Automation Studio Basis

TM210








Perfection in Automation

www.br-automation.com

Requirements

Training modules:	TM200 – B&R Company Presentation
	TM201 – The B&R Product Palette
Software:	Automation Studio 3
	Automation Runtime 2.90
Hardware:	None

Table of contents

1. INTRODUCTION

Automation Studio is a programming environment for the B&R automation components, which include the controller, motion control and visualization. The clear structure of projects and the ability to manage a wide range of configurations and machine variations makes working in teams that much easier.

In addition to a large number of diagnostics tools, the user is provided with different programming languages and editors. The use of standard libraries provided by B&R and IEC standards that are integrated in the system enables a highly efficient workflow.

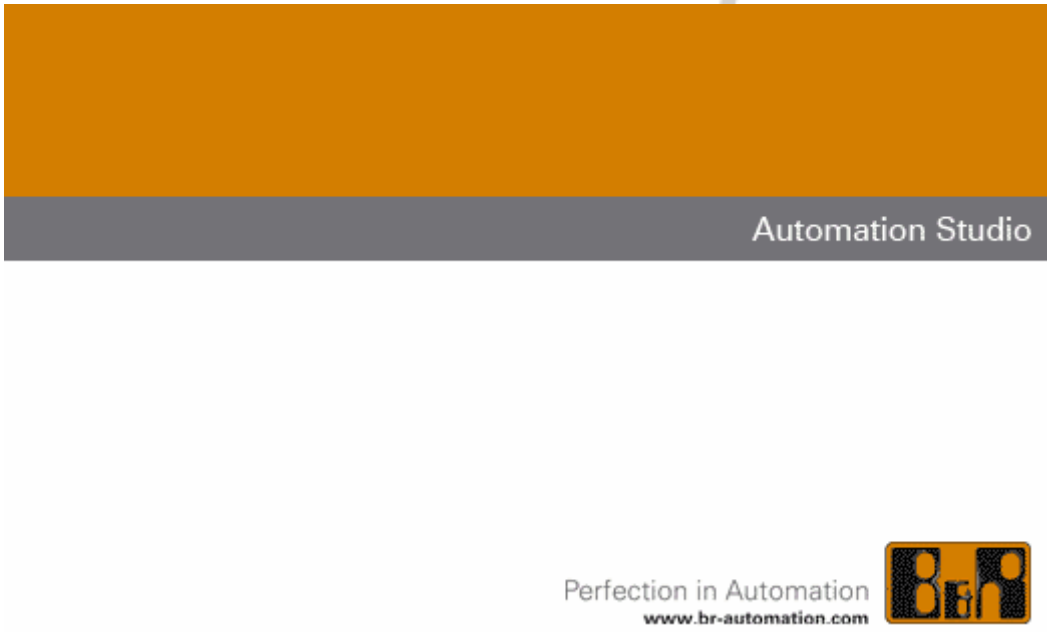


Fig. 1 The Automation Studio splash screen

This training module will use examples – with the aid of the extensive Automation Studio help system – to demonstrate how to use the great many tools available in Automation Studio.

1.1 Objectives

Participants will learn how to work with Automation Studio using examples that outline how a typical project works.

You will master the online help system and its navigation.

You will master data types and variables as well as their declaration.

You will get an overview of the different programming languages as well as the possibilities available for each.

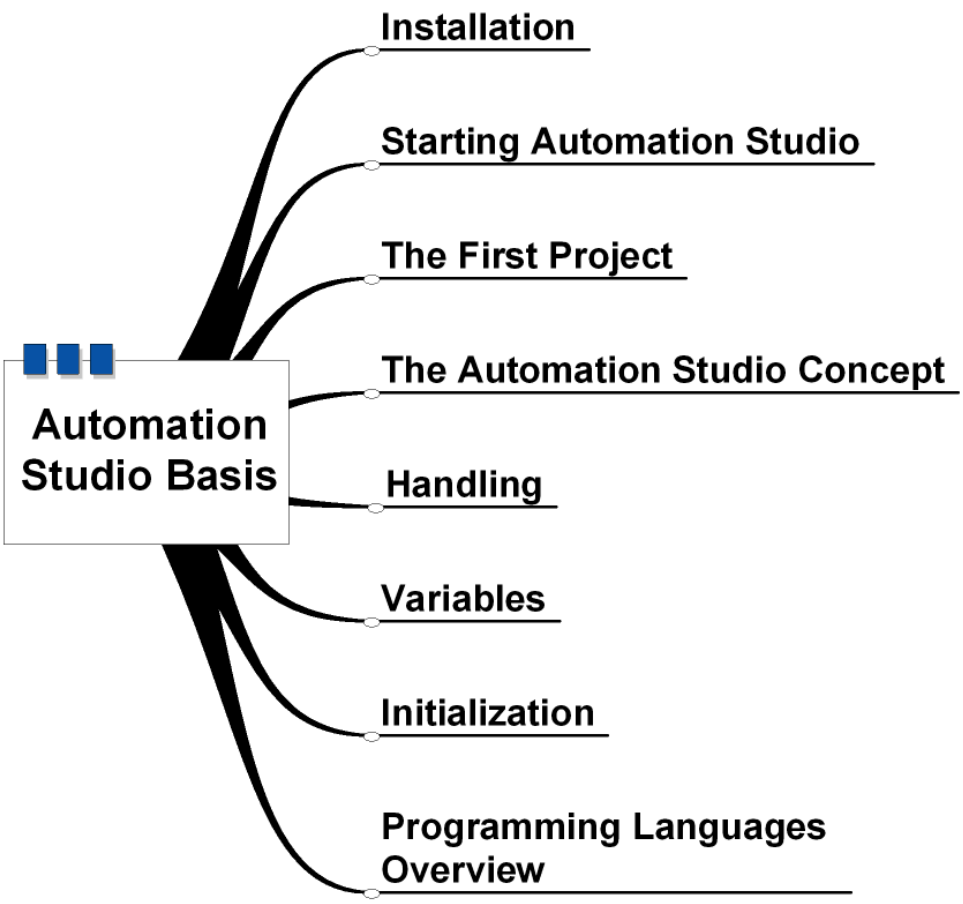


Fig. 2 Overview

2. INSTALLATION

This part of the training module will cover the Automation Studio installation. The following points will explain which actions are needed to select the necessary components.

2.1 Installation wizards

Your Automation Studio installation is started by the **Autorun** command after inserting the CD ROM or by running "**Install.exe**" in Explorer.

