function Description: To call the function **control_io_register_driver** in the module initialization function **module_init** to register the driver.

Return Values: 0 Failed
1 Successful

Parameters: *io_driver*: Data architecture of the IO driver module.

Example:

```
static ctrl_io_driver_t echodemo_drive={NULL, "echodemo", echodemo_init,
        echodemo_run, echodemo_release};
int module_init(char *params)
```

```
{
      control_io_register_driver(&echodemo_drive);
      return 1;
 }
```

14.4 Real-Time Database Read/Write Functions

14.4.1 rtdb_set_data_value_by_name

Original Function: int **rtdb_set_data_value_by_name**(**char** ***dbname*,**char** ***dataname*,**void** ***data_value*)

Function Description: You can use this function in the expansion module to write the data in the real-time database. This function sets the value of the data in the real-time database.

Return Values: 0 Failed
1 Successful

Parameters: *dbname*: Name of a database.
dataname: Name of a data.
data_value: Value of a data. The value of a data varies according to the data type. For example, the bit data has only 1 byte, the long data has 4 bytes, and the length of the string data is user-defined.

Example:

```
float value=1.0;
rtdb_set_data_value_by_name(“test”,”Ldata”,&value);
```

The above function sets the value of the data **Ldata** in the database **test** to 1.0.

14.4.2 rtdb_get_data_value_by_name

Original Function: int **rtdb_get_data_value_by_name**(**char** ***dbname*,**char** ***dataname*,**void** ***data_value*)

Function Description: You can use this function in the expansion module to read the data in the real-time database. This function obtains the value of the data from the real-time database.

Return Values: 0 Failed

1 Successful

Parameters: *dbname*: Name of a database.

dataname: Name of a data.

data_value: Value of a data. The parameter **data_value** requires you to assign the space in advance. For example, you need to assign 1 byte of space for the bit data and 4 bytes of space for the long data.

Example:

float value;

rtdb_get_data_value_by_name(“test”,”Ldata”,&value);

The above function obtains the current value of the data **Ldata** from the database **test** and saves the value to the variable **value**.

14.5 Serial Port Communication Functions

EASY HMI provides some standard serial port communication functions, which facilitates the user-defined serial port communication programming.

14.5.1 serial_open

Original Function: int **serial_open**(**const char ***device*,**int** *baud*,**int** *parity*,**int** *data_bits*,**int** *stop_bits*,**int** *timeout*)

Function Description: To open a serial port.

Return Values: -1 Failed

Other value Serial port handle

Parameters: *device*: Serial port name, for example, COM1, COM2, or COM3.

baud: Baud rate of the serial port. At present, the supported baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200.

parity: Parity check mode, including No Parity Check (SERIAL_PARITY_NO), Odd Parity Check (SERIAL_PARITY_ODD), and Even Parity Check (SERIAL_PARITY_EVENT).

data_bits: Number of data bits, including 5, 6, 7, and 8.

stop_bits: Number of stop bits, including 1 and 2.

timeout: Duration of communication timeout (unit: ms).

Example: serial_open("COM1", 9600, SERIAL_PARITY_NO, 8, 1, 100)

14.5.2 serial_close

Original Function: int **serial_close**(**int** *serial_id*)

Function Description: To close a serial port.

Return Values: 0 Failed

1 Successful

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

Example: serial_close(1)

14.5.3 serial_flush

Original Function: int **serial_flush**(**int** *serial_id*, **int** *flag*)

Function Description: To clear the buffering data of a serial port.

Return Values: 0 Failed

1 Successful

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

flag: Flush flag.

SERIAL_FLUSH_TX: clears the currently unsent data.

SERIAL_FLUSH_RX: clears the data in the receiving buffer.

SERIAL_FLUSH_TX|SERIAL_FLUSH_RX: clears the data in both the sending and receiving buffers.

Example: serial_flush(1, SERIAL_FLUSH_TX|SERIAL_FLUSH_RX)

14.5.4 serial_write

Original Function: int **serial_write**(**int** *serial_id*, **char ****buf*, **size_t** *size*)

Function Description: To send data to a serial port.

Return Values: -1 Failed

Other value Actual length of the data sent

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

buf: Buffer for keeping the data to be sent.

size: Length of the data to be sent (unit: byte).

Example: serial_write(1,buf, sizeof(buf))

14.5.5 serial_read

Original Function: int **serial_read**(**int** *serial_id*, **char ****buf*, **size_t** *size*)

Function Description: To receive data through a serial port.

Return Values: -1 Failed

Other value   Actual length of the data received

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

buf: Buffer for keeping the received data.

size: Size of the receiving buffer (unit: byte).

Example: serial_read(1,buf, sizeof(buf))

14.5.6 serial_poll

Original Function: int **serial_poll**(**int** *serial_id*, **int** *timeout*)

Function Description: To check whether there is data to read at a serial port.

Return Values: -1   Error

0   Timeout, which meas no data is received during the timeout period

1   Data available to be read at the serial port

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

timeout: For inquiring the timeout duration (unit: ms)

Example: serial_poll(1,100)

14.6 Other Functions

Expansion modules also support the functions programmed in the standard C programming language and the internal functions provided in EASY HMI; for example, the function **sys_sleep** (see EASY HMI User Manual for more details).

14.7 Expansion Module Commissioning

EASY HMI provides a commissioning solution which is based on Microsoft Visual C++ 6.0. For details, see the example project **echodemo**.

In the **echodemo** project file, the **echodemo** directory covers for expansion module projects, the **echodemohost** directory for serial port communication applications on the host, and the **project** directory for EASY HMI projects which are for the demo of the application of expansion modules.

The main function of the **Echodemo** project is to send a string to the HMI. The host-side application will receive this string and then send it back. The **Echodemo** expansion modules can call all the functions for serial port communication to realize user-defined communication scenarios. See the source codes for the working principles of expansion modules.

The main procedure for compiling the VC project for expansion modules is listed as follows:

1) Create a VC standard dynamic library DLL project (MFC support not required).
2) Add the source files for expansion modules (such as **echodemo.c**), the header file **EASY.h** and the DEF file (such as **echodemo.def**). The DEF file needs both export functions **module_init** and **module_exit**.
3) Add all applications and files of the EASY runtime library. For example, add the applications and files **lcrun.exe**, **lcrun.dll**, **lcrun.lib**, **iconv.dll**, **libiconv-2.dll**, **libxml2.dll**, **pic.dll**, and the files in the directory **conffile** to the project directory.
4) Set project parameters, and add the reference of the library **lcrun.lib** in the **Link** library.
5) Set project commissioning parameters. Set **lcrun.exe** as an executable file under **Debug**, and set the program parameter as a project in EASY HMI (such as **..\project**).
6) Set the project Post-build parameter to copy the module file to the HMI project directory; for example, **copy .\debug\echodemo.dll ..\project\echodemo.dll**.
7) Compile and then commission the project.

14.8 Expansion Module Management

After you complete compiling an expansion module, you need to integrate the module into the developed project, so that the module can be downloaded or run together with the project. This can be done through the EASY **Project Manager** window. The detailed procedure is listed as follows:

1) ConFiguur the expansion module in the **Project Manager** window.
2) Compile the expansion module.
3) Download the expansion module to the HMI.

The rest of this section describes this procedure in details.

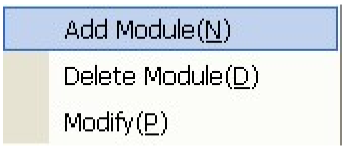
14.8.1 Configuring an Expansion Module

You can define more than one expansion module in a project. These expansion modules can be conFiguurd and managed through **Expansion Module Configuration** in the **Project Manager** window.

14.8.1.1 Adding a Module

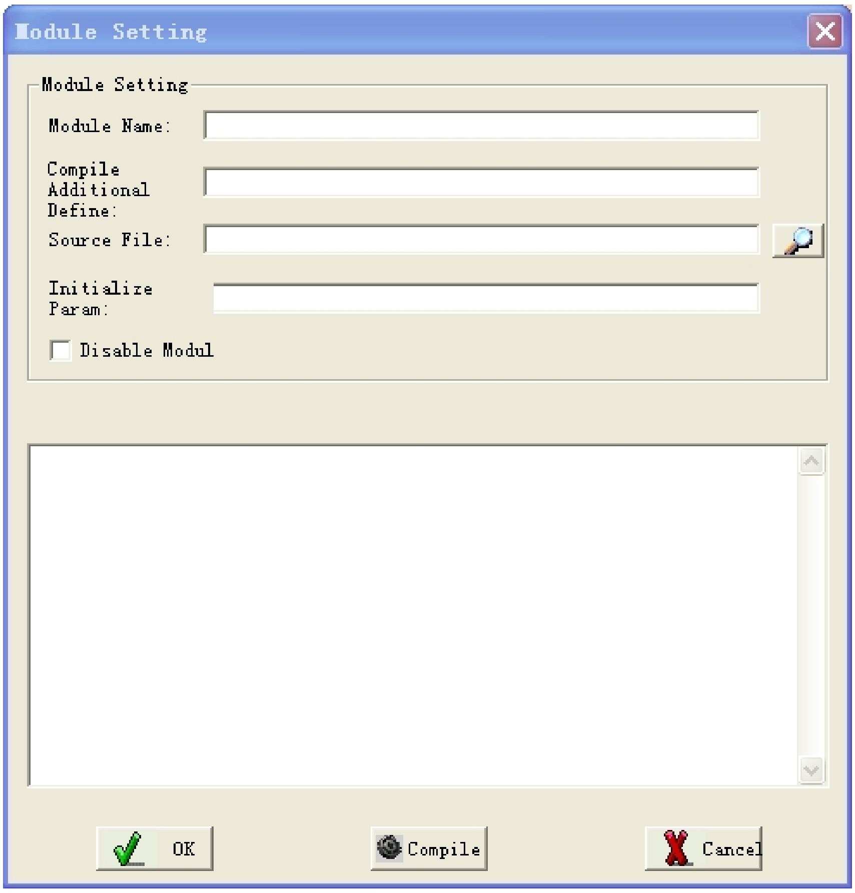
To add an expansion module, do as follows:

1) In the navigation tree on the left side of the **Project Manager** window, select **Expansion Module Configuration**. Right-click in the blank area on the right side of the window, and you will see a right-click menu as shown in Figuur 14.1.



Figuur 14.1

2) Select **Add a Module**, and you will see the dialog box as shown in Figuur 14.2.



Figuur 14.2

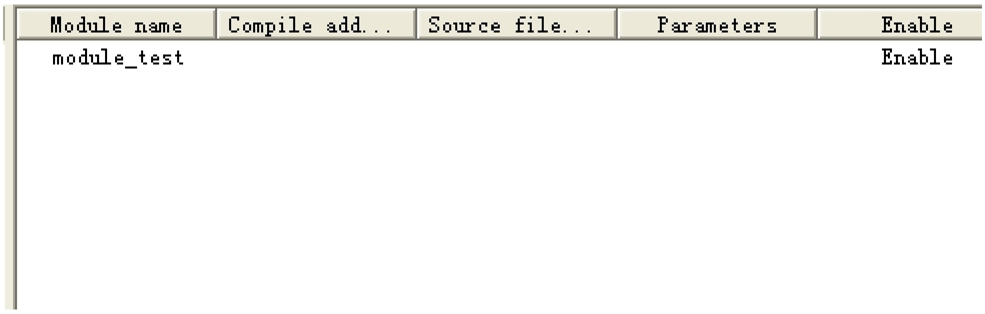
The configuration parameters in Figuur 14.2 are described as follows:

- **Module Name**: The name of the module. You can choose any name for an expansion module. However, the expansion modules in the same project cannot have the same name.
- **Additional Compiling Definition**: corresponds to the compiling option, such as **-D_DEBUG –D_MYDEF –Ic:\myinclude** in the C language.
- **Source File**: specifies the source file(s) to be compiled.
- **Initialization Parameter**: defines the initialization parameter for the expansion module. The value of this parameter is a user-defined string, which will be used as the value of **param** in the module initialization function **module_init** at the module initialization.

- **Disable the module**: If this option is checked, the module will be disabled, which means that the system will not load the module and that the module will not be executed.

3) Click on **OK** after the module configuration is complete.

   The conFiguurd module will be displayed in the module list on the right side of the **Project Manager** window, as shown in Figuur 14.3.



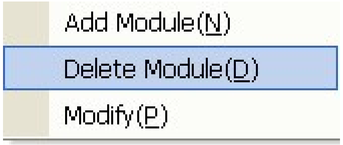
Module name	Compile add...	Source file...	Parameters	Enable
module_test				Enable

Figuur 14.3

14.8.1.2 Deleting a Module

To delete a module, do as follows:

1) In the navigation tree on the left side of the **Project Manager** window, select **Expansion Module Configuration**.
2) On the right side of the **Project Manager** window, right-click on the expansion module you want to delete, and you will see a right-click menu as shown in Figuur 14.4.



Figuur 14.4

3) Select **Delete a Module**.

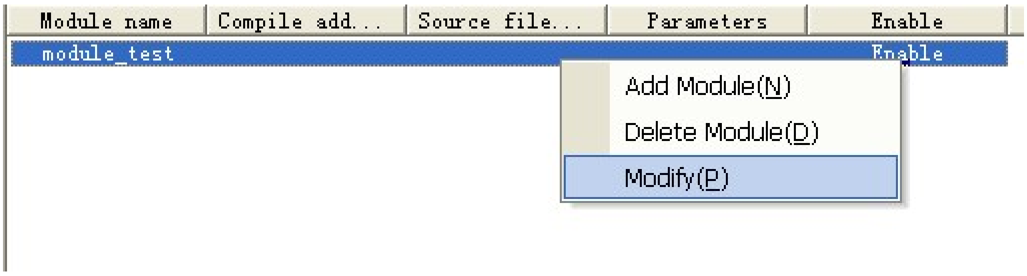
   The selected module will be deleted.

14.8.1.3 Modifying Module Cofiguration

To modify the configuration of an expansion module, do as follows:

1) In the navigation tree on the left side of the **Project Manager** window, select **Expansion Module Configuration**.
2) On the right side of the **Project Manager** window, right-click on the expansion module for which you want to modify the configuration, and you will see a

right-click menu as shown in Figuur 14.5.



Figuur 14.5

3) Select **Modify**.

   You can modify the module configuration according to your needs.

14.8.2 Compiling an Expansion Module

The conFiguurd expansion modules can be downloaded to the HMI only after they are compiled.

To compile an expansion module, do as follows:

1) In the navigation tree on the left side of the **Project Manager** window, select **Expansion Module Configuration**.
2) On the right side of the **Project Manager** window, double-click on the expansion module you want to compile, and you will see a dialog box as shown in Figuur 14.2.
3) Click on the **Compile** button to start compiling the module.

14.8.3 Downloading an Expansion Module

Once compiled successfully, the expansion modules will be copied automatically to the HMI project directory. They are downloaded together with the project to the HMI; you do not need to download the module individually.

Chapter 15 Access Management

15.1   Overview

In practical operation, some interfaces can only be accessed by the users with higher privileges considering security reasons. The role of operator does not have the privilege to access such interfaces. Regarding this, EASY provides the access management function.

The access management function of EASY is actually window-based, which means it allows configuring security privileges for each window.

The access management function of EASY involves the following concepts:

1. **Security Level**:

In the EASY system, different windows have different privileges. Actually, you can define the security level from each window.

At present, EASY supports 10 security levels from level 1 to level 10, among which level 1 is the lowest while level 10 is the highest. During the configuration, you can set a password for each level, and the passwords for the ten levels are saved individually in the system variables from **$hmi_system_set.security_level1_pass** to **$hmi_system_set.security_level10_pass**.

2. **Security Level of Window**:

During the configuration, you can set a security level for each window. This security level actually defines the access privilege of the window.

3. **Current Security Level of System**:

The system is always running under certain security level. The current security level of the system is saved in to the system variable **$hmi_system_set.cur_security_level**. To change the current security level of the system, you can change the value of this variable in the script.

When you access a window, the system will automatically compare whether the current security level of the system is Equal To Or Greater Than the security level of the window: If so, the system has higher security level compared to the window, and thus you can access the window; otherwise, the system is at a lower security level, and thus you need to pass the access authentication to access the window.

To implement the access management function, follow the steps below:

1. Set the password for each security level.
2. Set the window properties related to access management.
3. Change the current security level of the system.

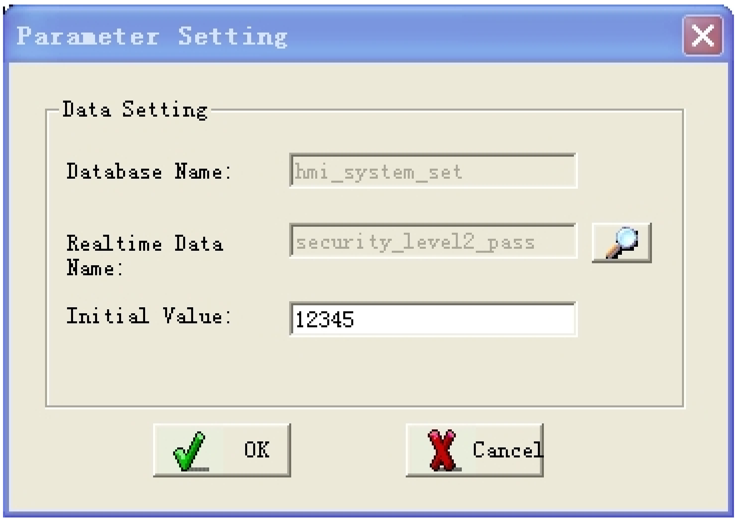
See the following sections for details.

15.2 Setting Security Level Password

The passwords for the ten security levels from level 1 to level 10 are saved individually to the system variables from **$hmi_system_set.security_level1_pass** to **$hmi_system_set.security_level10_pass**. You can modify the password of each security level according to your needs. If you leave the password of a security level to blank, it means that the specific security level is not in use. The security level password can be as long as 20 characters (including the end character **\0** of a character string).

You can set the security level in the following two ways:

- Change the value of the security level in the script during the configuration.
- Set the initial value for each password during the parameter configuration as shown in Figuur 15.1.



Parameter Setting

Data Setting

Database Name: hmi_system_set

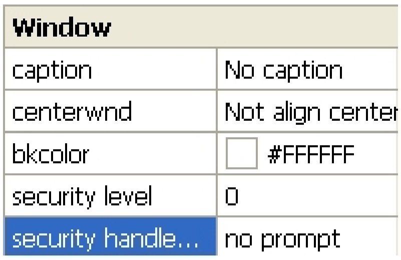
Realtime Data Name: security_level2_pass

Initial Value: 12345

Figuur 15.1

15.3 Setting Window Properties Related to Access Management

There are two window properties which are related to access management, as shown in Figuur 15.2:



Window	
caption	No caption
centerwnd	Not align center
bkcolor	□ #FFFFFF
security level	0
security handle...	no prompt

Figuur 15.2

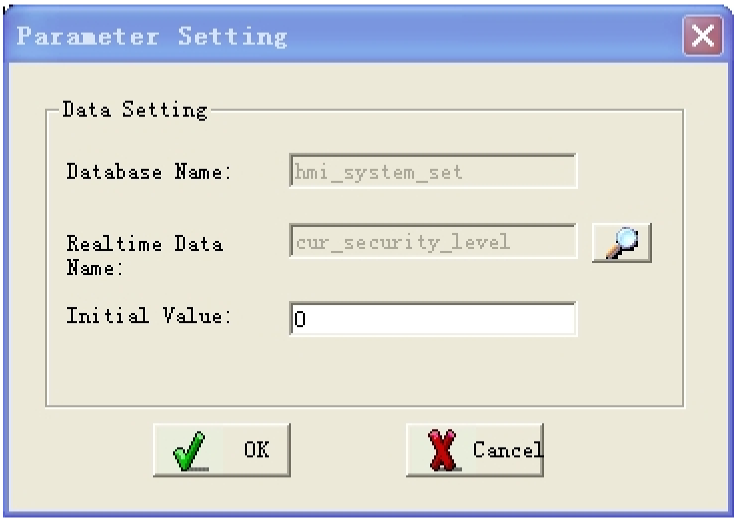
- **Security Level**: defines the security level of the window. The value range is from 0 to 10, among which the values from 1 to 10 correspond to security level 1 to security level 10 individually. If you set the security level of the window to 0, it means that you can access the window without any security authentication. Because the value 0 is the lowest security level you can set for the window. In this case, the current security level of the system can always be higher than that of the window, and thus you can directly access the window.
- **Security Processing Mode**: As stated above, when the current security level of the system is lower than that of the window, you are not allowed to access the window, and thus the system will process these kind of situations according to setting of the **Security Processing Mode** property of the window. The following settings are available for this property:
  1) **No Prompt**: You are not allowed to access this window, and no error message will be prompted.
  2) **Prompt No Access**: You are not allowed to access the window, and a dialog box will be displayed prompting that you do not have enough access to access the window.
  3) **Prompt for Password**: A window will pop out prompting for a password. The system will verify the entered password for authentication. You can access the window only after the entered password passes the authentication. The password authentication goes as follows: The system compares the entered password with the passwords for each of the security levels starting from the current security level of the window. The authentication is successful only when the entered password is consistent with that of one of the security levels among then 10; otherwise, the authentication fails.

15.4 Setting Current System Security Level

The current security level of the system is saved to the system variable **$hmi_system_set.cur_security_level**. The value range of this variable is 0 – 10, with the default value as 0.

You can change the current security level of the system in the following two ways:

- Change the value of the current security level in the script during the configuration.
- Set the initial value for this variable during the parameter configuration, as shown in Figuur 15.3.



Parameter Setting

Data Setting

Database Name: hmi_system_set

Realtime Data Name: cur_security_level

Initial Value: 0

Figuur 15.3

Chapter 16 Printing

16.1 Overview

EASY provides the window-based printing function, which allows you to print any area in the current window.

To print from the window, you need to do the following:

1. Set printing parameters.
2. Call the printing function to realize the printing function.

See the following sections for more details.

16.2 Setting Printing Parameters

At present, EASY supports the following printing parameters:

- Printer Type
- Printing Resolution
- Paper Type
- Top Margin
- Left Margin
- Printing Scaling

Each of the above printing parameters is associated with a system variable. Before you start printing, you must set the printing parameters appropriately according to your actual needs.

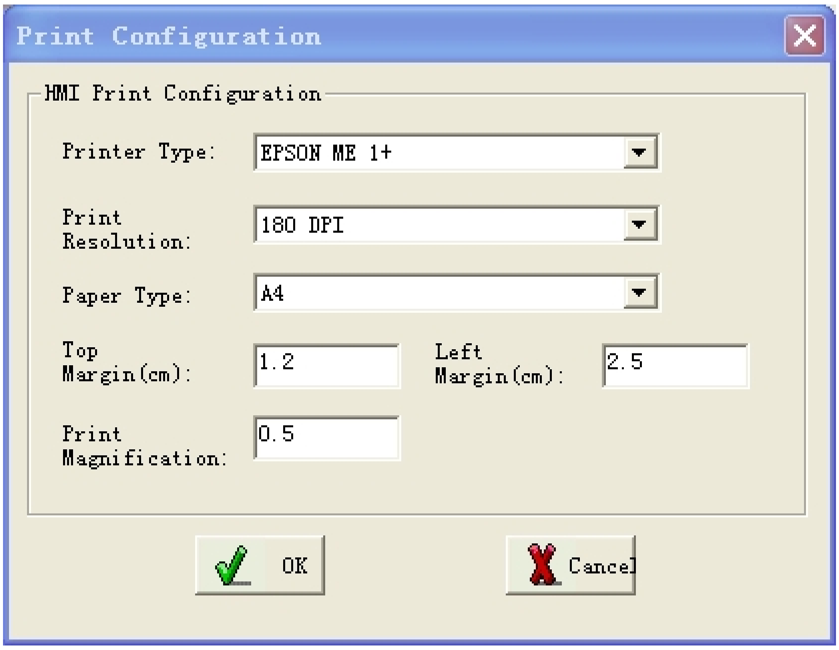
The printing parameters are described in details in the table below.

Parameter Name	**System Variable**	**Description**
Printer Type	printer.printer_name	Indicates the model of the printer.
Printing Resolution	printer.resolution	Defines the printing resolution (measured in DPI).
Paper Type	printer.paper	Defines the paper type, such as A4.
Left Margin	printer.left_margin	Defines the left margin of the printing (unit: cm).
Top Margin	printer.top_margin	Defines the top margin of the printing (unit: cm).

Printing Scaling	printer.scaling	Defines the printing scaling value.

You can set these printing parameters in the following three methods:

Method 1: In the **Project Manager** window, select **Configuration** and then **Printing Setting**, and you will see the dialog box as shown in Figuur 16.1.


Print Configuration

HMI Print Configuration

Printer Type: EPSON ME 1+

Print Resolution: 180 DPI

Paper Type: A4

Top Margin(cm): 1.2

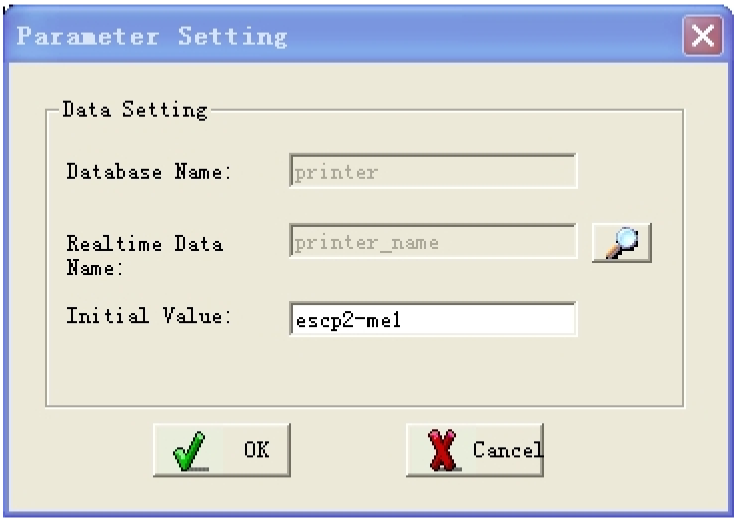
Left Margin(cm): 2.5

Print Magnification: 0.5

Figuur 16.1

Select the printing parameters according to your actual needs, and then click on **OK**.

Method 2: You can set an initial value for each printing parameter in the Parameter Configuration window, as shown in Figuur 16.2. (Here, it takes the **Printer Type** parameter for example.)


Parameter Setting

Data Setting

Database Name: printer

Realtime Data Name: printer_name

Initial Value: escp2-me1

Figuur 16.2

Method 3: You can set or change the value of the printing parameters in the script during the configuration.

16.3 Calling Printing Function

You can call the system function **print_window** for printing from the window. At present, this function allows only printing from the currently displayed window.

This function is described in details below:

Original Function: int **print_window**(**char** ***window_name*, **int** *x1*, **int** *y1*, **int** *x2*, **int** *y2*)

Function Description: To print the selected area from the currently displayed window.

Return Values: 1 Successful

Other value Failed

Parameters: *window_name*: Name of the window to be printed.

x1: X-axis value of the left top corner of the area to be printed.

y1: Y-axis value of the left top corner of the area to be printed.

x2: X-axis value of the right bottom corner of the area to be printed.

y2: Y-axis value of the right bottom corner of the area to be printed.

Considering that printing might take some time, the system function **print_window** will return the value immediately rather than waiting for the printing to be complete. And then the printing operation will be processed at the background. You can check the printing status by inquiring the system variable **printer.status**. The value of **printer.status** can be one of the following:

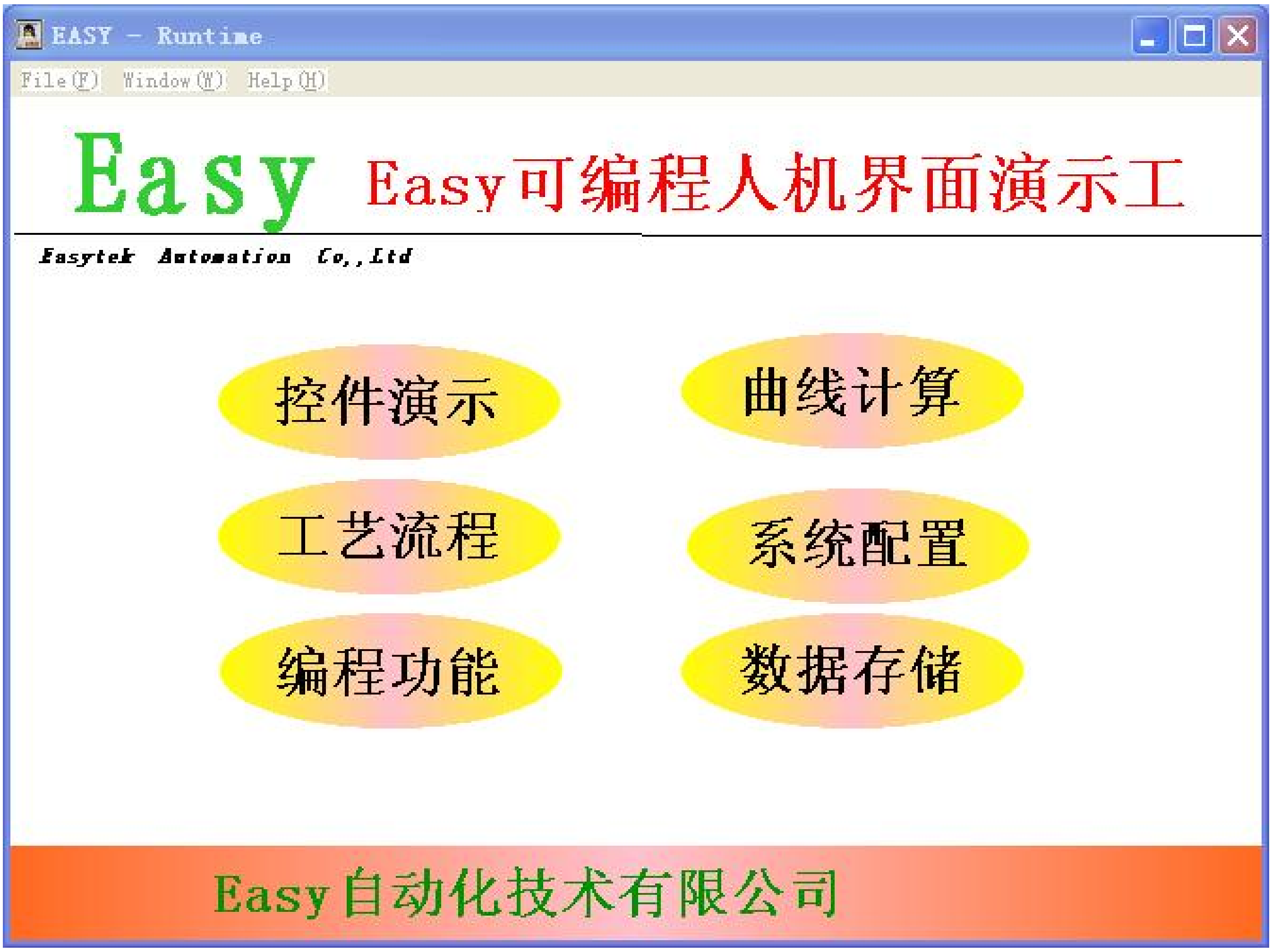
- 0 -- Idle
- 1 -- Printing
- 2 -- Printer not connected
- 3 -- Printer type not supported
- 4 -- Printing error

Chapter 17 Other Commonly-Used Functions

17.1 Offline Simulation

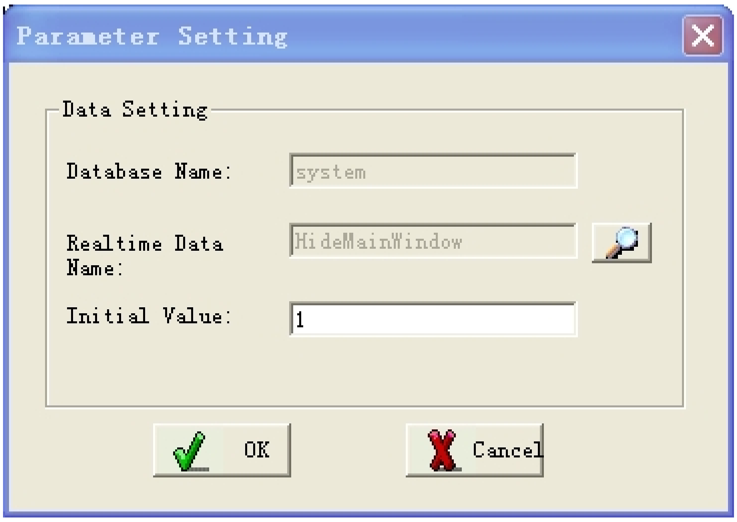
The Offline Simulation function allows you to simulate the project operation directly on the PC, which saves you the trouble of downloading the project to the HMI. This function greatly facilitates the programming and the commissioning. At present, EASY supports simulating almost all the functions on the PC, such as interface configuration, communication, and soft PLC.

To realize the offline simulation for a project after it is compiled and saved, select the menu **Tools** and then the sub-menu **Offline Simulation** in the **Project Manager** window. Alternatively, you can click on the button in the toolbar or press on the shortcut key F5. You will see the offline simulation window as shown in Figuur 17.1.



Figuur 17.1

By default, a set of menus will be displayed in the offline simulation window, which allows you to implement the project commissioning easily. If you do not want these menus to be displayed, you need to add the system parameter **$system.HideMainWindow** during the parameter configuration, as shown in Figuur 17.2.



Database Name: system

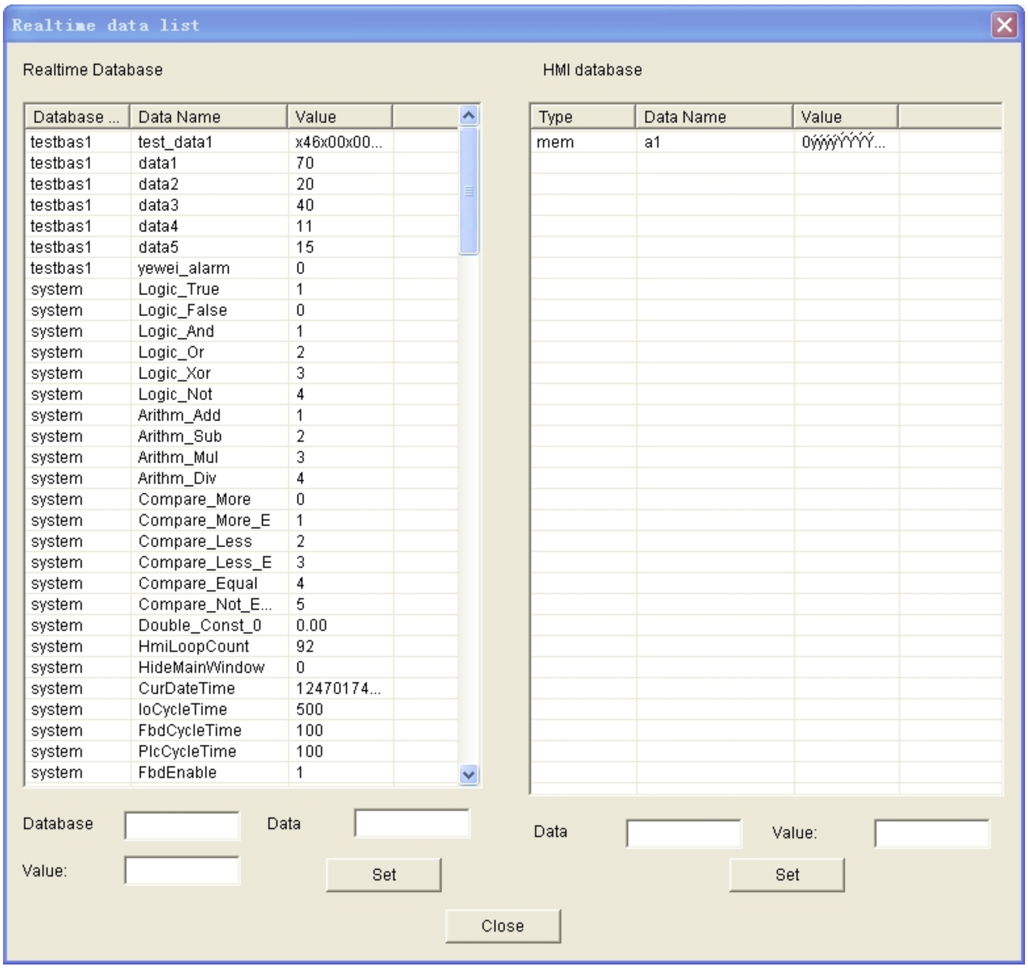
Realtime Data Name: HideMainWindow

Initial Value: 1

Figuur 17.2

If you set the value of the parameter **$system.HideMainWindow** to 1, the menus will not be displayed; if you set it to 0 or leave it blank, then the menus will be displayed.