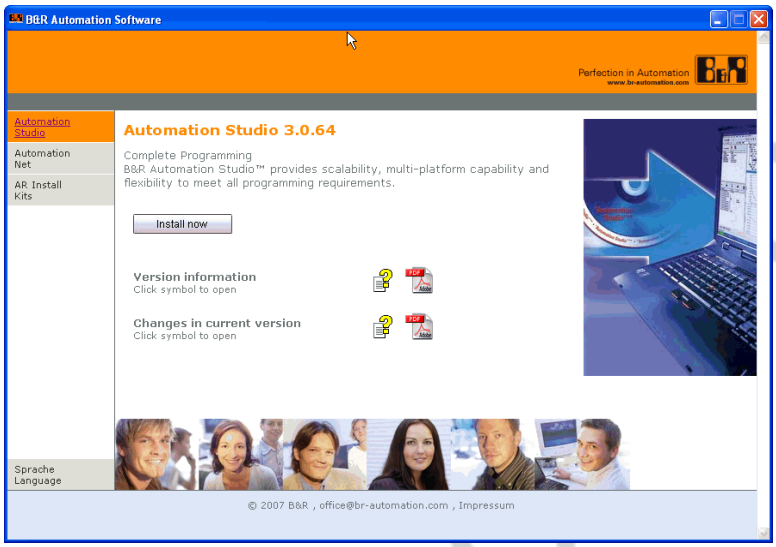


Fig. 3: Automation Studio installation: Selecting the desired language

The installation wizard will guide your through the entire installation after selecting the desired language.

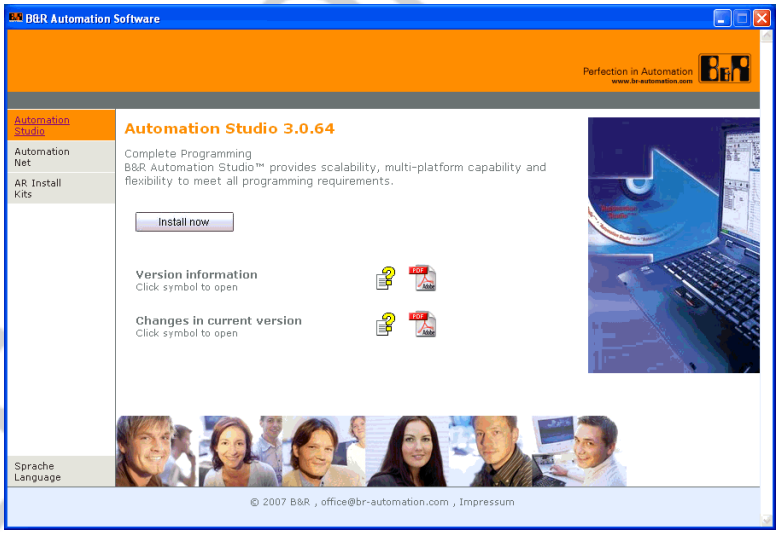


Fig. 4: Automation Studio installation: Starting the installation setup

2.2 Licensing

A dongle is needed to register Automation Studio. This is connected to the parallel interface or to a USB port on your PC.

2.3 Directory structure

After Automation Studio is installed, the following folders will be added underneath the target directory you specified during installation:

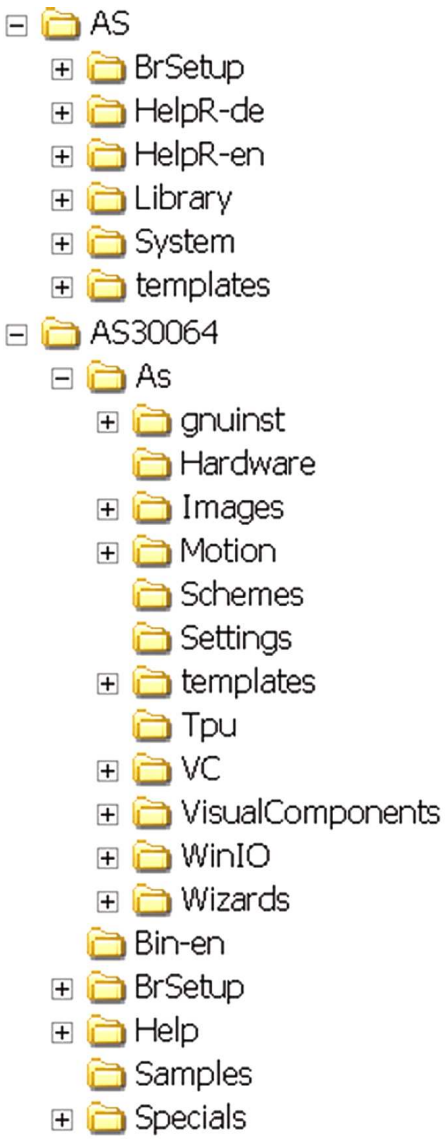


Fig. 5 Directory structure of an Automation Studio installation

3. STARTING AUTOMATION STUDIO

Installation creates an entry for Automation Studio in the Start menu. Automation Studio can now be launched from the Start menu.

Automation Studio can be started multiple times.