During the offline simulation mode, click on the menu **Window** and then the sub-menu **Real-Time Data Display**, and you can see the **Real-Time Data List** dialog box, as shown in Figuur 17.3. In this dialog box, you can monitor the data in both the real-time database and the interface database.



Realtime Database

Database ...	Data Name	Value	
testbas1	test_data1	x46x00x00...	
testbas1	data1	70	
testbas1	data2	20	
testbas1	data3	40	
testbas1	data4	11	
testbas1	data5	15	
testbas1	yewei_alarm	0	
system	Logic_True	1	
system	Logic_False	0	
system	Logic_And	1	
system	Logic_Or	2	
system	Logic_Xor	3	
system	Logic_Not	4	
system	Arithm_Add	1	
system	Arithm_Sub	2	
system	Arithm_Mul	3	
system	Arithm_Div	4	
system	Compare_More	0	
system	Compare_More_E	1	
system	Compare_Less	2	
system	Compare_Less_E	3	
system	Compare_Equal	4	
system	Compare_Not_E...	5	
system	Double_Const_0	0.00	
system	HmiLoopCount	92	
system	HideMainWindow	0	
system	CurDateTime	12470174...	
system	IoCycleTime	500	
system	FbdCycleTime	100	
system	PlcCycleTime	100	
system	FbdEnable	1	

HMI database

Type	Data Name	Value	
mem	a1	0ýÿÿÿÝÝÝÝ...	

Database Data

Value: Set

Data Value: Set

Close

Figuur 17.3

The **Real-Time Data List** dialog box is composed of two panes, left and right. The left side pane lists all the data in the current real-time database (including the user-defined data and the internal system data), and the rightside pane lists all the data in the interface database (including the user-defined data and the system internal data). In this dialog box, you can not only view the current value of the data, but also change the value forcibly. Do it as follows:

Double-click on the data you want to change in the data list, and you can see the corresponding database name, variable name, and variable value displayed in the editing area at the bottom of the window. You can enter the database name and variable name corresponding to the selected data, and then enter the data value to be changed for

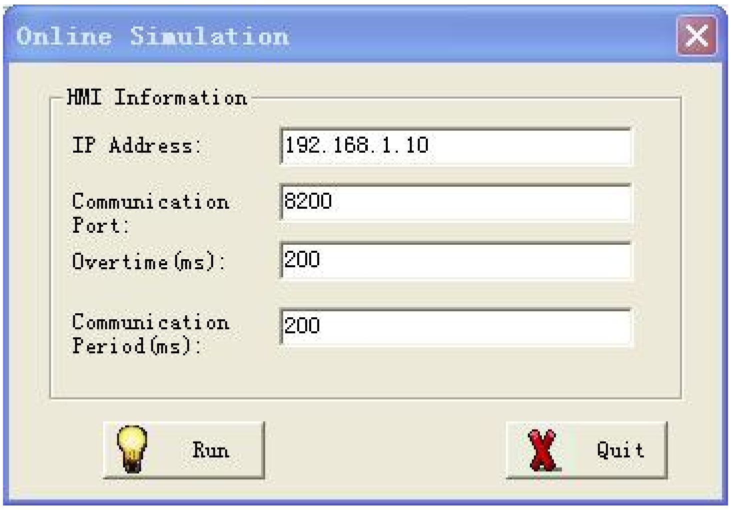
Variable Value, and then click on the **Settings** button.

Besides the **Real-Time Display** sub-menu, the **Window** menu also includes the sub-menus which stand for all the configuration interfaces of the current project. The names of the menus correspond to the names defined for **Window Title** of the interfaces in **Project Manager**. In other words, by selecting a sub-menu, you can go to the corresponding interface directly.

17.2 Online Simulation

Different from the offline simulation, the online simulation requires downloading the application to the HMI. Besides, the PC must communicate well with the HMI for the data required for the online simulation to be transferred from the HMI to the PC, so that the online simulation can be realized on the PC.

To realize the online simulation, select the **Tools** menu and then the **Online Simulation** sub-menu or click on the button in the toolbar, and you will see the **Online Simulation** dialog box as shown in Figuur 17.4.



Online Simulation

HMI Information

IP Address: 192.168.1.10

Communication Port: 8200

Overtime(ms): 200

Communication Period(ms): 200

Figuur 17.4

The configuration parameters in Figuur 17.4 are described as follows:

- **IP Adres**: specifies the IP adres of the HMI.
- **Communication Port**: specifies the number of the interception port of the HMI; the system default port number is 8200.
- **Timeout Time**: defines the period of timeout which occurs during the communication between the PC and the HMI (unit: ms).
- **Communication Cycle**: defines the interval for the PC to obtain data from the HMI (unit: ms).

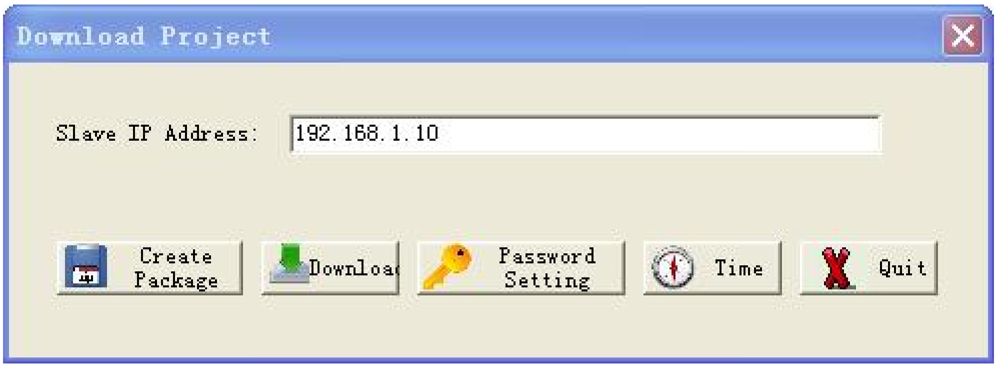
After you conFiguur all the parameters, click on **Run**, and you can start the online simulation of a project. Click on the **Windo**w menu and then the **Real-Time Data Display** sub-menu to monitor the data in both the real-time database and the interface database. For more details, see section 17.1 Offline Simulation.

17.3 Setting Project Password

You can set a password for each project. When uploading a project, you will be prompted to enter the password. You are allowed to upload the project only when you enter the correct password.

To set the project password, do as follows:

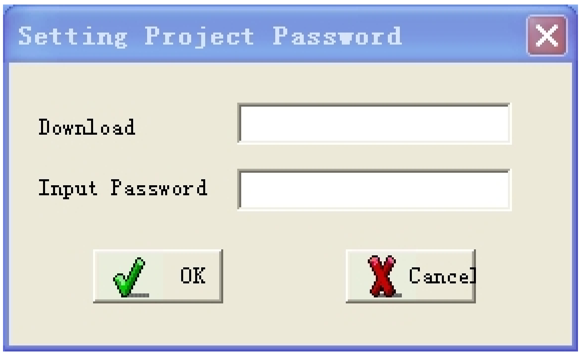
1) In the **Project Manager** window, click on the **Tools** menu and then the **Download a Project** sub-menu or click on the button in the toolbar, and you will see the **Download a Project** dialog box as shown in Figuur 17.5.



Slave IP Address: 192.168.1.10

Figuur 17.5

2) Click on the **Set Password** button, and you will see a dialog box as shown in Figuur 17.6.



Figuur 17.6

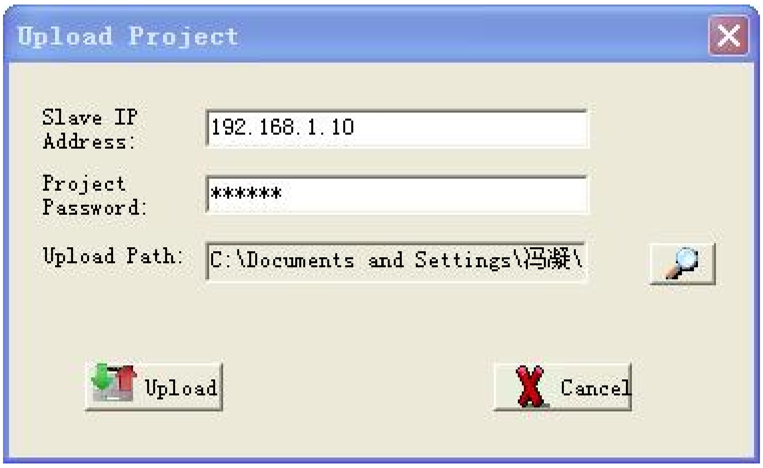
3) Set the password in Figuur 17.6, and then click on **OK**.

17.4 Uploading Projects

EASY allows you to upload projects to the PC from the HMI.

To upload a project, do as follows:

1) In the **Project Manager** window, click on the **Tools** menu and then the **Upload a Project** sub-menu or click on the button in the toolbar, and you will see the **Upload a Project** dialog box as shown in Figuur 17.7.



Upload Project

Slave IP Address: 192.168.1.10

Project Password: ******

Upload Path: C:\Documents and Settings\冯凝\

Figuur 17.7

The configuration parameters in Figuur 17.7 are described as follows:

- **Slave IP Adres**: refers to the IP adres of the HMI.
- **Project Password**: refers to the password you set for the project.
- **Upload Path**: defines the path for saving the project uploaded to the PC from the HMI.

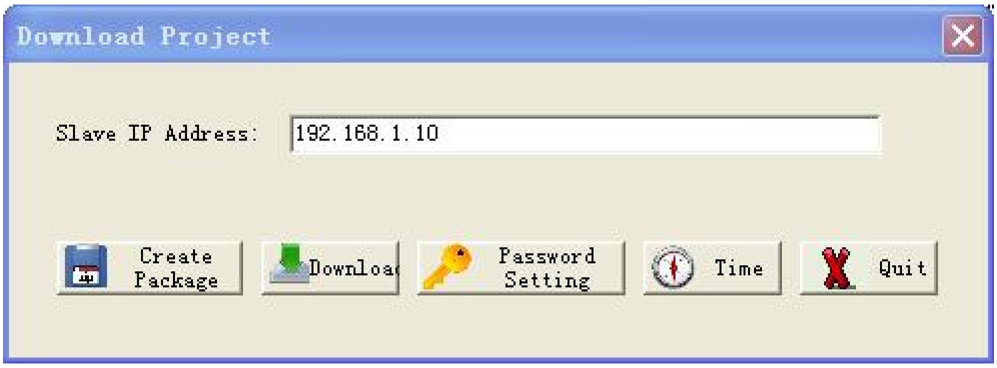
2) After you conFiguur all the parameters, click on the **Upload** button.

17.5 Time Calibration

The time calibration function of EASY is for calibrating the system time of the HMI, namely, to keep it consistent with that of the PC.

To calibrate the system time of the HMI, do as follows:

1) In the **Project Manager** window, click on **Tools** menu and then the **Download a Project** sub-menu, or click on the button in the toolbar, and you will see the **Download a Project** dialog box as shown in Figuur 17.8.



Download Project

Slave IP Address: 192.168.1.10

Figuur 17.8

2) Specify the IP adres of the HMI behind **Slave IP Adres**, and then click on the **Calibrate** button.

17.6 Downloading Upgrade Packages

The Download Upgrade Packages function allows you to specify and then download a upgrade package to the HMI.

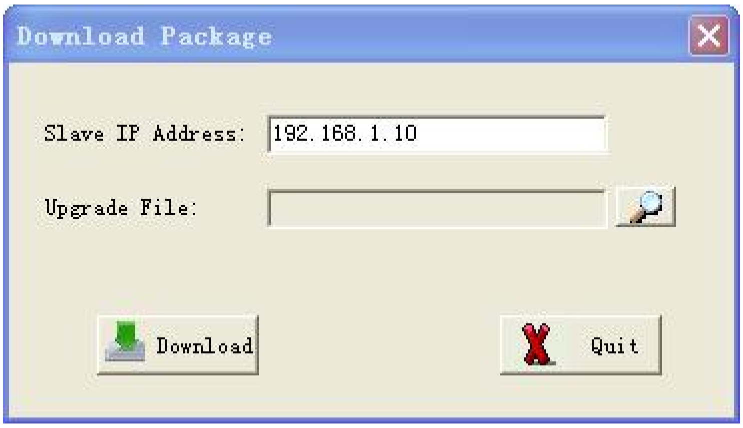
A upgrade package actually refers to the upgrade file with the file extension name as **lcp**. A upgrade package may refer to either of the following:

- Upgrade file generated after you click on the **Generate a Download Package** button in the **Download a Project** dialog box as shown in Figuur 17.8.

  This upgrade file is saved in the sub-folder **download** in the project path. By downloading this upgrade package, you can upgrade the project in the HMI.
- Other upgrade packages provided by the system, which are for upgrading the bottom system or realizing other functions.

To download a upgrade package, do as follows:

1) In the **Project Manager** window, click on the **Tools** menu and then the **Download Upgrade Package** sub-menu, or click on the button in the toolbar, and you will see the **Download Upgrade Package** dialog box as shown in Figuur 17.9.



Slave IP Address: 192.168.1.10

Figuur　17.9

The configuration parameters in Figuur 17.9 are described as follows:

- **Slave IP Adres**: refers to the IP adres of the HMI.
- **Upgrade File**: specifies the upgrade file which is required to be downloaded to the HMI.

2) After you conFiguurd all the parameters, click on the **Download** button.

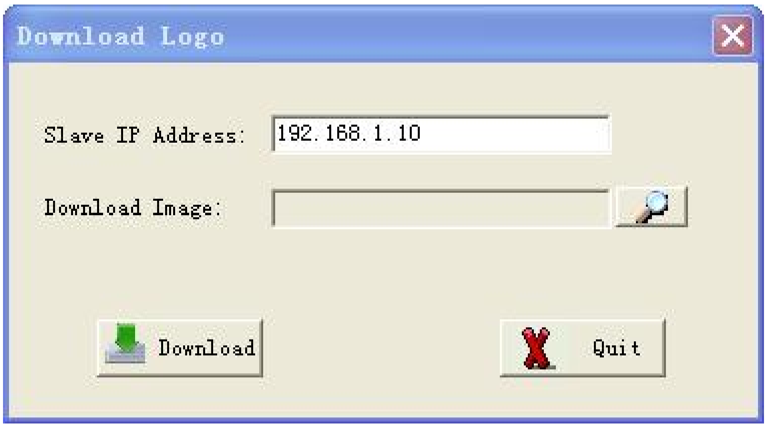
   Once the downloading of the upgrade package is complete, the HMI might need a restart depending on the contents of the upgrade package.

17.7　Downloading Startup Interface

The Downloading Startup Interface function allows you to set the start-up interface of the HMI to be displayed at the system start.

To download the startup interface, do as follows:

1) In the **Project Manager** window, click on the **Tools** menu and then the **Download Startup Interface** sub-menu, or click on the button in the toobar, and you will see the **Download Startup Interface** dialog box as shown in Figuur 17.10.



Slave IP Address: 192.168.1.10

Figuur 17.10

The configuration parameters are described as follows:

- **Slave IP Adres**: refers to the IP adres of the HMI.
- **Interface to Download**: specifies the name of the file which contains the startup interface.

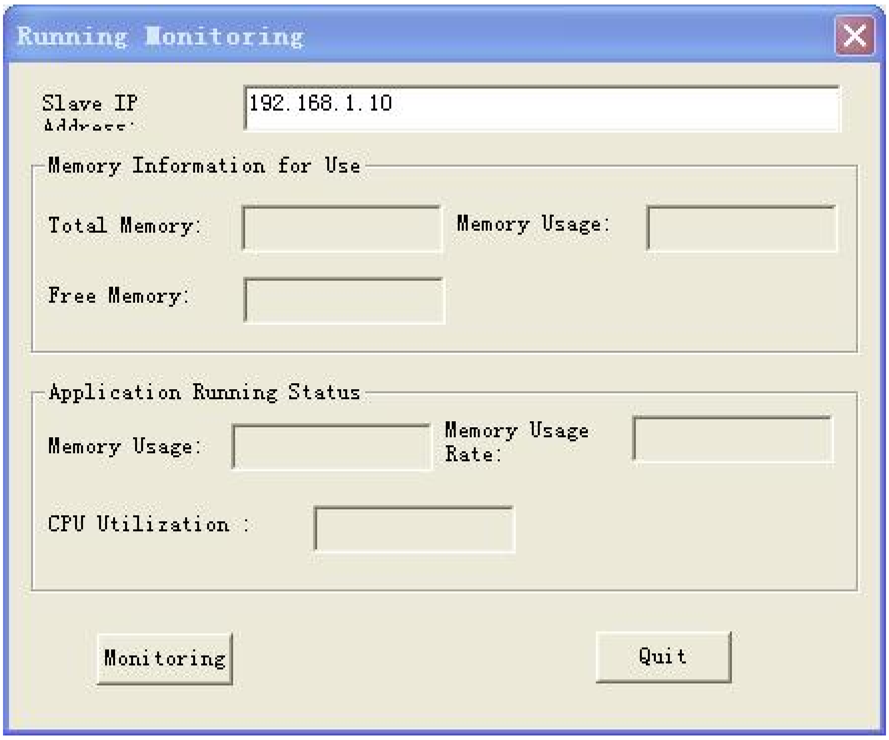
2) After you conFiguur all the parameters, click on the **Download** button.

17.8 Monitoring HMI Operation

The Monitoring HMI Operation function allows you to monitor the CPU and Memory utilization during the HMI operation.

To monitor the HMI operation, do as follows:

1) In the **Project Manager** window, click on the **Tools** menu and then the **Monitor HMI Operation** sub-menu, and you will see the **Operation Monitoring** dialog box as shown in Figuur 17.11.



Slave IP Address: 192.168.1.10

Figuur 17.11

2) Set the IP adres of the HMI behind **Slave IP Adres**, and then click on the **Start Monitoring** button, which starts the monitoring of the HMI operation.

In this dialog box, you can monitor the following aspects of the HMI operation:

- ➢ Memory Utilization

- Total Memory: Total memory size of the HMI.
- Used Memory: Size of the HMI memory which is currently being used for the HMI operation.The memory here refers to the memory taken by all the applications running in the HMI, not only the memory taken by the user-defined project, but also that taken by the bottom system.
- **Available Memory**: Size of the HMI memory which is available for other usages.

➢ Application Status

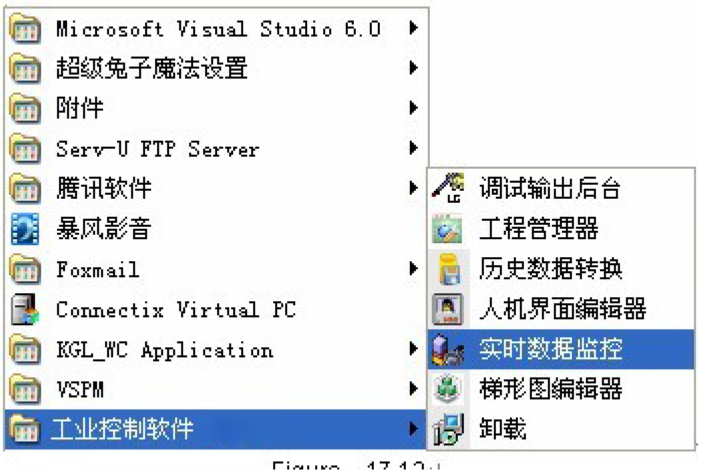
- Used Memory: Size of the HMI memory taken by the user-defined project.
- Used Memory Ratio: Ratio of the memory taken by the user-defined project against the total size of the HMI memory.
- CPU Utilization: Amount of CPU utilized by the user-defined project.

17.9 Monitoring Real-Time Data

The Monitoring Real-Time Data function allows you to monitor the real-time data of the project currently running in the HMI.

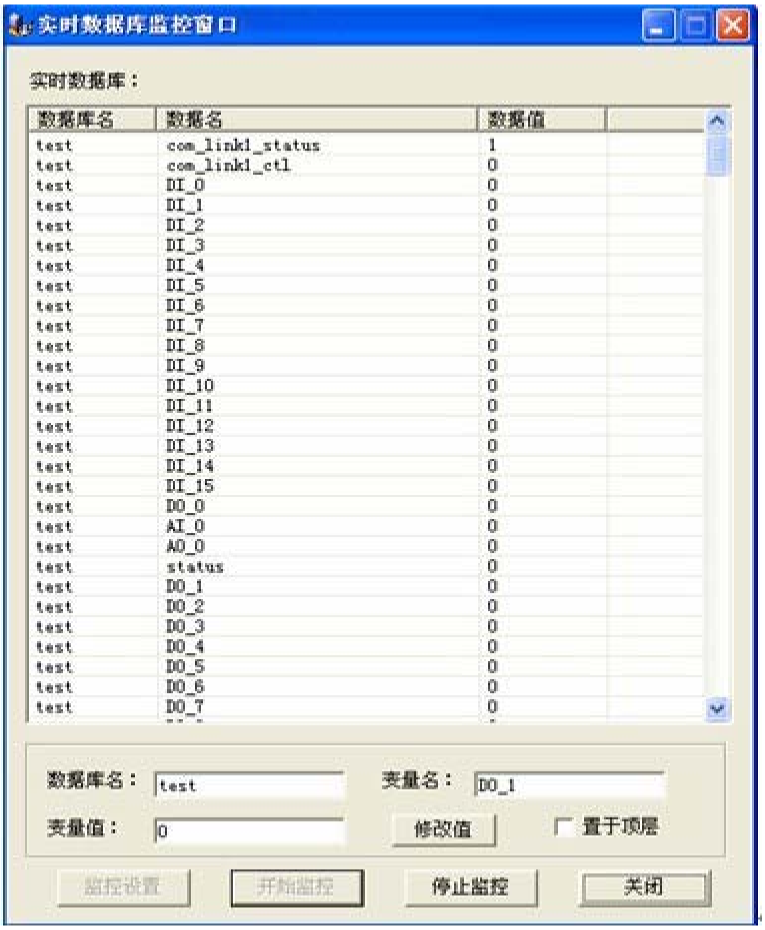
To monitor the real-time data of a project, do as follows:

1) Click on **Start**, go to **EASY Industrial Control Software**, and you can see a **Real-Time Data Monitoring** tool, as shown in Figuur 17.12.



Figuur 17.12

2) Click on **Real-Time Data Monitoring**, and the **EASY Real-Time Data Monitoring** window will be displayed, as shown in Figuur 17.13.



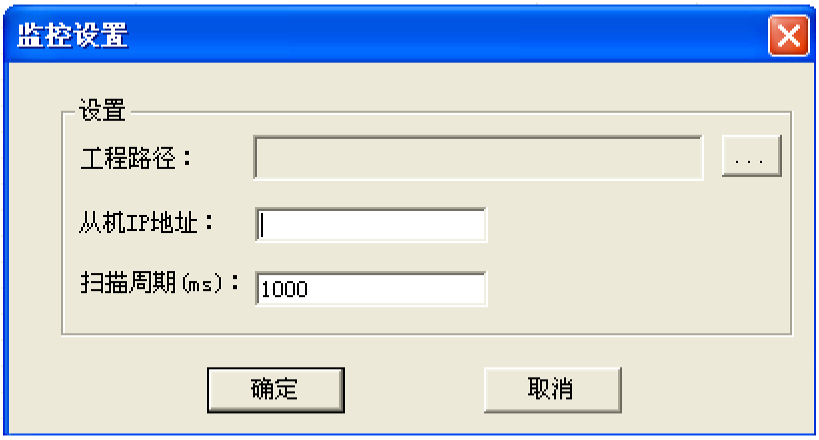
数据库名	数据名	数据值
test	com_link1_status	1
test	com_link1_ctl	0
test	DI_0	0
test	DI_1	0
test	DI_2	0
test	DI_3	0
test	DI_4	0
test	DI_5	0
test	DI_6	0
test	DI_7	0
test	DI_8	0
test	DI_9	0
test	DI_10	0
test	DI_11	0
test	DI_12	0
test	DI_13	0
test	DI_14	0
test	DI_15	0
test	DO_0	0
test	AI_0	0
test	AO_0	0
test	status	0
test	DO_1	0
test	DO_2	0
test	DO_3	0
test	DO_4	0
test	DO_5	0
test	DO_6	0
test	DO_7	0

数据库名：test　变量名：DO_1

变量值：0　□ 置于顶层

Figuur　17.13

3) Before starting monitoring the real-time data, you need to conFiguur the monitoring settings: Click on the **Monitoring Setting** button, and you will see the **Monitoring Setting** dialog box as showin in Figuur 17.14.



工程路径：

从机IP地址：

扫描周期(ms)：1000

Figuur　17.14

The configuration parameters in Figuur 17.14 are described as follows:

- **Project Path**: specifies the path where the user-defined project is saved. The project specified here must be the one currently running in the HMI.
- **Slave IP Adres**: specifies the IP adres of the HMI.

- **Scanning Cycle**: defines the interval period for the monitoring tool to obtain data from the HMI.

4) After you conFiguur all the above parameters, click on **OK** in the **Monitoring Setting** dialog box, and all the data in the real-time database of the selected project will be listed in the **Real-Time Database** list in Figuur 17.13.
5) Click on the **Start Monitoring** button, and the system starts monitoring the currently running project.

   During the monitoring, you can not only obtain the real-time value but also forcibly change the value of the data in the real-time database.

   Double-click on the data you want to change in the data list, and you can see **Database Name**, **Variable Name**, and **Variable Value** related to the selected data displayed at the bottom of the **EASY Real-Time Data Monitoring** window as shown in Figuur 17.13.

   Alternatively, you can enter the database name and variable name of the data you want to change at the bottom of the **EASY Real-Time Data Monitoring** window, enter the value to be changed to behind **Variable Value**, and then click on **Modify Value** button.
6) To stop the monitoring, click on the **Stop Monitoring** button.
7) To always see the **EASY Real-Time Data Monitoring** window on top, you can check the **Always on top** option.

17.10 Downloading Projects to Flash Drive

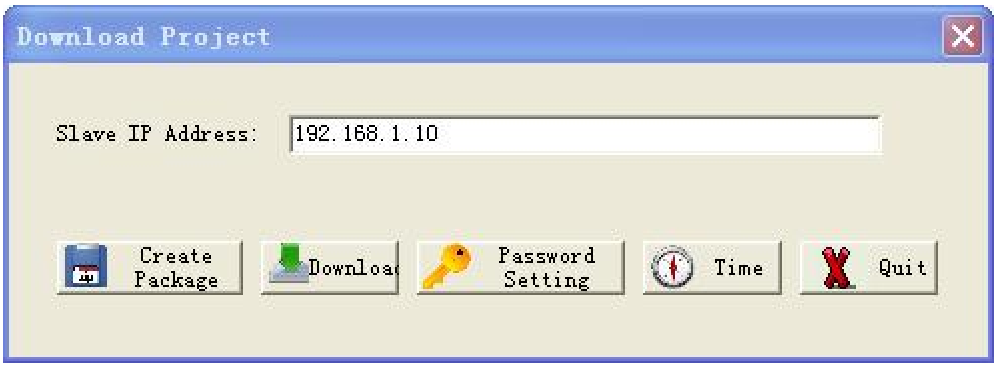
The Downloading Projects to Flash Drive function allows you to download project files to the HMI using the flash drive.

To download project files through the flash drive, follow the steps below:

1) Generate the downloading package.

To generate a downloading package, do as follows:

a) In the **Project Manager** window, click on the **Tools** menu and then the **Download a Project** sub-menu, or click on the button in the toolbar, and you will see the **Download a Project** dialog box as shown in Figuur 17.15.



Figuur 17.15

b) Specify the IP adres of the HMI behind **Slave IP Adres**, and then click on the the **Generate a Download Package** button.

The file **lcupdate.lcp** will be generated and saved to the **download** sub-folder in the path where the project is saved. This **lcupdate.lcp** file is the upgrade package file for this project.

2) Copy the downloading package to a flash drive.

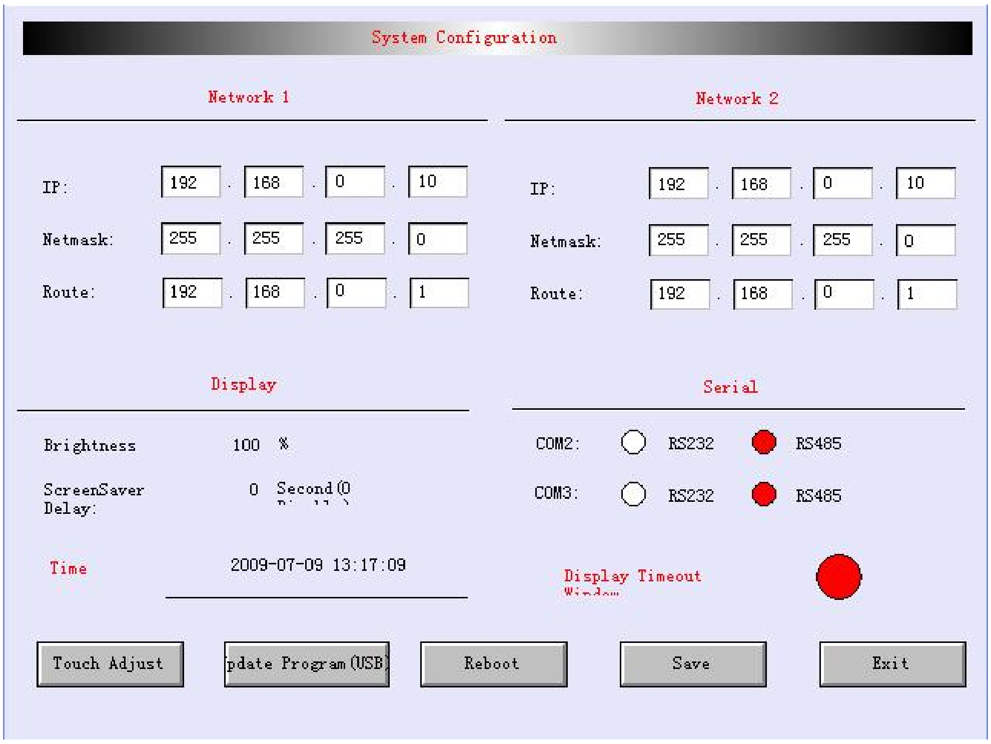
Copy the upgrade file **lcupdate.lcp** from the **download** sub-folder of the project path to a flash drive.

3) Import the package file from the flash drive to the HMI.

Insert the flash drive to the USB port of the HMI. You can import the upgrade package file to the HMI in the following two ways:

- Use the **Program Upgrade (USB)** button in the **System Setting** window.

  To see the **System Setting** window, press down in sequence the top left corner, the top right corner, and then the bottom left corner of the HMI. Th e **System Setting** window will be shown as Figuur 17.16.



System Configuration

Network 1

IP: 192 . 168 . 0 . 10

Netmask: 255 . 255 . 255 . 0

Route: 192 . 168 . 0 . 1

Network 2

IP: 192 . 168 . 0 . 10

Netmask: 255 . 255 . 255 . 0

Route: 192 . 168 . 0 . 1

Display

Brightness 100 %

ScreenSaver Delay: 0 Second (0

Time 2009-07-09 13:17:09

Serial

COM2: ○ RS232 ● RS485

COM3: ○ RS232 ● RS485

Display Timeout ●

Figuur 17.16

Click on the **Program Upgrade (USB)** button, and the upgrade file **lcupdate.lcp** will be copied to the HMI.

- Use the system function **prog_upgrade**.

  During the interface configuration, you can call directly the system function **prog_upgrade** to copy the upgrade file **lcupdate.lcp** from the flash drive to the HMI.

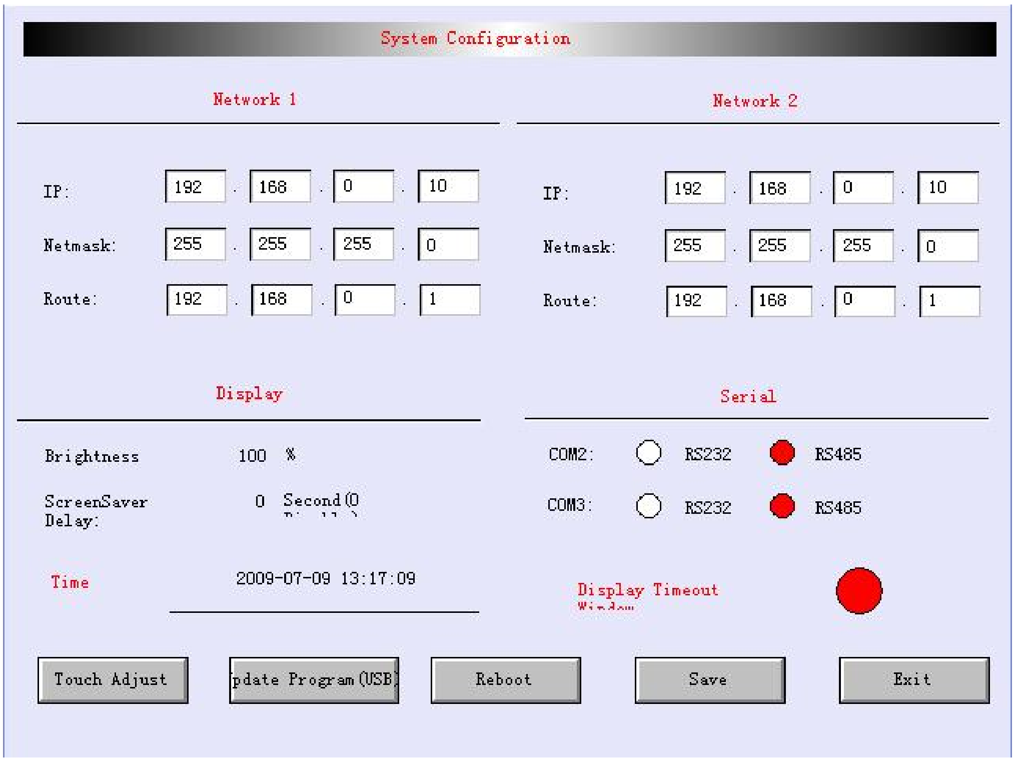
17.11 System Setting Window

EASY allows you to conFiguur the basic HMI system parameters, such as those related to the network card and the touch screen, in the **System Setting** window.

To see the **System Setting** window, use one of the following two ways:

- Press down in sequence the top left corner, the top right corner, and then the bottom left corner of the HMI.
- Call the system function **hmi_sys_set_wnd** during the interface configuration.

The **System Setting** window is shown as Figuur 17.17.



Network 1

IP: 192 . 168 . 0 . 10

Netmask: 255 . 255 . 255 . 0

Route: 192 . 168 . 0 . 1

Network 2

IP: 192 . 168 . 0 . 10

Netmask: 255 . 255 . 255 . 0

Route: 192 . 168 . 0 . 1

Display

Brightness 100 %

ScreenSaver Delay: 0 Second(0

Time 2009-07-09 13:17:09

Serial

COM2: ○ RS232 ● RS485

COM3: ○ RS232 ● RS485

Display Timeout Window ●

Figuur 17.17

The **System Setting** window allows you to achieve the following four functions.