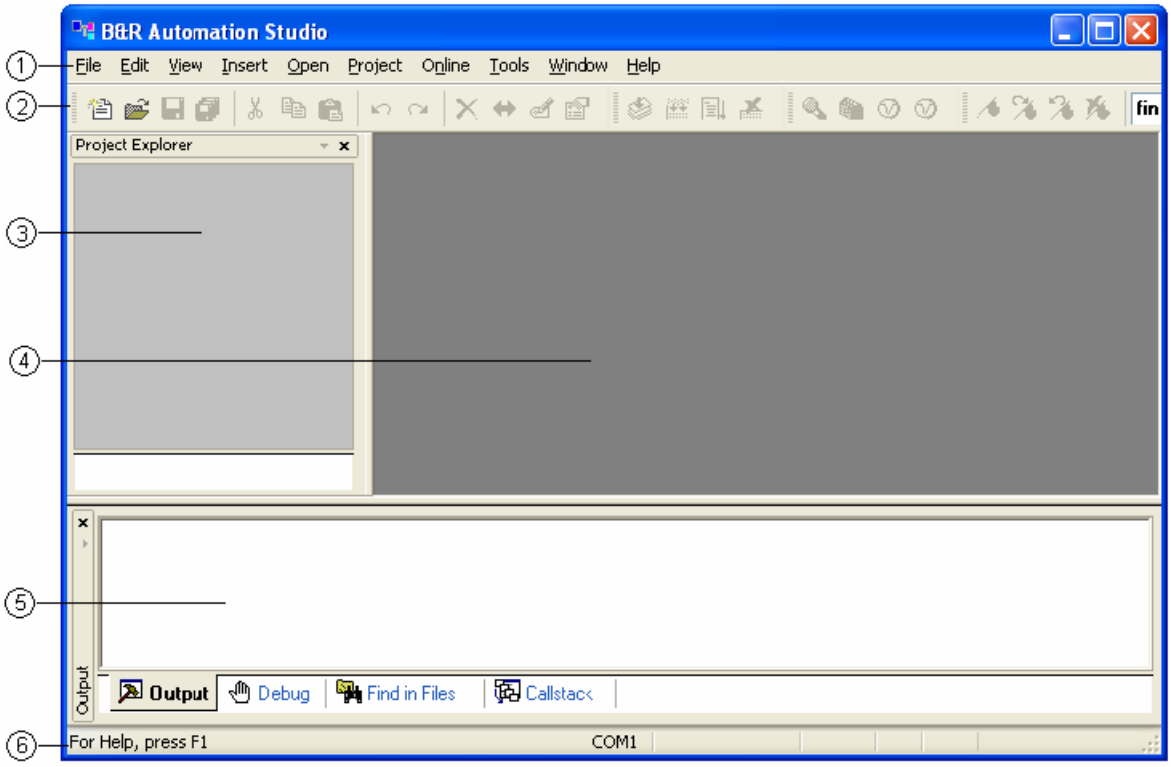


Fig. 6 Automation Studio user interface

The B&R Automation Studio user interface shown above is displayed once the program has been started.

This interface consists of the following elements:

1. Main menu
   The main menu in B&R Automation Studio provides access to all available functions.

2. Toolbars
   Contain buttons that provide fast access to a wide selection of commands and functions. If the mouse is placed over a button, an explanatory text (tooltip) is shown. Toolbars can be shown or hidden using the **View** / **Toolbars** menu item.

3. Project Explorer
   When a project is open, this area shows three different tabs that provide different views of the project and allow it to be structured.

4. Workspace
   This is where the window for an open project is shown. The project window can either be maximized to fit into this area or sized accordingly. This area can also be shown as a workbook.

5. Output window
   The output window is located at the bottom of the program window. It is used to display compiler and debugger messages, etc. In addition, it is where the search results for the "Find in Files" function are output

6. Status bar
   The status bar at the bottom of the window displays the following information:
   - Brief help about menu commands or toolbar icons
   - Brief information concerning editing procedures
   - Status of the online connection between the programming device and the target system
   - Status data for the currently active window

More in-depth information about editors and how they are used – as well as the philosophy behind an application – will be given in the next few sections.

4. THE FIRST FROJECT

In this section, we will be creating a new project, writing a program, and transferring everything to the target system.
The individual steps for each of these will be explained in detail.

Step-by-step procedure:

- Creating a new project
- Creating a program
- Compiling a project
- Transferring the project
- Testing the program sequence

4.1 Creating new projects

To create a new project with Automation Studio, use the **File: New project...** menu item.

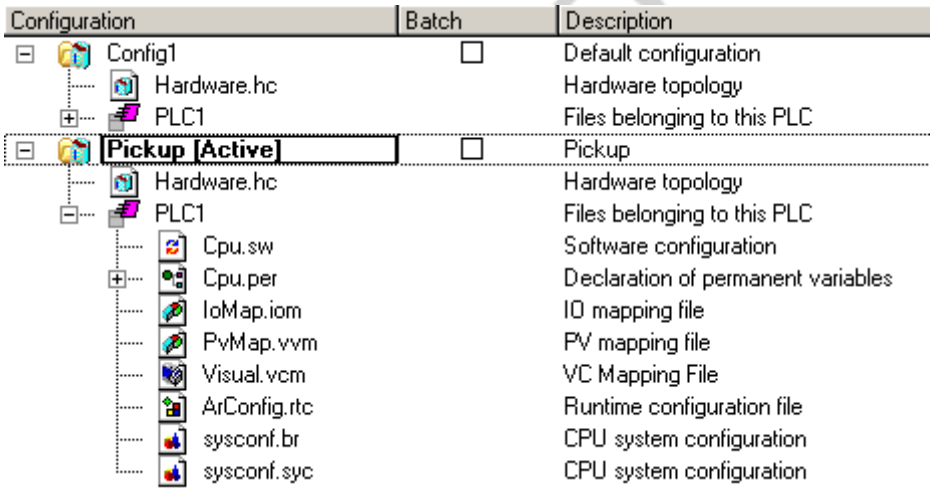


Fig. 7 Creating a new project

The New Project Wizard helps you to complete this task.

The following settings need to be made:

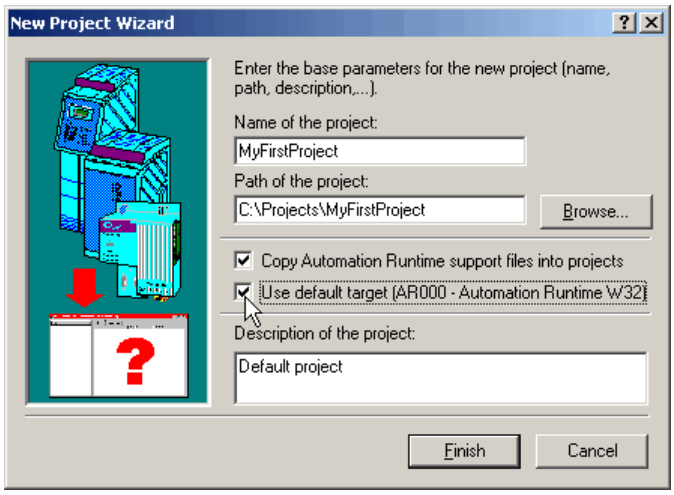


Fig. 8 Creating a new project

- Give your project a name (e.g. MyFirstProject).
- Select the path where your project will be stored (e.g. C:\Projects).
- **Copy Automation Studio Runtime support files into projects** means that operating system files will be stored in the project.
- Select the option **Use default target (AR000 – Automation Runtime W32)**, to use the simulation in the new project.
- Enter a short description of the project.

Click on **Finish** once you have made all of the settings mentioned above.

Now you'll see the following view:

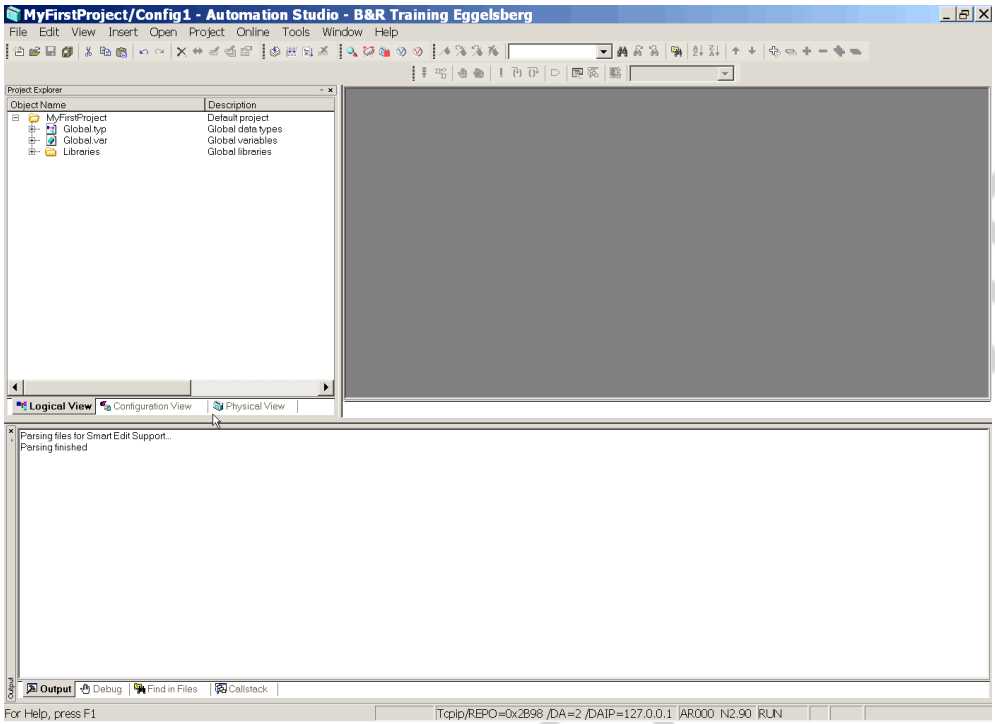


Fig. 9 New project

4.2 Creating a program

The following steps are necessary to insert a ladder diagram into the project:

- Inserting a ladder diagram program
- Declaring variables
- Programming the ladder diagram

4.2.1 Inserting a ladder diagram program

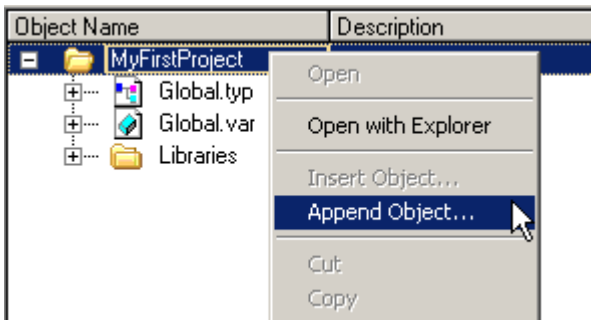


Fig. 10 Adding an object

The wizard for inserting an object is opened in the **shortcut menu: Append Object**.

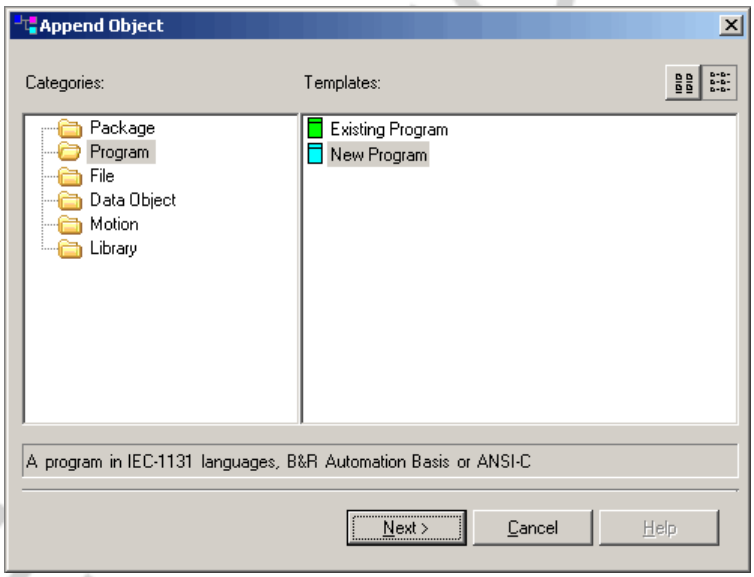


Fig. 11 Selecting the object type

Select **Categories - Program** and then choose **New Program**. Clicking on **Next** will continue with the wizard.

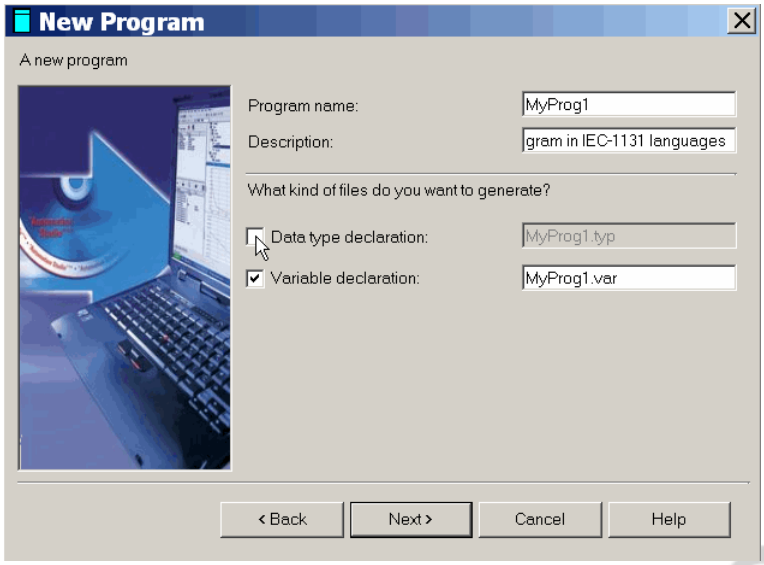


Fig. 12 Object properties 1

Enter a **Program name** (e.g. MyProg1). A short description can also be entered in the **Description** field. The **Data type declaration** is not needed now for the purpose of this example and can therefore be disabled.

Selecting **Variable declaration** generates a file for the variable declaration of local variables for the new program. Continue with the wizard by clicking on **Next**.

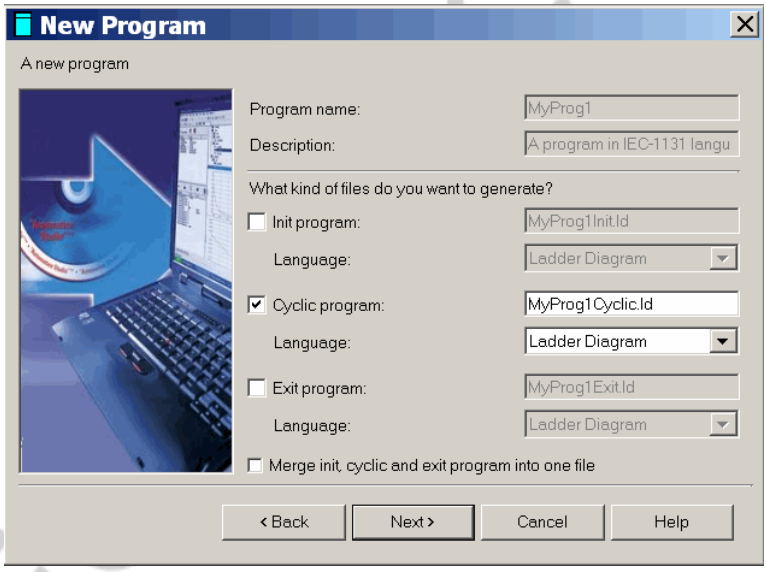


Fig. 13 Object properties 2

Deselect the option **Init program** and select **Ladder Diagram** as **Language**.

Clicking on **Next** will continue with the wizard.

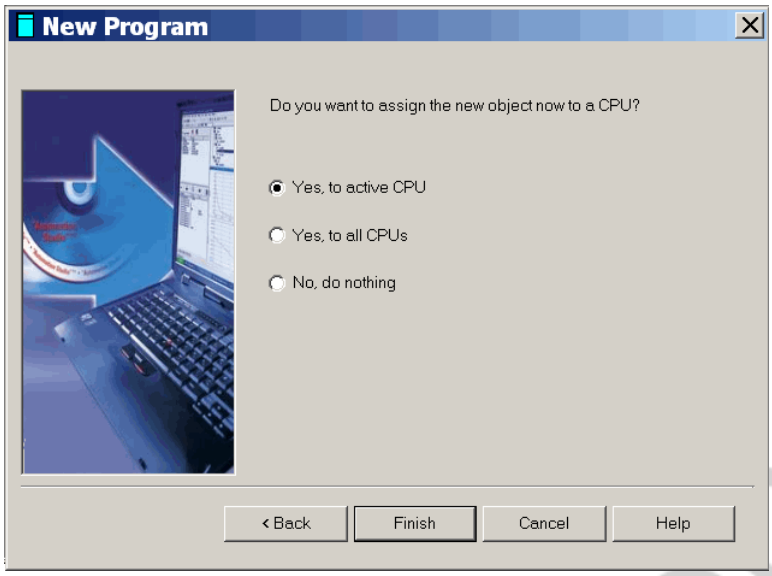


Fig. 14 Hardware-related object setting

Assigning the object to the CPU (**Yes to active CPU**) means that the created program is automatically assigned the software configuration.

Click on **Finish** to close the wizard.

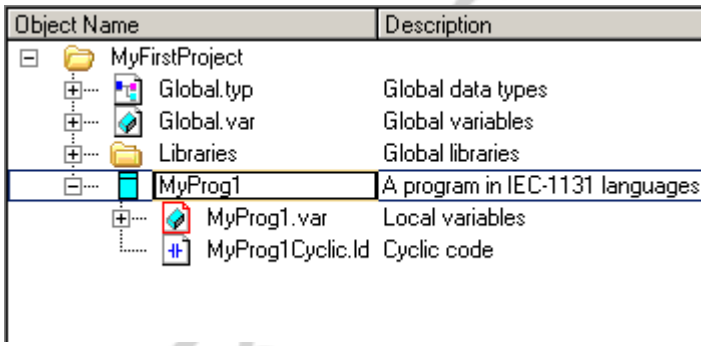


Fig. 15 New object in the project explorer

Three items have now been added in the Project Explorer. The variables for the Ladder Diagram will now be created in the next step.

4.2.2 Declaring variables

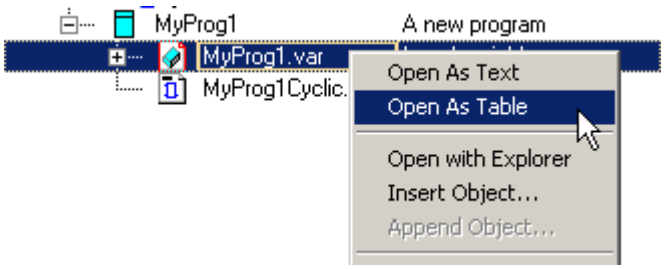


Fig. 16 Opening the declaration window

The variable declaration, for adding new variables, is opened by right-clicking **MyProg1.var** and selecting **Open as Table**.

A blank white area now appears in the right half of the screen. Add a new variable by right-clicking and selecting **Insert Variable**.

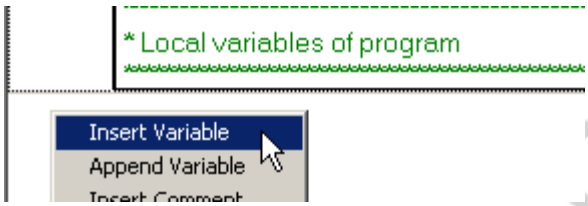


Fig. 17 Inserting a variable

Enter the name of the variable here. Use the name **"diLight"**.

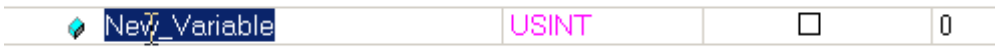


Fig. 18 Setting the variable name

We now just have to set the variable data type. This should be set to the data type **"BOOL"**.

Fig. 19 Setting the variable type

A variable's data type can be changed by entering the data type name directly in the **Type** column or by **double-clicking** on the ... icon.

Repeat this procedure on your own for the variable "**doLight**" and you will arrive at the following variable declaration.

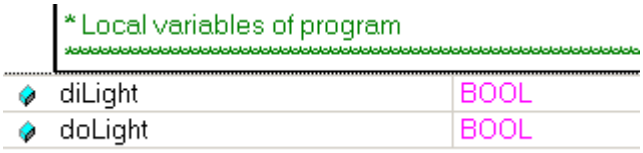


Fig. 20 Variables for MyProg1

Now save your changes using the save icon  .

The variables that we want to use in our ladder diagram are now defined. That means that they can now be used in ladder diagram.

4.2.3 Programming the ladder diagram

The following actions are necessary to program the ladder diagram.

Open the ladder diagram editor by double-clicking on **MyProg1Cyclic.ld** in the Project Explorer.

Fig. 21 Editing the Ladder Diagram

The ladder diagram editor appears on the right side of the screen.

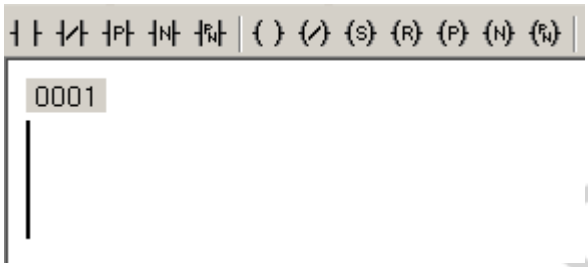


Fig. 22 Ladder Diagram editor

The cursor (shown as a cross) blinks in the editor.
Now click on the icon to insert a normally open contact. A field with a blinking cursor appears over the contact.
Press the **space bar**.

This opens the following window.