17.11.1 Setting Parameters

You can modify all the parameters listed in the **System Setting** window. Click on **Save Settings**, and all the modified values will be saved to the system.

You can modify the following parameters:

1. **Network Card Settings**

The network card settings cover two network cards: **Network Card 1** and **Network Card 2**. Depending on the model of the HMI, some HMIs might have only one network port. For these HMIs, you need only to conFiguur settings for **Network Card 1**. For the HMIs which have two network ports, you need to conFiguur settings for both of them.

The network card settings include the following parameters:

- **IP Adres**: specifies the IP adres of the network card.
- **Subnet Mask**: specifies the subnet mask of the network where the network card is conFiguurd.
- **Default Route**: specifies the default route of the network where the network card is conFiguurd.

2. **Display Settings**

The display settings include the following parameters:

- **Screen Brightness**: sets the brightness level of the touch screen.

- **Screensaver Delay**: defines the delay time of the screensaver.
  If you do not do any operations on the HMI after the defined delay time period, the sysem will enable the screensaver.
  If you set this parameter to 0, the screensaver will be disabled.

3. Serial Port Settings

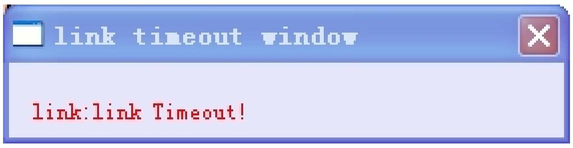
Depending on the model of the HMI, some HMIs provide 3 serial ports, among which COM1 is a dedicated RS232 serial port, while COM2 and COM3 work as both RS232 and RS485 serial ports. You can choose whether to have COM2 and COM3 work as RS232 or RS485 serial ports.

4. Time Settings

In the **Time Settings** part, you can set the system time of the HMI.

5. Timeout Window Display Settings

As described in a previous section, you can conFiguur links through the **Device Configuration** node in the **Project Manager** window. If the communication on one link stops, the **Communication Timeout** window will be displayed by default, as shown in Figuur 17.18.



link:link Timeout!

Figuur 17.18

Click on 🔴 behind **Timeout Window Display** to conFiguur whether to display the **Communication Timeout** window: If you set it to red as 🔴, the window will be displayed; if you set it to white as ⚪, the window will not be displayed.

17.11.2 Adjusting the Touch Screen

In the **System Setting** window as shown in Figuur 17.17, click on the **Adjust Touch Screen** button, and you will see the touch screen adjustment interface. You can see a cross sign + on the top left corner, top right corner, bottom left corner, bottom right corner, and the middle of the interface. You can click on the middle point of each + to adjust the touch screen.

17.11.3 Program Upgrade (Through the USB Port)

Click on the **Program Upgrade (USB)** button in the **System Setting** window as shown in Figuur 17.17, and you can download the project upgrade file to the HMI from the flash drive. For more details, see section 17.10 Downloading Projects to Flash Drive.

17.11.4 System Restart

To restart the HMI system, click on the **System Restart** button in the **System Setting** window as shown in Figuur 17.17.

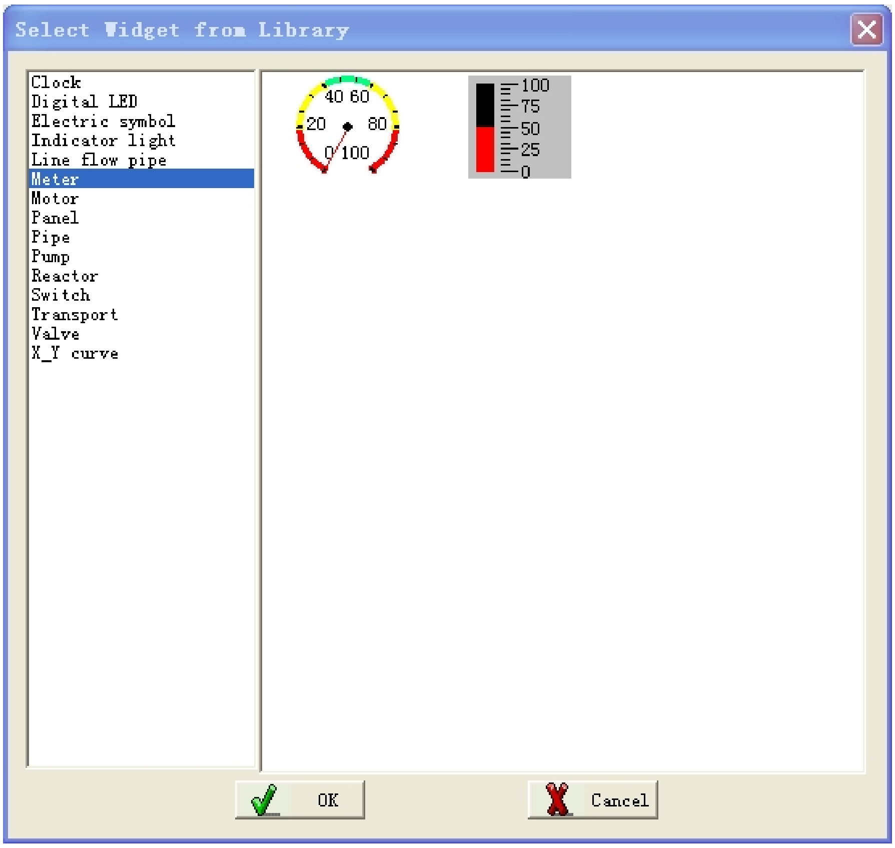
Chapter 18 Gallery Controls

18.1   Overview

EASY provides a control gallery with various powerful control components. They can be used to easily and vividly show the control procedure and process according to the requirements of various projects.

To add a control component from the gallery, do as follows:

1) Click on the **Gallery** button 图库 in the toolset on the left side of the **Interface Editor** window, and you will see the dialog box **Select from Gallery** as shown in Figuur 18.1.



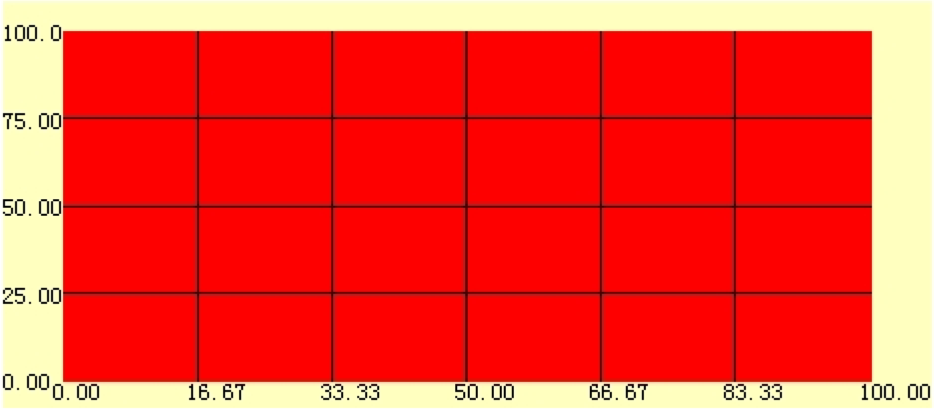
Figuur 18.1

2) Click on the graphic component you need, and click on **OK**.
3) Move the cursor to the editing area on the right side of the window, and you can see the cursor in the shape of a cross.
4) Drag the cursor, and you can see the selected component is displayed.

18.2 X_Y Curve

18.2.1 Overview

In the **Select from Gallery** dialog box, click on the **X_Y Curve** component and move the cursor to the editing area on the right side of the window, and you can see the cursor become a cross. Grag the mouse to draw a rectangle, and the X_Y curve will be displayed within this rectangle, as shown in Figuur 18.2.



Figuur 18.2

In the middle of the displayed X_Y curve component is the drawing area with gridlines. The X and Y curves will be drawn within this drawing area: The X-axis will be on the left and the Y-axis at the bottom. Select the X_Y curve component, and you will see 8 small rectangles on the four sides. You can use these 8 small rectangles to move or resize the component.

The gridlines in the drawing area are composed of two types of dividing lines: the ones vertical to the X-axis and the ones vertical to the Y-axis. You can set the number of dividing lines at each direction. For example, if you set the number of vertical lines to 5, then the whole drawing area will be divided into 6 identical areas by the 5 vertical lines.

The X_Y curve is different from the real-time trend curve as follows:

- Real-Time Trend Curve: The X-axis is time and the Y-axis is variable. Th real-time trend curve shows the change tendency of the variable against the change of time.
- X_Y Curve: Both the X-axis and the Y-axis are variables. The X_Y curve shows the change tendancy of one variable against the change of the other.

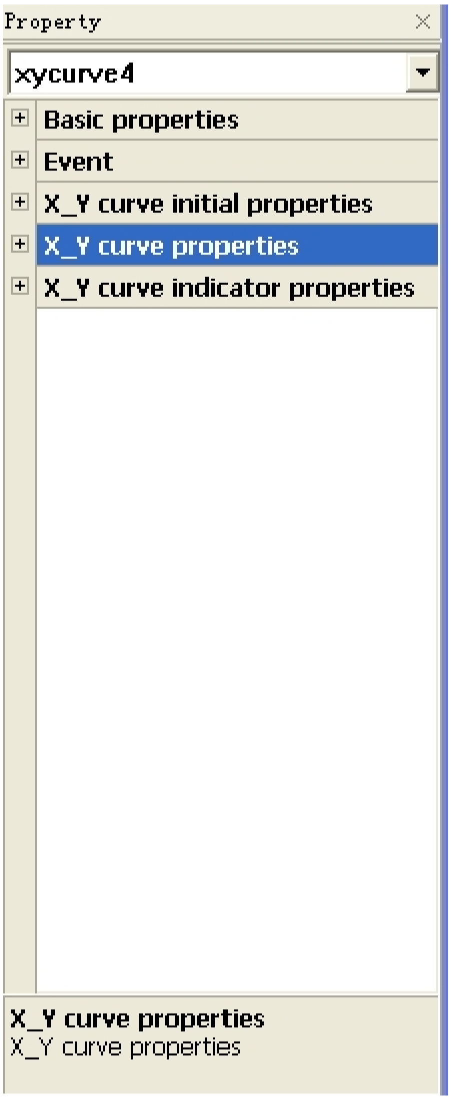
18.2.2 Properties of the X_Y Trend Curve

After you draw an X_Y trend curve, select the curve with a left click, the **Property List** pane will be displayed on the right side of the editing area listing all the properties of this

X_Y curve.

The properties of the X_Y curve are composed of the following five property nodes, as shown in Figuur 18.3:

- Basic Properties
- Events
- Initial X_Y Trend Curve Properties
- X_Y Trend Curved Line Properties
- X_Y Trend Indicator Line Properties



X_Y curve properties
X_Y curve properties

Figuur 18.3

For more details on basic properties, see section 5.3.3 Basic Properties.

For more details on events, see section 5.3.5 Events.

18.2.2.1 Initial X_Y Trend Curve Properties

An X_Y trend curve control can display curves for multiple data. This section describes the common properties of all the X_Y trend curves.

Property	Description	Remarks on Dynamic Properties
Minimum X-axis Value	Value of startpoint on the X-axis.	The return value is numeric. Changing these two values will zoom the curve vertically.
Maximum X-axis Value	Value of the endpoint on the X-axis.	
Minimum Y-axis Value	Value of the startpoint on the Y-axis.	The return value is numeric. Changing these two values will zoom the curve horizontally.
Maximum Y-axis Value	Value of the endpoint on the Y-axis.	
Maximum Number of Points	Maximum number of data points distributed horizontally across the curve.	No dynamic properties.
Number of Displayed Points	Number of the data points displayed horizontally across the curve.	The return value is an integer. This property, together with the **Maximum Number of Data Points** property, zoom the curve horizontally.
Number of Vertical Lines	Number of the dividing lines vertical to the Y-axis.	The return value is an integer.
Number of Vertical Lines	Number of the dividing lines vertical to the X-axis.	The return value is an integer.
Horizontal Spacing	Left or right spacing between the curve drawing area of the control and the left or right margin.	The return value is an integer.
Vertical Spacing	Top or bottom spacing between the curve drawing area of the control and the top or bottom margin.	The return value is an integer.
Background Color	Background color of the control.	The expression of the dynamic script returns the RGB values of the color.
Background Color of Curves	Backgroun color of the curve drawing area of the control.	
Color of Vertical Lines	Color of the dividing lines vertical to the Y-axis.	
Color of Horizontal Lines	Color of the dividing lines vertical to the X-axis.	
Text Color	Color of the text beside the X- and Y- axis.	
Number of Curved Lines	Number of the curved lines to be displayed; 16 the most.	No dynamic properties.
Data Source	Data source of the trend curve:	No dynamic properties.

Property	Description	Remarks on Dynamic Properties
	• **Real-Time Memory Record**: The data used in the trend curve comes from the current values of the data in a real-time data record. • File: If you save a real-time data record to a file, you can set the data source to File. In this case, the curve is a saved trend curve.	
Trend Name	Valid when **Data Source** is set to **Real-Time Memory Record**. The trend name is actually the name of a real-time data record. This name must be defined in a real-time data record in the **Project Manager** window.	No dynamic properties.
File Location	Valid when **Data Source** is set to **File**. This property defines where the real-time data record is kept, internal flash or the CF card.	The return value is 0 or 1: • 0: Internal flash • 1: CF card
File Name	Valid when **Data Source** is set to **File**. This property defines the name of the real-time data record file.	The return value is a string with the name of the real-time record file.
Start Point	Valid when **Data Source** is set to **File**. This property defines from which data point of the record the curve is to display.	The return value is int. Changing this value will move the curve horizontally.

18.2.2.2 X_Y Trend Curved Line Properties

You can conFiguur data and color properties for each curve, and you can conFiguur for up to 16 curves.

Properties	Description	Remarks on Dynamic Properties
Variable X	Name of the variable corresponding to the X-axis of the trend curve. This variable must be defined in the real-time data record. If you do not specify this variable, then the curve will not be displayed.	The return value is a string with the name of the data displayed by the curve. If the returned string is a blank string, (“”), then the curve will not be displayed.
Variable Y	Name of the variable corresponding to the Y-axis of the trend curve. This variable must be defined in the real-time data	

	record. If you do not specify this variable, then the curve will not be displayed.	
Curve Color	Color of the trend curve.	The expression or dynamic script returns the RGB values of the color.

18.2.2.3 X_Y Trend Indicator Line Properties

Properties	Description	Remarks on Dynamic Properties
Allow Indicator Line	Defines whether the control allows indicator lines.	The return value is int: • 0: Not Allow • Non 0: Allow
Color of Indicator Line	Color of the indicator lines.	The expression or dynamic script returns the RGB values of the color.
X Value Variable	The X-axis data value of the curve the indicator line points to is saved to this variable.	The return value is a string with the name of the time variable.
Y Value Variable	The Y-axis data value of the curve the indicator line points to is saved to this variable.	The return value is a string with the name of the data value variable.

18.3 Pump

Properties	Description	Remarks on Dynamic Properties
Pump Start Color	Defines the color of the indicator light.	The expression or dynamic script returns the RGB values of the color.
General Pump Color	Defines the color of the whole pump.	

18.4 Conveying Belt

Properties	Description	Remarks on Dynamic Properties
Wheel Color	Defines the color of the wheel.	The expression or dynamic script returns the RGB values of the color.
Transmitter Color	Defines the color of the transmitter.	

18.5 Valve

Properties	Description	Remarks on Dynamic Properties
Valve Start Color	Defines the color for the start of the valve.	The expression or dynamic script returns the RGB values of the color.
Valve End Color	Defines the color for the end of the valve.	

18.6 Reactor

Properties	Description	Remarks on Dynamic Properties
Reactor Body Start Color	Defines the color for the start of the reactor body.	The expression or dynamic script returns the RGB values of the color.
Reactor Body End Color	Defines the color for the end of the reactor body.	
Reactor Leg Start Color	Defines the color for the start of the reactor leg.	
Reactor Leg End Color	Defines the color for the end of the reactor leg.	
Reactor Foot Start Color	Defines the color for the start of the reactor foot.	
Reactor Foot End Color	Defines the color for the end of the reactor foot.	

18.7 Pipe

Properties	Description	Remarks on Dynamic Properties
Start Color	Defines the color for the start of the pipe.	The expression or dynamic script returns the RGB values of the color.
End Color	Defines the color for the end of the pipe.	
Direction	Defines the direction of the pipe.	The expression or the dynamic script returns the value ranging from 1 to 4, which stand for Top Right, Bottom Right, Top Left, and Bottom Left from top to bottom in the drop-down list.
Pipe Width	Defines the width of the pipe.	The expression or dynamic script returns an int value, which is the width of the pipe.

18.8 Switch

Properties	Description	Remarks on Dynamic Properties
Variable Name	Specifies the name of the associated variable.	No dynamic properties.
General Background Color	Sets the general background color.	The expression or dynamic script returns the RGB values of the color.
Button Background Color	Sets the background color for the small buttons in the middle.	
Top Button Shadow Color	Sets the color of the top shadow for the small buttons in the middle.	
Bottom Button Shadow Color	Sets the color of the bottom shadow for the small buttons in the middle.	
Top Shadow Color	Sets the color of the top shadow for the whole switch.	
Bottom Shadow Color	Sets the color of the bottom shadow for the whole	

	switch.	
Button Direction	Sets the direction of the button.	The expression or dynamic script returns the value 0 or 1, which stand for Bounced Up and Pressed Down from top to bottom in the drop-down list.
Shadow Width	Sets the shadow width for the whole switch.	The expression or dynamic script returns an int, which is the width of the shadow.
Button Shadow Width	Sets the shadow width for the button.	

18.9 Motor

Property	Description	Remarks on Dynamic Properties
Motor Indicator Color	Sets the color of the indicator light of the motor.	The expression or dynamic script returns the RGB values of the color.