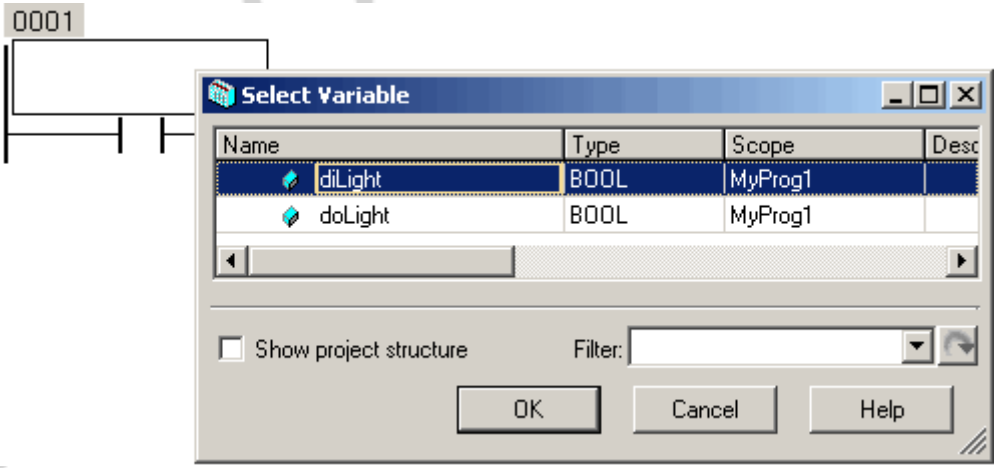


Fig. 23 Connecting a normally open contact

Click OK to connect the variable **diLight** to the contact.

Insert a coil by clicking on the { } icon. Repeat the same procedure as earlier with the variable **doLight** for the normally open contact.

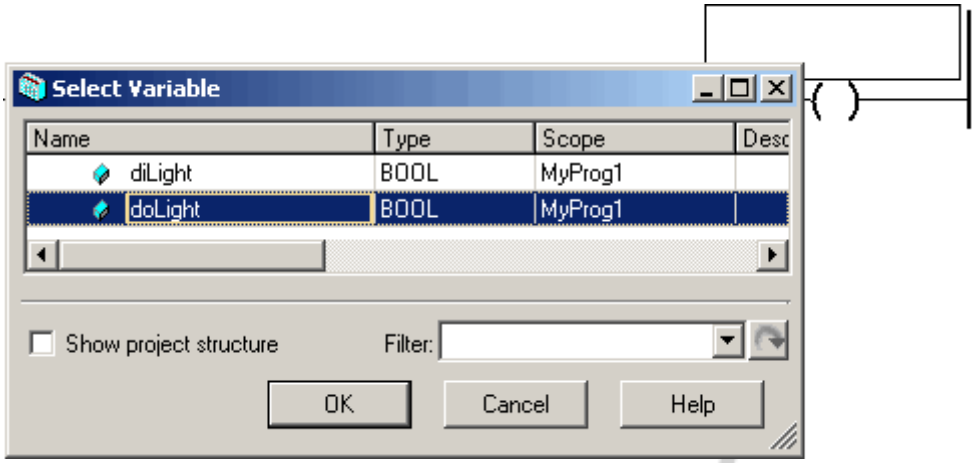


Fig. 24 Connecting a coil

Your ladder diagram now looks like this:

Fig. 25 Finished ladder diagram

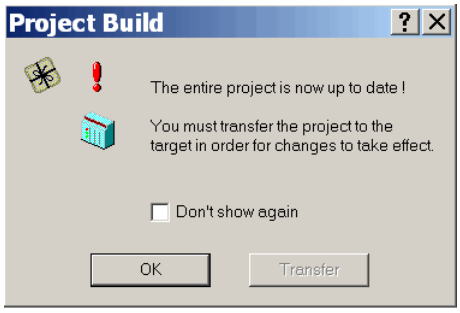
Save your work with the  icon.

4.3 Compiling a project

We have now successfully finished creating the project and programming the ladder diagram. We will now compile our project. In this step, we are going to learn about any errors that might exist in our project or program.

Compile your project using the  icon.

The following window appears after the project has been successfully compiled:



This window informs us that the entire project has been compiled is now up to date. It is also contains the note informing us that the project can now be transferred to the target system.

In the following section we will learn how to define a target system for the project.

4.4 Transferring the project

We will use Simulations Runtime AR000 as target system. This allows us to test program sections during cyclic operation, even without any physical hardware.

4.4.1 Starting the emulator

The AR000 must first be started before it can be used.

To do this, click on the **Tools**: **AR000** menu item.

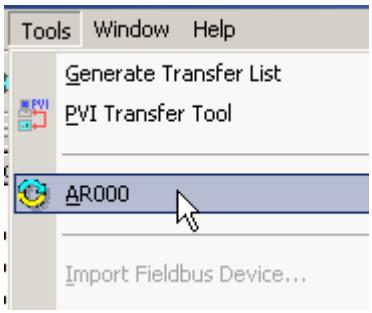


Fig. 26 Starting the emulator

This opens the following dialog box:

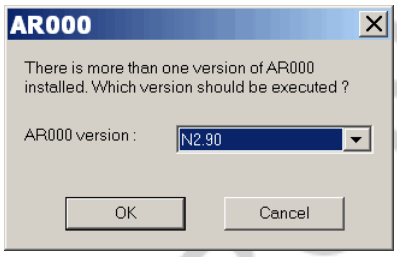


Fig. 27: Choosing the AR000 version

Here you can choose which AR000 version to use. We will use the default version and confirm our selection by clicking **OK**.

The simulation is then started and the following window appears.

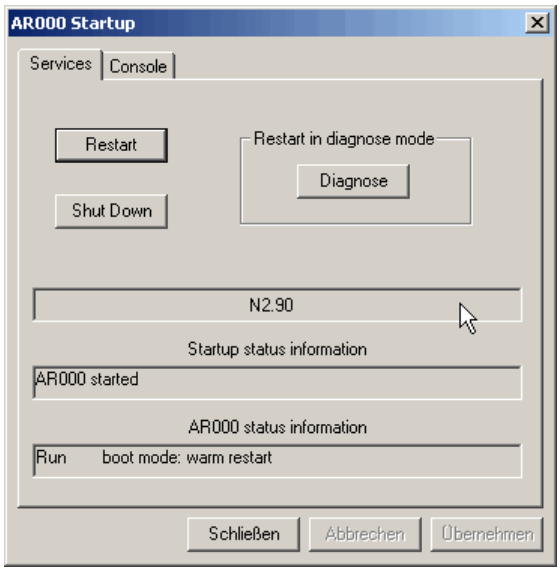


Fig. 28 AR000 runtime emulation

The AR000 has now been successfully started and can now be used as target system for testing our first project.