18.10 Panel

Properties	Description	Remarks on Dynamic Properties
External Panel Status	Defines the status of the external panel, Raised or Embedded.	The expression or dynamic script returns the value 0 or 1, which stands for Raised and Embedded from top to bottom in the drop-down list.
External Panel Highlight Color	Sets the highlight color for the external panel.	The expression or dynamic script returns the RGB values of the color.
External Panel Shadow Color	Sets the shadow color for the external panel.	
Internal Panel Highlight Color	Sets the highlight color for the internal panel.	
Internal Panel Shadow Color	Sets the shadow color for the internal panel.	
Internal Panel Filling Color	Sets the filling color for the internal panel.	
External Panel Filling Color	Sets the filling color for the external panel.	

Internal Panel Status	Defines the status of the internal panel, Raised, Embedded, or Transparent.	The expression or dynamic script returns the value ranging from 0 to 2, which stand for Raised, Embedded, and Transparent from top to bottom in the drop-down list.
Panel Space	Sets the space between the internal and external panels.	The expression or dynamic script returns an int value, which is the space between the internal and external panels.

18.11 Soft Keyboard

Properties	Description	Remarks on Dynamic Properties
Background Color	Sets the background color of the keyboard.	The expression or dynamic script returns the RGB values of the color.
Text Color	Sets the color of the keyboard text.	
Key Color	Sets the color of the keys on the keyboard.	
Key Font Size	Sets the font size for the text on the keyboard.	The expression or dynamic script returns an int value: • 0: Default font size • Other int: Font size specifies by the returned int value

18.12 Clock

18.12.1 Properties of the Panel Clock

Properties	Description	Remarks on Dynamic Properties
Hour Hand Color	Sets the color of the hour hand.	The expression or dynamic script returns the RGB values of the color.
Minute Hand Color	Sets the color of the minute hand.	
Second Hand Color	Sets the color of the second hand.	
Disk Background Color	Sets the background color of the clock disk.	
Arc Color	Sets the color of the frame arc.	

Arc Width	Sets the width of the frame arc.	The expression or dynamic script returns an int, which is the width of the outer arc.

18.12.2 Properties of the Digital Clock

Properties	Description	Remarks on Dynamic Properties
Background Color	Sets the background color.	The expression or dynamic script returns the RGB values of the color.
Digital Display Color	Sets the color of the digital display.	
Digital Background Color	Sets the color of the digital background.	
Frame Start Color	Sets the color for the start of the frame.	The expression or dynamic script returns the RGB values of the color.
Frame End Color	Sets the color for the end of the frame.	
LED Width	Sets the width of the LED.	The expression or dynamic script returns an int value, which is the width of the LED on the clock.
Edge Width	Sets the width of the edge of the clock.	The expression or dynamic script returns an int value, which is the width of the edge of the clock.
Rim Width	Sets the width of the outer rim of the clock.	The expression or dynamic script returns an int value, which is the width of the outer rim of the clock.

18.13 Digitron

Properties	Description	Remarks on Dynamic Properties
Background Color	Sets the background color for the control.	The expression or dynamic script returns the RGB values of the color.
Digital Display Color	Sets the color of the displayed digits.	
Digit Background Color	Sets the background color of the displayed digits.	
Frame Start Color	Sets the color for the start of the frame.	
Frame End Color	Sets the color for the end of the frame.	
Value	Sets the data value.	The expression or dynamic script returns

		a float or int value, which is the displayed value of the digitron.
Integral Digits	Sets the number of digits for the integer.	The expression or dynamic script returns an int value, which is the number of digits of the integer.
LED Width	Sets the width of the LED.	The expression or dynamic script returns an int value, which is the width of the LED.
Decimal Digits	Sets the number of decimal digits.	The expression or dynamic script returns an int value, which is the number of decimal digits.
Edge Width	Sets the width of the edge.	The expression or dynamic script returns an int value, which is the width of the edge.
Frame width	Sets the width of the frame.	The expression or dynamic script returns an int value, which is the width of the frame.

18.14 File List

Properties	Description	Remarks on Dynamic Properties
Number of Rows	Defines the number of rows to be displayed in the file saved in the HMI.	The expression or dynamic script returns an int value, which is the number of rows to be displayed in the file.
Redrawing Variable	Defines regarding the update of the file contents.	The expression or dynamic script returns an bit value: • 0: No action • 1: Refresh
File Location	Files can be saved in the following locations: • Internal Flash Historical Data • Internal Flash User Data • CF Card User Data • CF Card Historical Data • USB Flash Drive The designer can set the source of the file displayed in the file list.	The expression or dynamic script returns an int value: • 0: FlashL Historical Data • 1: Internal FlashL User Data • 2: CF Card User Data • 3: CF Card Historical Data • 4: USB Flash Drive
List File	Corresponds to the name of the	The expression or dynamic

Properties	Description	Remarks on Dynamic Properties
	associated file variable. For files of the universal class, you can use the wildcard.	script returns a string, which is the type of the file name.
File Name Variable	Defines the list of file names to be displayed.	The expression or dynamic script returns a string, which is the file name.
List Item Height	Sets the height of each list item.	No dynamic properties.

18.15 Meter

18.15.1 Disk Meter

Properties	Description	Remarks on Dynamic Properties
Background Type	• Transparent: sets the background color to transparent. In this case, the set outer and inner background colors do not work. • Non-Transparent: sets the inner and outer background colors to different colors.	The expression or dynamic script returns an int value: • 0: The background color is set to transparent. In this case, the set outer and inner background colors do not work. • 1: The inner and outer background colors are set to different colors.
Drawing Type	Two types of meter hands are available: fast and normal.	No dynamic properties.
Number of Secondary Scales	Sets the number of secondary scales.	The expression or dynamic script returns an int value, which is the number of secondary scales.
Inner Background Color	Sets the inner background color.	The expression or dynamic script returns the RGB values of the color.
Outer Background Color	Sets the outer background color.	
Scale Color	Sets the color of the scale.	
Text Color	Sets the color of the text.	
Arc Color	Sets the color of the arc.	
Hand Color	Sets the color of the meter hand.	
Color of Normal Interval	Sets the color of the normal interval.	
Color of	Sets the color of the abnormal	

Abnormal Interval	interval.	
Color of Warning Interval	Sets the color of the warning interval.	
Arc Width	Sets the width of the disk arc.	The expression or dynamic script returns an int value, which is the width of the disk arc.
Start Angle	Sets the start angle.	The expression or dynamic script returns an int value, which is the start point of the minimum value on the meter disk.
End Angle	Sets the end angle.	The expression or dynamic script returns an int value, which is the start point of the maximum value on the meter disk.
Minimum Value	Sets the minimum value.	The expression or dynamic script returns a float value, which is the minimum value.
Maximum Value	Sets the maximum value.	The expression or dynamic script returns a float value, which is the maximum value.
Data Value	Sets the data value.	The expression or dynamic script returns a float value, which sets the location of the hand.
Normal, Warning, and Abnormal Interval Range	Sets the range of each interval.	The expression or dynamic script returns a float value, which defines the range of each interval.

18.15.2 Scale Meter

Properties	Description	Remarks on Dynamic Properties
Background Type	• Transparent: sets the background color to transparent. In this case, the set outer and inner background colors do not work. • Non-Transparent: sets the inner and outer background colors to different colors.	The expression or dynamic script returns an int value: • 0: The background color is set to transparent. In this case, the set outer and inner background colors do not work. • 1: The inner and outer background colors are set to different colors.
Scale Type	• With Scale: displays the scale.	No dynamic properties.

Properties	Description	Remarks on Dynamic Properties
	- Without Scale: does not display the scale.	
Data Bar Direction	- Vertical: displays the meter disk vertically. - Horizontally: displays the meter disk horizontally.	The expression or dynamic script returns an int value, which is the number of the secondary scales.
Scale Location	- Left/Top: displays the meter disk to the left (with the data bar vertical) or to the top (with the data bar horizontal). - Right/Bottom: displays the meter disk to the right (with the data bar vertical) or to the bottom (with the data bar horizontal).	No dynamic properties.
Number of Primary Scales	Sets the number of the primary scales.	No dynamic properties.
Number of Secondary Scales	Sets the number of the secondary scales.	No dynamic properties.
Background Color	Sets the background color.	The expression or dynamic script returns the RGB values of the color.
Scale Color	Sets the scale color.	
Text Color	Sets the text color.	
Background Color of Data Bar	Sets the background color of the data bar.	
Filling Color of Data Bar	Sets the filling color of the data bar.	
Width of Data Bar	Sets the width of the data bar.	
Minimum Value	Sets the minimum value.	The expression or dynamic script returns a float value, which is the minimum value.
Maximum Value	Sets the maximum value.	The expression or dynamic script returns a float value, which is the maximum value.
Data Value	Sets the data value.	The expression or dynamic script returns a float value, which sets the location of the hand.

18.16 Line Pipe Moving Control

Properties	Description	Remarks on Dynamic Properties
Start Color	Sets the start color of the pipe.	The expression or dynamic script returns
End Color	Sets the end color of the pipe.	
Color of Filling Object	Sets the color of the filling object.	The expression or dynamic script returns the RGB values of the color.
Length of Filling Object	Sets the length of the filling object.	The expression or dynamic script returns an integer, which defines the length of the filling object.
Moving Direction	Sets the moving direction, to the left or to the right.	The expression or dynamic script returns bit signal corresponding to 0 (Left) or 1 (Right) in the drop-down list.
Moving Status	Sets the moving status, moving or fixed.	The expression or dynamic script returns bit signal corresponding to 0 (Fixed) or 1 (Moving) in the drop-down list.

18.17 Indicator

Properties	Description	Remarks on Dynamic Properties
Color of Indicator Light	Sets the color of the indicator light.	The expression or dynamic script returns
Base Color	Sets the color of the base.	No dynamic properties.

Chapter 19 System Variables

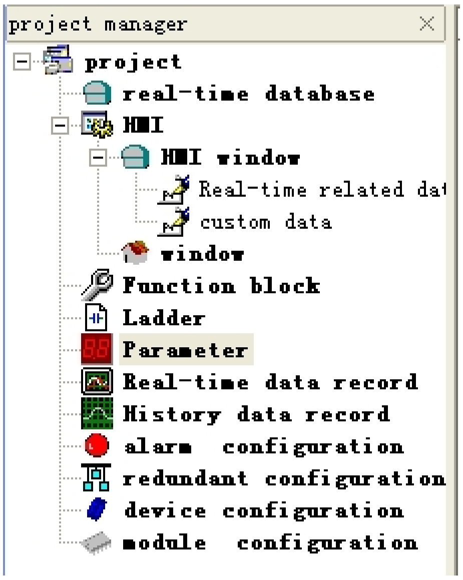
EASY defines some internal data variables which are called EASY system variables. You can use these system variables directly during the interface configuration.

With these system variables, you can read or modify the internally-defined parameters in the system to realize some special functions.

This Hoofdstuk describes all the system variables for your reference.

19.1 Configuring System Variables

The system variables are conFiguurd in the **Parameter** node in the **Project Manager** window, as shown in Figuur 19.1.



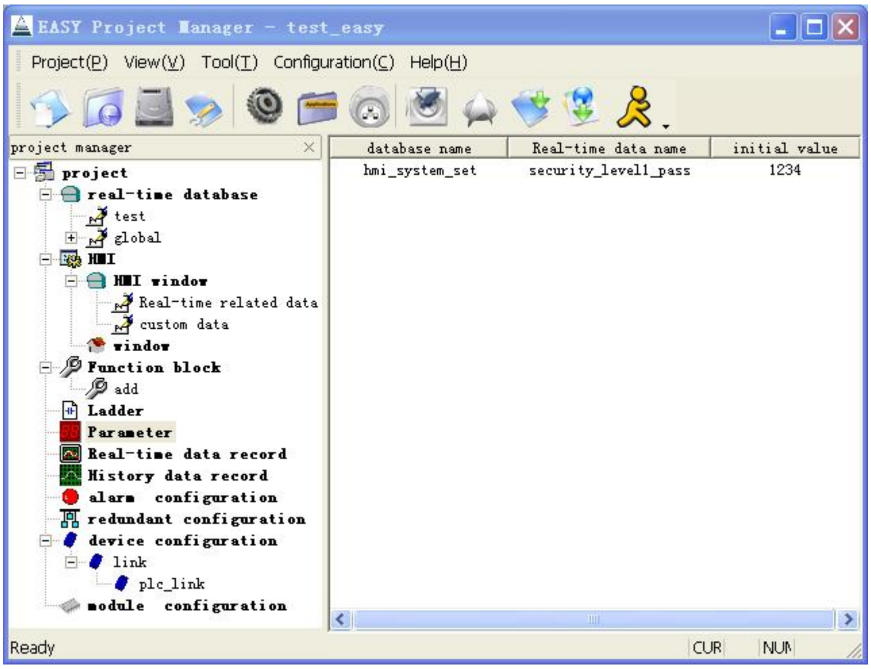
Figuur 19.1

19.1.1 Adding a System Parameter

All the system parameters are defined internally in the system. Adding a system parameter is actually setting the initial value of a system parameter. All the system parameters have an initial value during the system development. Changing the initial value actually changes the value of the system parameter.

To add a system parameter, do as follows:

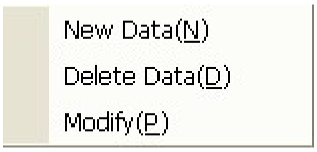
1) Select the **Parameter** node with a left click, and you will see a window as shown in Figuur 19.2.



database name	Real-time data name	initial value
hmi_system_set	security_level1_pass	1234

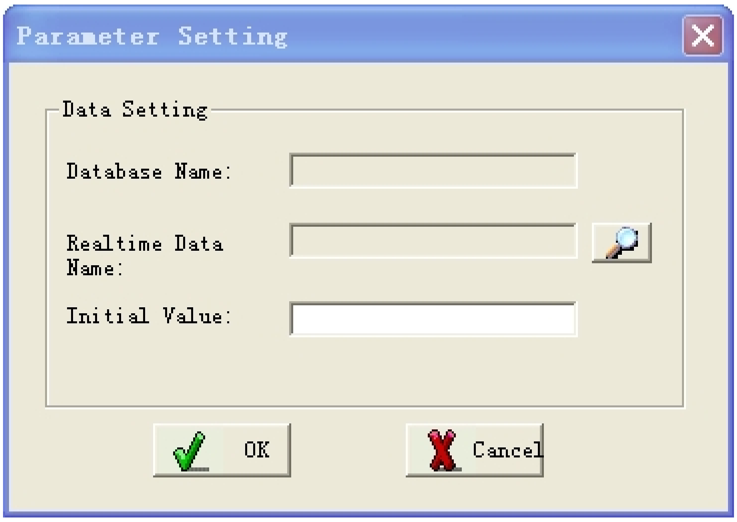
Figuur 19.2

2) In the parameter list on the right side of the window, right-click in the blank area, and you will see a right-click menu as shown in Figuur 19.3.



Figuur 19.3

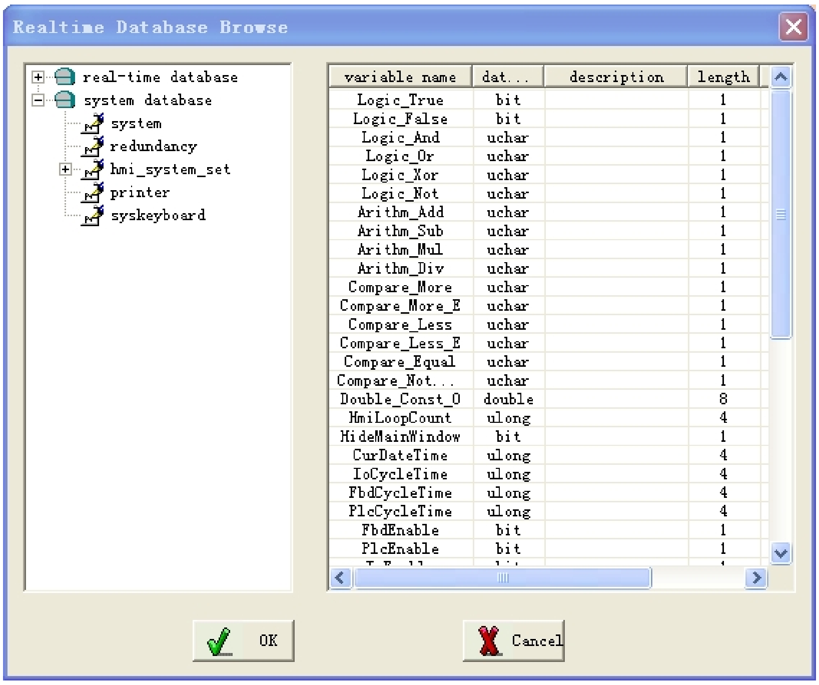
3) Select **Add Data**, and you will see the **Parameter Configuration** dialog box as shown in Figuur 19.4.



Figuur 19.4

4) Select the database name and the real-time data name.