Before we can transfer the project, we must first establish a connection to the target system.

4.4.2 Creating a connection

In order to be able to work with a controller, it's necessary to establish a connection to it. This connection is needed for transferring the project to the target system.

The following describes how to specify the type of connection.

Open the menu item **Online:Settings.**

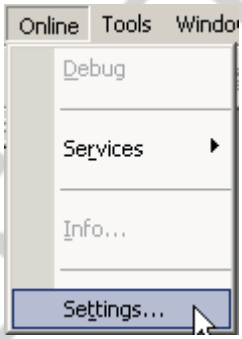


Fig. 29 Connection menu

This opens the following window:

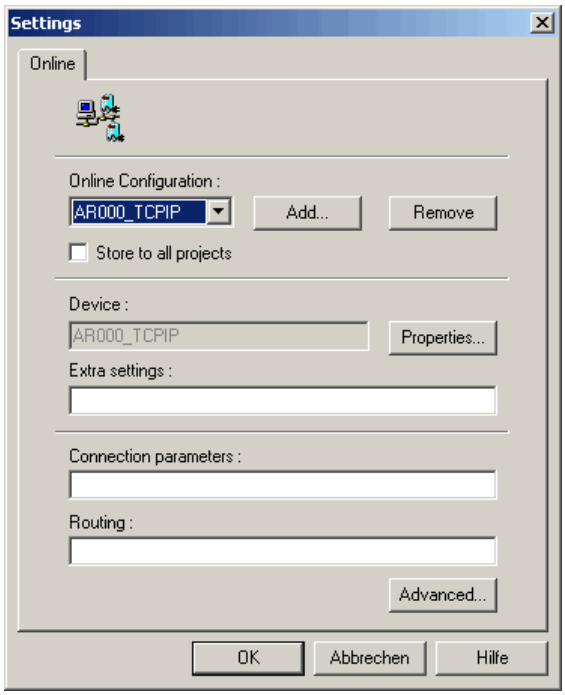


Fig. 30 Connection settings

Pre-defined connection configurations can be selected from this window. Select **AR000_TCPIP** from the **Online Configuration** combo box.

Accept the settings by clicking on **OK**.

As we can see on the status bar (online connection status "**RUN**"), a connection has been established to the target system (AR000).

Fig. 31: Online connection status

4.4.3 Transferring a project

To transfer the ladder diagram to the target system, click on the  icon.

The following dialog box is shown:

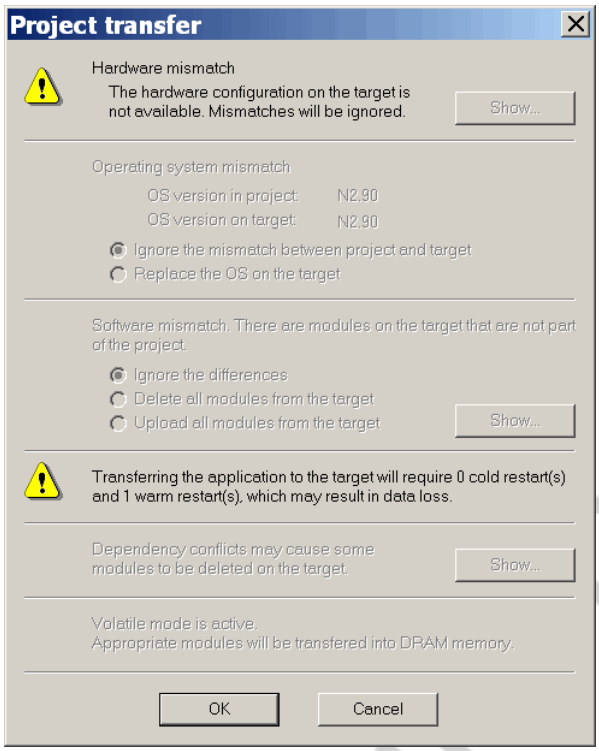


Fig. 32 Dialog box before project transfer

This dialog box will inform you if there are already modules on your CPU. **Delete all** gets rid of all of the objects on the target system that don't belong to your project.

You are also informed whether the system requies a **warm or cold restart**. Operating system parameters can only be applied after the system is restarted.

Successful transfer of the project is confirmed with the following dialog box.

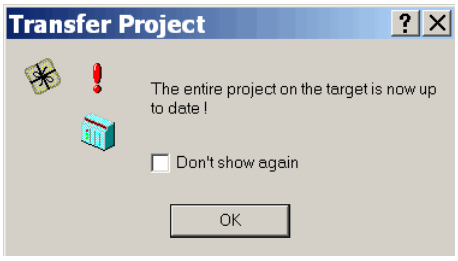


Fig. 33: Project successfully transferred

The ladder diagram program is now running on the target system. Now we can test the program to make sure that it's working correctly.

4.5 Testing the program sequence

To enable monitor mode for the ladder diagram, click on the  icon.

The following view then appears.

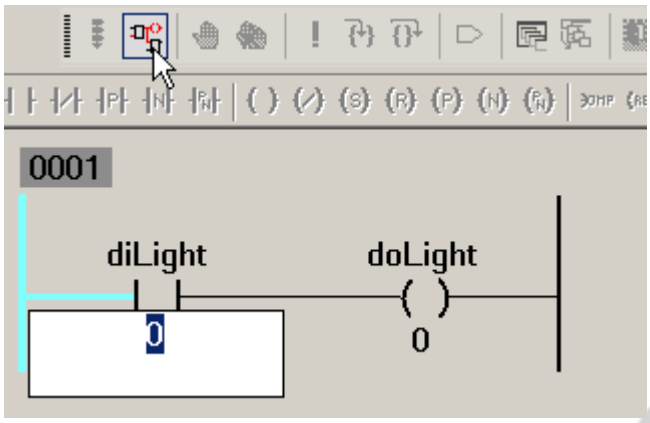


Fig. 34: Monitor view of the Ladder Diagram with signal flow display

Now you can check whether the value changed at the input produces the desired result at the output.

The signal flow display can be turned on with the  icon.

The signal flow is shown by coloring both the lines and the symbols. In addition to the values in the program, this provides another way to carry out diagnostics.

We have now created an empty project with Automation Studio in addition to establishing a connection to the target system.
After a new program was inserted, we added new variables to it. The variables we created were connected in the ladder diagram editor. After the transfer, we used the ladder diagram monitor to check the cause and effect of entering different values.

5. THE AUTOMATION STUDIO CONCEPT

There are several windows and menus in Automation Studio that can be accessed when a project is opened. The relationship of these parts must be explained and understood.

5.1 Using the AS online help system

The Automation Studio online help is a reference guide for working with Automation Studio.
It contains all the information you need about operating Automation Studio, its editors, and its user interface. Hardware documentation for modules is also included.