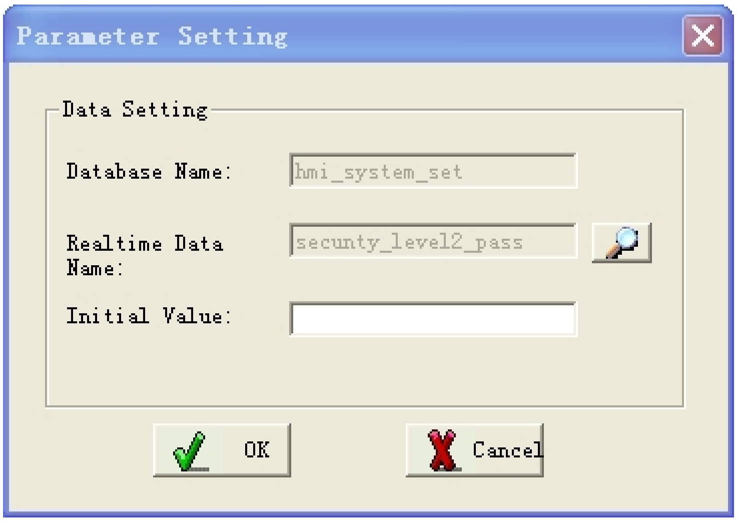
To select the database name and the real-time data name, click on the button , and you will see the **Browse Real-Time Database** window as shown in Figuur 19.5. In this window, you can see that all the system variables in the system database node are managed through the browser. Each parameter is described in details.



variable name	dat...	description	length
Logic_True	bit		1
Logic_False	bit		1
Logic_And	uchar		1
Logic_Or	uchar		1
Logic_Xor	uchar		1
Logic_Not	uchar		1
Arithm_Add	uchar		1
Arithm_Sub	uchar		1
Arithm_Mul	uchar		1
Arithm_Div	uchar		1
Compare_More	uchar		1
Compare_More_E	uchar		1
Compare_Less	uchar		1
Compare_Less_E	uchar		1
Compare_Equal	uchar		1
Compare_Not...	uchar		1
Double_Const_0	double		8
HmiLoopCount	ulong		4
HideMainWindow	bit		1
CurDateTime	ulong		4
IoCycleTime	ulong		4
FbdCycleTime	ulong		4
PlcCycleTime	ulong		4
FbdEnable	bit		1
PlcEnable	bit		1

Figuur 19.5

5) Double-click on a system parameter, and the selected system parameter will be filled to the corresponding text box automatically, as shown in Figuur 19.6.



Database Name: hmi_system_set

Realtime Data Name: secunty_level2_pass

Initial Value:

Figuur 19.6

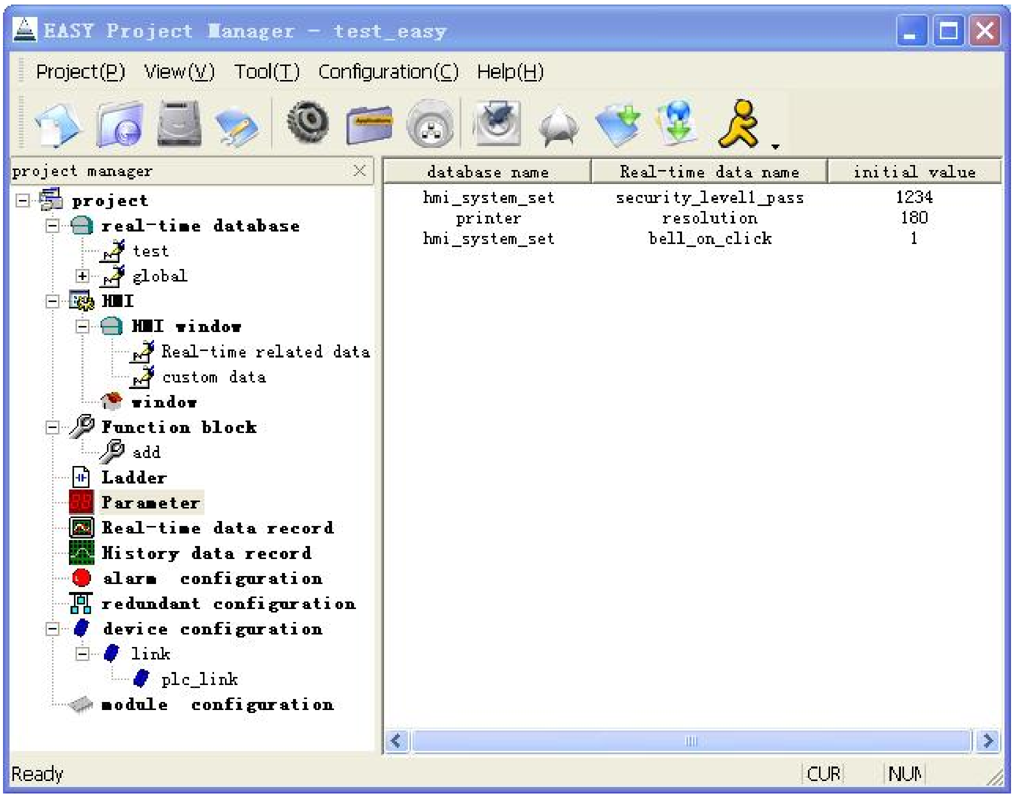
The parameter **Initial Value** defines the initial value of the system parameter according to the meaning of the system parameter.

6) After all the system parameters are conFiguurd, click on **OK**.

The system parameter is conFiguurd (or added) successfully. You may add other system parameters in the same way.

After you conFiguur all the related system parameters, you will see a window as shown in Figuur 19.7.

The following sections of this Hoofdstuk describe more details about the meaning of the system parameters.



database name	Real-time data name	initial value
hmi_system_set	security_level1_pass	1234
printer	resolution	180
hmi_system_set	bell_on_click	1

Figuur 19.7

19.2 System Variables for the Interface

Database Name	Variable Name	Data Type	Default Value	Description
system	HmiLoopCount	ulong		Count value of how many times the interface refreshes. The value of this variable is added 1 every time the interface refreshes.

	HideMainWindow	bit	• 0: Simulated operation on PC • 1: Operation on the HMI	• 1: Menus in the target Windows window are hidden. • 0: Menus in the target Windows window are displayed.
	HmiHeartbeat	bit		Heartbeat of the HMI during operation. The value of this variable switches between 0 and 1 repeatedly during the interface refreshing. The value of this variable switches every time the interface refreshes.
	HmidbDefCycleTime	ulong	500	Cycle for the repeated interface refreshing (unit: ms).

19.3 System Variables for Device Configuration

Database Name	Variable Name	Data Type	Default Value	Description
system	IoEnable	bit	1	• 1: Device management enabled. • 0: Device management disabled.
hmi_system_set	link_timeout_wnd_on	bit	1	• 1: The system displays the **Communication Timeout** window automatically when communication timeout occurs. • 0: The system does not display the **Communication Timeout** window when communication timeout occurs.
	link_timeout_wnd_x	short	-1	Value of the X-axis on the top left corner of the **Communication Timeout** window.
	link_timeout_wnd_y	Short	-1	Value of the Y-axis on the top left corner of the

Database Name	Variable Name	Data Type	Default Value	Description
				Communication Timeout window.

19.4 System Variables for Function Blocks

Database Name	Variable Name	Data Type	Default Value	Description
system	FbdEnable	bit	1	• 1: Function block support enabled. • 0: Function block support disabled. In this case, none of the conFiguurd function blocks will be executed.
	FbdCycleTime	ulong	100	Cycle for executing all function blocks (unit: ms).
	FbdHeartbeat	bit		Heartbeat of the execution of function blocks. The value of this variable switches between 0 and 1 repeatedly during the running of the function blocks. The value of this variable switches every time when all function blocks are executed in a cycle.

19.5 System Variables for the Ladder Diagram

Database Name	Variable Name	Data Type	Default Value	Description
system	PlcEnable	bit	1	• 1: Ladder diagram enabled. • 0: Ladder diagram disabled.
	PlcCycleTime	ulong	100	Cycle for executing the ladder diagram (unit: ms).
	PlcHeartbeat	bit		Heartbeat of the ladder diagram during its operation. The value of this variable switches between 0 and 1 repeatedly during the operation of the ladder diagram. The value of this variable switches every time the ladder diagram is executed for a new round.

19.6 System Variables for Access Management

Database Name	Variable Name	Data Type	Default Value	Description
hmi_system_set	cur_security_level	ushort	0	Current security level of the system.
	security_level1_pass~ security_level10_pass	string	Blank	Stands individually for the password of each security level from level 1 to level 10. You can set the password of each security level according to your needs or requirements. If you do not set the password of a security level or you leave it to blank, it means that this security level is not in use. The password can be set to up to 20 characters (including the ending character **\0** of a string)

19.7 System Variables for Printing

Database Name	Variable Name	Data Type	Default Value	Description
printer	printer_name	string	escp2-me1	Printer model
	resolution	string	360sw	Printer resolution (DPI)
	paper	string	A4	Paper type, such as A4
	left_margin	float	1	Left printing margin (unit: cm)
	top_margin	float	1	Top printing margin (unit: cm)
	scaling	float	1	Printing scaling
	status	long	0	Printer status: • 0: Idle • 1: Printing • 2: Printer not connected • 3: Unsupported printer type • 4: Printing error

19.8 System Variables Related to Time

Database Name	Variable Name	Data Type	Default Value	Description
system	Year	ushort	0	Current date (year).
	Month	uchar	0	Current date (month).
	Day	uchar	0	Current date (day).
	Hour	uchar	0	Current date (hour).
	Minute	uchar	0	Current date (minute).
	Second	uchar	0	Current date (second).
	CurDateTime	ulong		From the date 1970/1/1 to the second read of the current time.
hmi_system_set	Year_set	ushort	0	Sets the current date (year).
	Month_set	uchar	0	Sets the current date (month).
	Day_set	uchar	0	Sets the current date (day).
	Hour_set	uchar	0	Sets the current date (hour).
	Minute_set	uchar	0	Sets the current date (minute).
	Second_set	uchar	0	Sets the current date (second).

19.9 System Variables for LCD Settings

Database Name	Variable Name	Data Type	Default Value	Description
system	ScreenWidth	ushort	640	Width of the screen pixels of the HMI
	ScreenHeight	ushort	480	Height of the screen pixels of the HMI
hmi_system_set	brightness	uchar	100	LCD backlight value (range: 0-100)
	blankdelaysec	ushort	0	Screensaver delay time (unit: seconds) (0 means to disable the screensaver.)

19.10 System Variables for Serial Port Communication

Database Name	Variable Name	Data Type	Default Value	Description

hmi_system_set	com2_is_rs232	bit	0	• 1: Serial port 2 works as RS232. • 0: Serial port 2 works as RS485.
	com3_is_rs232	bit	0	• 1: Serial port 3 works as RS232. • 0: Serial port 3 works as RS485.

19.11 System Variables for Touch Alarm

Database Name	Variable Name	Data Type	Default Value	Description
hmi_system_set	bell_on_click	bit	0	If you set the value of this variable to 1, an alarm will be triggered automatically when you click on the touch screen.
	bell_loop_count	ulong	1	Defines how many times an alarm is triggered.
	bell_freq	ulong	10	Defines how long the alarm lasts.

19.12 System Variables for the Keyboard

Database Name	Variable Name	Data Type	Default Value	Description
syskeyboard	NumberMinValue	float	0	Minimum value to be entered from the digital keyboard.
	NumberMaxValue	float	100	Maximum value to be entered from the digital keyboard.
	KeyboardBuffer	string		Buffer for keyboard input
	Prompt	string		Prompt information for keyboard input.
	NumberKeyboardWindow	string		Name of the window for digital keyboard input. The internal keyboard is used when the value of this variable is set to blank.
	PasswordChar	char	0	Password characters. No password is available when the value of this variable is set to 0.
	IsEnterPressed	bit	0	The value 1 is returned when you press down the **Enter** key.
	TextKeyboardWindow	string		Name of the window for text

				keyboard input. The internal keyboard is used when the value of this variable is set to blank.

19.13 System Variables for Links

Database Name	Variable Name	Data Type	Default Value	Description
hmi_system_set	link_timeout_wnd_on	bit	1	• 1: The **Communication Timeout** window is displayed when timeout occurs during the communication over the link. • 0: The **Communication Timeout** window is not displayed when timeout occurs during the communication over the link.
	link_timeout_wnd_x	short	-1	Location of the start point on the X-axis in the **Communication Timeout** window. The default value -1 means that the **Communication Timeout** window will be centrally aligned.
	link_timeout_wnd_y	short	-1	Location of the start point on the Y-axis in the **Communication Timeout** window. The default value -1 means that the **Communication Timeout** window will be centrally aligned.

19.14 System Variables for the Disk Space

Database Name	Variable Name	Data Type	Default Value	Description
hmi_system_set	flash_free_disk_space	ulong		Size of free space of the

				internal flash (unit: Kbyte).
	cf_free_disk_space	ulong		Size of free space of the CF card (unit: Kbyte)
	low_space_wnd_on	bit	1	• 1: A prompt window will be displayed when the disk does not have enough space. • 0: No prompt window will be displayed when the disk does not have enough space.
	low_space_text	string		Prompt information to be displayed when the disk does not have enough space.

19.15 System Variables for the Scroll Bar

Database Name	Variable Name	Data Type	Default Value	Description
hmi_system_set	HScrollSize	ulong	0	Width of the horizontal scroll bar. The system internal width is used when the value of this variable is 0.
	VScrollSize	ulong	0	Width of the vertical scroll bar. The system internal width is used when the value of this variable is 0.

Chapter 20 System Functions

EASY provides some internally defined system functions. With these function, you can realize specific system functions, such as window display, historical data process, printing, and script commissioning.

Besides, you can also call these system functions directly during the interface configuration.

This Hoofdstuk lists all the system functions supported by EASY.

20.1 System Functions for the Interface

20.1.1 hmi_window_show

Original Function: int **hmi_window_show**(**char** *window_name*)

Function Description: To show a specified window.

Return Values: 0 Failed

1 Successful

Parameters: *window_name*: Name of the target window you want to display.

Example: hmi_window_show("test")

20.1.2 hmi_window_hide

Original Function: int **hmi_window_hide**(**char** *window_name*)

Function Description: To close a specified window.

Return Values: 0 Failed

1 Successful

Parameters: *window_name*: Name of the target window you want to close.

Example: hmi_window_hide("test")

20.1.3 hmi_window_show_modal

Original Function: int **hmi_window_show_modal**(**char** *window_name*)

Function Description: To display a modal dialog box.

Return Values: 0 Failed

1 Successful

Parameters: *window_name*: Name of a window.

Example: hmi_window_show_modal("test")

20.1.4 hmi_window_exit_modal

Original Function: int **hmi_window_exit_modal**(**char** **window_name*)

Function Description: To exit the **Modal** dialog box. Call this function when you want to exit a modal dialog box

Return Values: 0 Failed

1 Successful

Parameters: *window_name*: Name of a wiondow.

Example: hmi_window_exit_modal("test")

20.1.5 data_input_window

Original Function: int **data_input_window**(**char ****varname*, **char ****caption*, **double** *minvalue,* **double** *maxvalue*, **int** *dec_num*)

Function Description: Function for data input. When you call this function, the **Data Input** window will be displayed. The data you enter in this window will be assigned as the value for the parameter ***varname***.

Return Values: 0 Failed

1 Successful

Parameters: *varname*: Name of a variable. The data you enter in the **Data Input** window will be assigned as the value for this parameter.

caption: Prompt information to be displayed as the title of the **Data Input** window.

minvalue: Minimum value allowed for the data entered.

maxvalue: Maximum value allowed for the data entered.

dec_num: Number of decimals.

Example: data_input_window("test.data", "test", 0, 100, 2)

20.1.6 data_input_window_pwd

Original Function: int **data_input_window_pwd**(**char ****varname*, **char ****caption*, **double** *minvalue,* **double** *maxvalue*, **int** *dec_num,* **int** *passwd*)

Function Description: Function for data input (password display option available). When you call this function, the **Data Input** window will be displayed. The data you enter in this window will be assigned as the value for the parameter ***varname***.

Return Values: 0 Failed

1 Successful

Parameters: *varname*: Name of a variable. The data you enter in the **Data Input** window will be assigned as the value for this parameter.

caption: Prompt information to be displayed as the title of the **Data Input** window.

minvalue: Minimum value allowed for the data entered.

maxvalue: Maximum value allowed for the data entered.

dec_num: Number of decimals.

passwd: 1: Password Display; 0: Normal Display.

Example: data_input_window_pwd("test.data", "test", 0, 100, 2, 1)

20.1.7 text_input_window

Original Function: int **text_input_window**(**char ****varname*, **char ****caption,* **int** *passwd*)

Function Description: Function for text input. When you call this function, the **Text Input** window will be displayed. The text you enter in this window will be assigned as the value for the parameter ***varname***.

Return Values: 0 Failed

1 Successful

Parameters: *varname*: Name of a variable. The text you enter in the **Text Input** window will be assigned as the value for this parameter.

caption: Prompt information to be displayed as the title of the **Text Input** window.

passwd: 1: Password Display; 0: Normal Display.

Example: text_input_window("test.data", "test", 1)

20.1.8 msgbox

Original Function: int **msgbox**(**char ****caption*, **char ****text*, **int** *type*)

Function Description: Function for displaying a message box.

Return Values: MSG_IDOK: You can click on the **OK** button.

MSG_IDCANCEL: You can click on the **Cancel** button.

MSG_IDABORT: You can click on the **Abort** button.

MSG_IDRETRY: You can click on the **Retry** button.

MSG_IDIGNORE: You can click on the **Ignore** button.

MSG_IDYES: You can click on the **Yes** button.

MSG_IDNO: You can click on the **No** button.

Parameters: *caption*: Title of a window.

text: Message content.

type: Type of a message box, valued as follows:

MSG_MB_OK: The **OK** button is displayed.

MSG_MB_OKCANCEL: The **OK** and **Cancel** buttons are displayed.

MSG_MB_YESNO: The **Yes** and **No** buttons are displayed.

MSG_MB_RETRYCANCEL: The **Retry** and **Cancel** buttons are displayed.

MSG_MB_ABORTRETRYIGNORE: The **Abort**, **Retry** and **Ignore** buttons are displayed.

MSG_MB_YESNOCANCEL: The **Yes**, **No** and **Cancel** buttons are displayed.

MSG_MB_ICONSTOP: The **Stop** icon is displayed.

MSG_MB_ICONQUESTION: The **Question** icon is displayed.

MSG_MB_ICONEXCLAMATION: The **Exclamation** icon is displayed.

MSG_MB_ICONINFORMATION: The **Information** icon is displayed.

MSG_MB_DEFBUTTON1: The first button is defined as default.

MSG_MB_DEFBUTTON2: The second button is defined as default.

MSG_MB_DEFBUTTON3: The third button is defined as default.

Example: msgbox("Error", "Open file failed", MSG_MB_OK)

20.1.9 hmi_center_window

Original Function: int **hmi_center_window**(**char** **window_name*)

Function Description: Function for displaying the window in the center.

Return Values: 0 Failed

1 Successful

Parameters: *window_name*: Name of a window.

Example: hmi_center_window("test")

20.2 System Functions for Parameters

20.2.1 rtdb_param_mem_to_rtdb

Original Function: int **rtdb_param_mem_to_rtdb**()

Function Description: To copy the data from the HMI memory to the real-time database as the value of parameters.

Return Values: 0 Failed
1 Successful

Parameters: N/A

Example: rtdb_param_mem_to_rtdb()

20.2.2 rtdb_param_rtdb_to_mem

Original Function: int **rtdb_param_rtdb_to_mem**()

Function Description: To copy the value of parameters from the real-time database to the HMI memory.

Return Values: 0 Failed
1 Successful

Parameters: N/A

Example: rtdb_param_rtdb_to_mem()

20.2.3 sys_save_param

Original Function: int **sys_save_param**()

Function Description: To save parameter data to devices.

Return Values: 0 Failed
1 Successful

Parameters: N/A

Example: sys_save_param()

20.3 System Functions for Script Commissioning

20.3.1 debug_set_ip

Original Function: void **debug_set_ip**(**const char** ***ip*)

Function Description: To set the IP adres of the host on which the

Commissioning Output Background tool is running.

Return Values: N/A

Parameters: *ip*: IP adres of the host on which the **Commissioning Output Background** tool is running.

Example: debug_set_ip("127.0.0.1");

20.3.2 debug_printf

Original Function: void **debug_printf**(**const char** **format*, …)

Function Description: To export the printing commissioning information to the commissioning host. Use this function in the same way as the library function **printf** in the standard C language.

Return Values: N/A

Parameters: *format*: String for format control. Same as for the library function **printf** in the standard C language.

…: Optional parameter. Same as for the library function **printf** in the standard C language.

Example: debug_printf("i=%d\n", i);

20.4 System Functions for Real-Time Trend Curves

20.4.1 rtdb_log_save_file

Original Function: int **rtdb_log_save_file**(**char ****logname*, **int** *save_dir*, **char ****filename*)

Function Description: To save a real-time data record into a file.

Return Values: 0 Failed

1 Successful

Parameters: *logname*: Name of a real-time data record.

save_dir: 0: HMI internal flash; 1: CF card.

filename: Name of the file for saving the real-time data record.

Example: rtdb_log_save_file("real", 0, "recfile.log")

20.4.2 rtdb_get_log_data

Original Function: int **rtdb_get_log_data**(**char ****logname*, **char ****dataname*, **u8 ****buf*, **int** *log_number*)

Function Description: To obtain data from the current real-time data record.

Return Values: 0: Failed

Other values: Actual volume of data collected.

Parameters: *logname*: Name of a real-time data record.

dataname: Name of the data variable for which data is to be collected.

buf: Buffer for the collected data. You need to assign space for the butter in advance.

log_number: Volume of data to be collected.

Example: rtdb_get_log_data("real", "test.data1", buf, 100)

20.4.3 rtdb_get_log_data_from_file

Original Function: int **rtdb_get_log_data_from_file**(**int** *file_path,* **char** **filename,* **char ****dataname,* **u8** **buf,* **int** *log_number,* **int** *start_pt*)

Function Description: To obtain data from the saved real-time record file.

Return Values: 0: Failed

Other values: Actual volume of data collected.

Parameters: *file_path*: 0: HMI internal flash; 1: CF card.

filename: Name of the file where the real-time data record is saved.

dataname: Name of the data variable for which data is to be collected.

buf: Buffer for the collected data. You need to assign space for the butter in advance.

log_number: Volume of data to be collected.

start_pt: Start point from where data is collected.

Example: rtdb_get_log_data_from_file(0, " recfile.log ", "test.data1", buf, 100, 0)

20.4.4 rtdb_log_save_usr_file

Original Function: int **rtdb_log_save_usr_file**(**usr_log_file_info_t ****usr_log*, **int** *save_dir,***char ****filename*)

Function Description: To save a user record of data into a file.

Return Values: 0: Failed

1: Saved Successfully

Parameters: *usr_log*: Name of the user record.

save_dir: Directory for saving the user record file.

0: File saved to the HMI internal flash.

1: File saved to the CF card.

Filename: Name of the file for saving the user record.

Example: rtdb_log_save_usr_file("test ","0 ", "testfile")

20.5 System Functions for Historical Data Processing

20.5.1 sys_history_download

Original Function: int **sys_history_download**()

Function Description: Function for downloading historical data; to copy historical data from the HMI internal flash or the CF card to a thumb drive.

Return Values: 0 Failed

1 Successful

Parameters: N/A

Example: sys_history_download()

20.5.2 history_query_all

Original Function: int **history_query_all**(**char ***query_var_name*)

Function Description: To query historical data records from all historical databases by the specified fields. For details, see section 9.5.1 Inquiring Historical Data by Specified Fields.

Return Values: 0: Failed

1: Successful

Parameters: *query_var_name*: Name of the historical data to which the field to be searched is associated.

Example: history_query_all("test.query_data1")

20.5.3 history_query_data

Original Function: int **history_query_data**(**char ***query_var_name,* **char ***history_name*)

Function Description: To query historical data records from the defined historical database by the specified fields. For details, see section 9.5.1 Inquiring Historical Data by Specified Fields.

Return Values: 0: Failed

1: Successful

Parameters: *query_var_name*: Name of the historical data with which the field to be searched is associated.

history_name: Name of the historical data record.

Example: history_query_data("test.query_data1", "his")

20.5.4 hislist_query_data

Original Function: int **hislist_query_data**(**char** ***window_name, **char** ***widget_name*)

Function Description: To query the values of the various fields of the record currently selected in the **Historical List** control.

Return Values: 0: Failed

1: Successful

Parameters: *window_name*: Name of the window where the **Historical List** control is located.

widget_name: Name of the graphic component of the **Historical List** control.

Example: history_query_data("main_pic", "hislist1")

20.5.5 hislist_delete_data

Original Function: int **hislist_delete_data**(**char** ***window_name, **char** ***widget_name*)

Function Description: To delete the currently selected record from the **Historical List** control.

Return Values: 0: Failed

1: Successful

Parameters: *window_name*: Name of the window where the **Historical List** control is located.

widget_name: Name of the graphic component of the **Historical List** control.

Example: history_delete_data("main_pic", "hislist1")

20.6 System Function for Alarming

20.6.1 alarm_confirm

Original Function: int **alarm_confirm**(**char** ***window_name, **char**

***widget_name*)

Function Description: To confirm the alarm record currently selected in the **Alarm Window** control.

Return Values: 0: Failed

1: Successful

Parameters: *window_name*: Name of the window where the **Alarm Window** control is located.

widget_name: Name of the graphic component in the alarm window.

Example: alarm_confirm("main_pic", "alarmwnd1")

20.7 System Function for Printing

20.7.1 print_window

Original Function: int **print_window**(**char** **window_name*, **int** *x1*, **int** *y1*, **int** *x2*, **int** *y2*)

Function Description: To print the selected area from the currently displayed window.

Return Values: 1: Successful

Other values: Failed

Parameters: *window_name*: Name of the window to be printed.

x1: X-axis value of the top left corner of the area to be printed.

y1: Y-axis value of the top left corner of the area to be printed.

x2: X-axis value of the bottom right corner of the area to be printed.

y2: Y-axis value of the bottom right corner of the area to be printed.

20.8 System Functions for Real-Time Database Read/Write

20.8.1 rtdb_set_data_value_by_name

Original Function: int **rtdb_set_data_value_by_name**(**char** **dbname*,**char** **dataname*,**void** **data_value*)

Function Description: You can use this function in the expansion module to write the data in the real-time database. This function sets the value of the data in the real-time database.

Return Values: 0 Failed

1 Successful

Parameters: *dbname*: Name of a database.

Dataname: Name of a data.

data_value: Value of a data. The value of a data varies according to the data type. For example, the bit data has only 1 byte, the long data has 4 bytes, and the length of the string data is user-defined.

Example:

float value=1.0;

rtdb_set_data_value_by_name(“test”,”Ldata”,&value);

The above function sets the value of the data **Ldata** in the database **test** to 1.0.

20.8.2 rtdb_get_data_value_by_name

Original Function: int **rtdb_get_data_value_by_name**(**char** ***dbname*,**char** ***dataname*,**void** ***data_value*)

Function Description: You can use this function in the expansion module to read the data in the real-time database. This function obtains the value of the data from the real-time database.

Return Values: 0 Failed

1 Successful

Parameters: *dbname*: Name of a database.

dataname: Name of a data.

data_value: Value of a data. The parameter **data_value** requires you to assign the space in advance. For example, you need to assign 1 byte of space for the bit data and 4 bytes of space for the long data.

Example:

float value;

rtdb_get_data_value_by_name(“test”,”Ldata”,&value);

The above function obtains the current value of the data **Ldata** from the database **test** and saves the value to the variable **value**.

20.9 System Functions for Serial Port Communication

20.9.1 serial_open

Original Function: int **serial_open**(**const char ****device***,int** *baud***,int** *parity***,int** *data_bits***,int** *stop_bits***,int** *timeout*)

Function Description: To open a serial port.

Return Values: -1 Failed

Other value Serial port handle

Parameters: *device*: Serial port name, for example, COM1, COM2, or COM3.

baud: Baud rate of the serial port. At present, the supported baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200.

parity: Parity check mode, including No Parity Check (SERIAL_PARITY_NO), Odd Parity Check (SERIAL_PARITY_ODD), and Even Parity Check (SERIAL_PARITY_EVENT).

data_bits: Number of data bits, including 5, 6, 7, and 8.

stop_bits: Number of stop bits, including 1 and 2.

timeout: Duration of communication timeout (unit: ms).

Example: serial_open("COM1", 9600, SERIAL_PARITY_NO, 8, 1, 100)

20.9.2 serial_close

Original Function: int **serial_close**(**int** *serial_id*)

Function Description: To close a serial port.

Return Values: 0 Failed

1 Successful

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

Example: serial_close(1)

20.9.3 serial_flush

Original Function: int **serial_flush**(**int** *serial_id*, **int** *flag*)

Function Description: To clear the buffering data of a serial port.

Return Values: 0 Failed

1 Successful

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

flag: Flush flag.

SERIAL_FLUSH_TX: clears the currently unsent data.

SERIAL_FLUSH_RX: clears the data in the receiving buffer.

SERIAL_FLUSH_TX|SERIAL_FLUSH_RX: clears the data in both the sending and receiving buffers.

Example: serial_flush(1, SERIAL_FLUSH_TX|SERIAL_FLUSH_RX)

20.9.4 serial_write

Original Function: int **serial_write**(**int** *serial_id*, **char** ****buf,*** **size_t** *size*)

Function Description: To send data to a serial port.

Return Values: -1 Failed

Other value Actual length of the data sent

Parameters: *serial_id* Serial port handle (returned by the function **serial_open**)

buf Buffer for keeping the data to be sent.

size: Length of the data to be sent (unit: byte).

Example: serial_write(1,buf, sizeof(buf))

20.9.5 serial_read

Original Function: int **serial_read**(**int** *serial_id*, **char** ****buf,*** **size_t** *size*)

Function Description: To receive data through a serial port.

Return Values: -1 Failed

Other value Actual length of the data received

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

buf: Buffer for keeping the received data.

size: Size of the receiving buffer (unit: byte).

Example: serial_read(1,buf, sizeof(buf))

20.9.6 serial_poll

Original Function: int **serial_poll**(**int** *serial_id*, **int** *timeout*)

Function Description: To check whether there is data to read at a serial port.

Return Values: -1 Error

0 Timeout, which meas no data is received during the timeout period

1 Data available to be read at the serial port

Parameters: *serial_id*: Serial port handle (returned by the function **serial_open**)

Timeout: For inquiring the timeout duration (unit: ms)

Example: serial_poll(1,100)

20.10 System Functions for File Directory

20.10.1 hmi_get_usr_data_dir

Original Function: const char * **hmi_get_usr_data_dir()**

Function Description: To get an available file directory for saving user-defined files.

Return Value: Name of the available file directory.

Parameters: N/A

Example: hmi_get_usr_data_dir( )

20.10.2 hmi_get_cfcard_dir

Original Function: const char ***hmi_get_cfcard_dir**( )

Function Description: To get the CF card directory.

Return Value: Name of the CF card directory.

Parameters: N/A

Example: hmi_get_cfcard_dir( )

20.10.3 hmi_get_usb_dir

Original Function: const char ***hmi_get_cfcard_dir**( )

Function Description: To get the directory for your thumb drive.

Return Values: Name of the directory for your thumb drive.

Parameters: N/A

Example: hmi_get_usb_dir( )

20.10.4 filelist_delete_file

Original Function: int filelist_delete_file(**char ****window_name***, char** **window_name*)

Function Description: To delete the currently selected file from the file list control component.

Return Values: 0 Failed

1 Successful

Parameters: *window_name*: Name of the window where the file list control component is located.

widget_name: Name of the file list control component.

Example: filelist_delete_file("OpenLogFile","filelist1")

20.10.5 sys_mount_usb_disk

Original Function: int **sys_mount_usb_disk**( )

Function Description: To mount the thrum drive.

Return Values: 0 Failed

1 Mounting successful

Parameters: N/A

Example: sys_mount_usb_disk( )

20.10.6 sys_unmount_usb_disk

Original Function: int **sys_unmount_usb_disk**( )

Function Description: To unmount the thumb drive.

Return Values: 0 Failed

1 Unmounting successful

Parameters: N/A

Example: sys_unmount_usb_disk( )

20.10.7 sys_copy_file_to_usb_disk

Original Function: int **sys_copy_file_to_usb_disk**(**char** **filename*, **int** *automount*, **int** *file_dir*)

Function Description: To copy files from the internal flash or CF card to a thumb drive.

Return Values: 0 Failed

1 Copying successful

Parameters: *filename*: Name of the file to be copied.

Automount: Status of the thumb drive.

1: Automatic mounting or unmounting the thumb drive

0: Mounting the thumb drive by calling the function **sys_mount_usb_disk**.

file_dir: Directory for saving the file.

0: Internal flash user data directory

1: CF card

2: Internal flash historical data directory

3: CF card historical data directory

Example: sys_copy_file_to_usb_disk(“file1”,1,0)

20.10.8 sys_copy_file_from_usb_disk

Original Function: int **sys_copy_file_from_usb_disk**(**char** **filename*, **int** *automount*, **int** *file_dir*)

Function Description: To copy files from the thumb drive to the internal flash or CF card.

Return Values: 0 Failed

1 Copying successful

Parameters: *filename*: Name of the file to be copied.

automount: Status of the thumb drive.

1: Automatic mounting or unmounting the thumb drive

0: Mounting the thumb drive by calling the function **sys_mount_usb_disk**.

file_dir: Directory for saving the file.

0: Internal flash user data directory

1: CF card

2: Internal flash historical data directory

3: CF card historical data directory

Example: sys_copy_file_from_usb_disk(“file1”,1,0)

20.11 System Functions for Time and Date

20.11.1 setsystime

Original Function: int **setsystime**(**int** *year*, **int** *month*, **int** *day*, **int** *hour*, **int** *minu*, **int** *sec*)

Function Description: To set the system time.

Return Values: 1 Successful

0 Failed

Parameters: *year*: Year

month: Month

day: Day

hour: Hour

minu: Minute

sec: Second

Example: setsystime(2008, 10, 11, 10, 30, 10)

20.11.2 gettimeinfo

Original Function: int **gettimeinfo** (**unsigned long** *cur_datetime*, **int** **year*, **int** **month*, **int** **day*, **int** **hour*, **int** **minu*, **int** **sec*)

Function Description: To convert the current system time displayed in seconds (for example, the system variable **system.CurDateTime**) to the time displayed with year, month, day, hour, minute, and second.

Return Values: 1 Successful
0 Failed

Parameters: *cur_datetime*: Current time displayed in seconds.
year: Year
month: Month
day: Day
hour: Hour
minu: Minute
sec: Second

Example: int year, month, day, hour, minu, sec;
getsystime($system.CurDateTime, &year, &month, &day, &hour, &minu, &sec)

20.11.3 datetime_add

Original Function: int **datetime_add** (**int** **year*, **int** **month*, **int** **day*, **int** **hour*, **int** **minu*, **int** **sec,* **int** *add_seconds*)

Function Description: To add certain seconds on top of the current time.

Return Values: 1 Successful
0 Failed

Parameters: *year*: Year after the adding.
month: Month after the adding
day: Day after the adding
hour: Hour after the adding
minu: Minute after the adding
sec: Second after the adding
add_seconds: Number of seconds added

Example: int year, month, day, hour, minu, sec;
datatime_add(&year, &month, &day, &hour, &minu, &sec, 60)

20.12 System Functions for the Window

20.12.1 touch_adjust

Original Function: int **touch_adjust**()

Function Description: To display the window for adjusting the touch screen.

Return Values: 1 Successful
0 Failed

Parameters:

Example: touch_adjust()

20.12.2 hmi_sys_set_wnd

Original Function: int **hmi_sys_set_wnd**()

Function Description: To display the **System Setting** window.

Return Values: 1 Successful
0 Failed

Parameters:

Example: hmi_sys_set_wnd()

20.12.3 hmi_sys_set_wnd

Original Function: int **sys_set_time_wnd**()

Function Description: To display the window for adjusting the time.

Return Values: 1 Successful
0 Failed

Parameters:

Example: sys_set_time_wnd()

20.13 Other System Functions

20.13.1 sys_shutdown

Original Function: int **sys_shutdown**(**int** *type*)

Function Description: To shut down the system.

Return Values: 1 Successful
0 Failed

Parameters: *type*: 2-Shut down the system; 3-Shut down and then restart the system.

Example: sys_shutdown(2)

20.13.2 prog_upgrade

Original Function: int **prog_upgrade**()

Function Description: To upgrade the system or the user program through the thumb drive.

Return Values: 1 Successful

0 Failed

Parameters:

Example: prog_upgrade()

20.13.3 sys_sleep

Original Function: void **sys_sleep**(**unsigned long** *ms*)

Function Description: To put the program to sleep for a period of time.

Return Values:

Parameters: *ms:* Number of milliseconds for putting the program to sleep.

Example: sys_sleep(100)