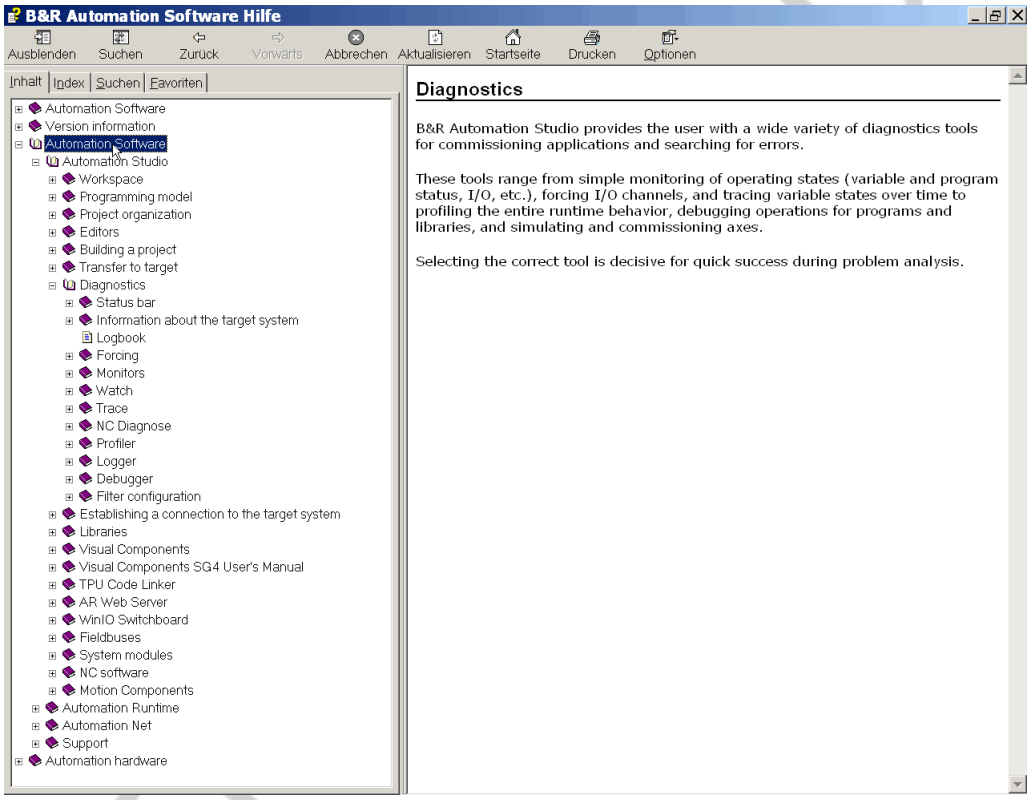


Fig. 35 Excerpt from the online help system

We recommend getting as much information as possible from the Automation Studio online help. Like Automation Studio itself, this documentation is constantly being revised and improved.

Pressing the **F1** key opens up the help topic for the element that is selected in Automation Studio. You can also use the search function in the help to find information about a certain topic.

Note:

Using the **Tools:Options** menu item, the language of the Automation Studio online help can be set **(German/English)**.

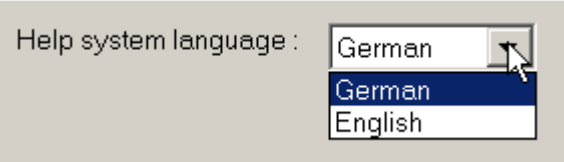


Fig. 36: Setting the language for the Automation Studio online help

Exercise: Using the Online Help system

Look for information about monitor mode in Automation Studio.
Determine which information you can find out about monitor mode.
Open up the online help for "Automation Software – Automation Studio - Diagnosis".

5.2 Basic concept

Creating software is structured in an Automation Studio project using the **machine structure**. This allows a software organization with a clear overview, because a real reference to the programs can be seen.

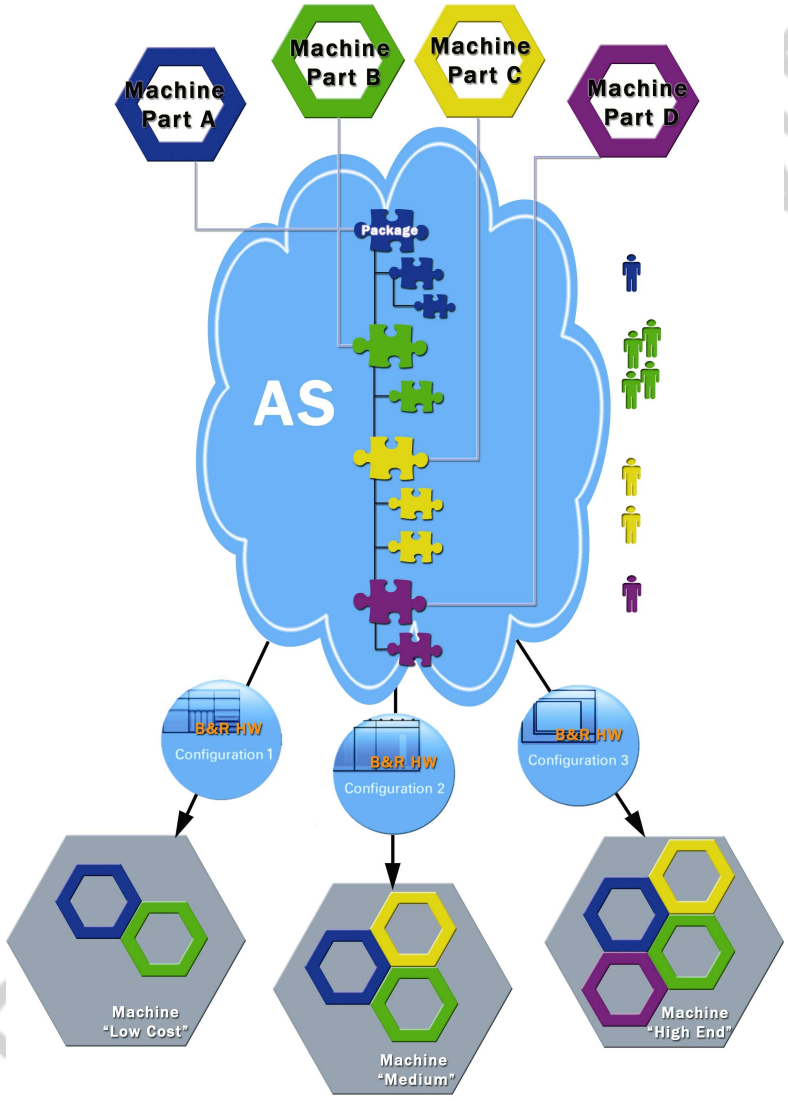


Fig. 37 Structure and principle of Automation Studio

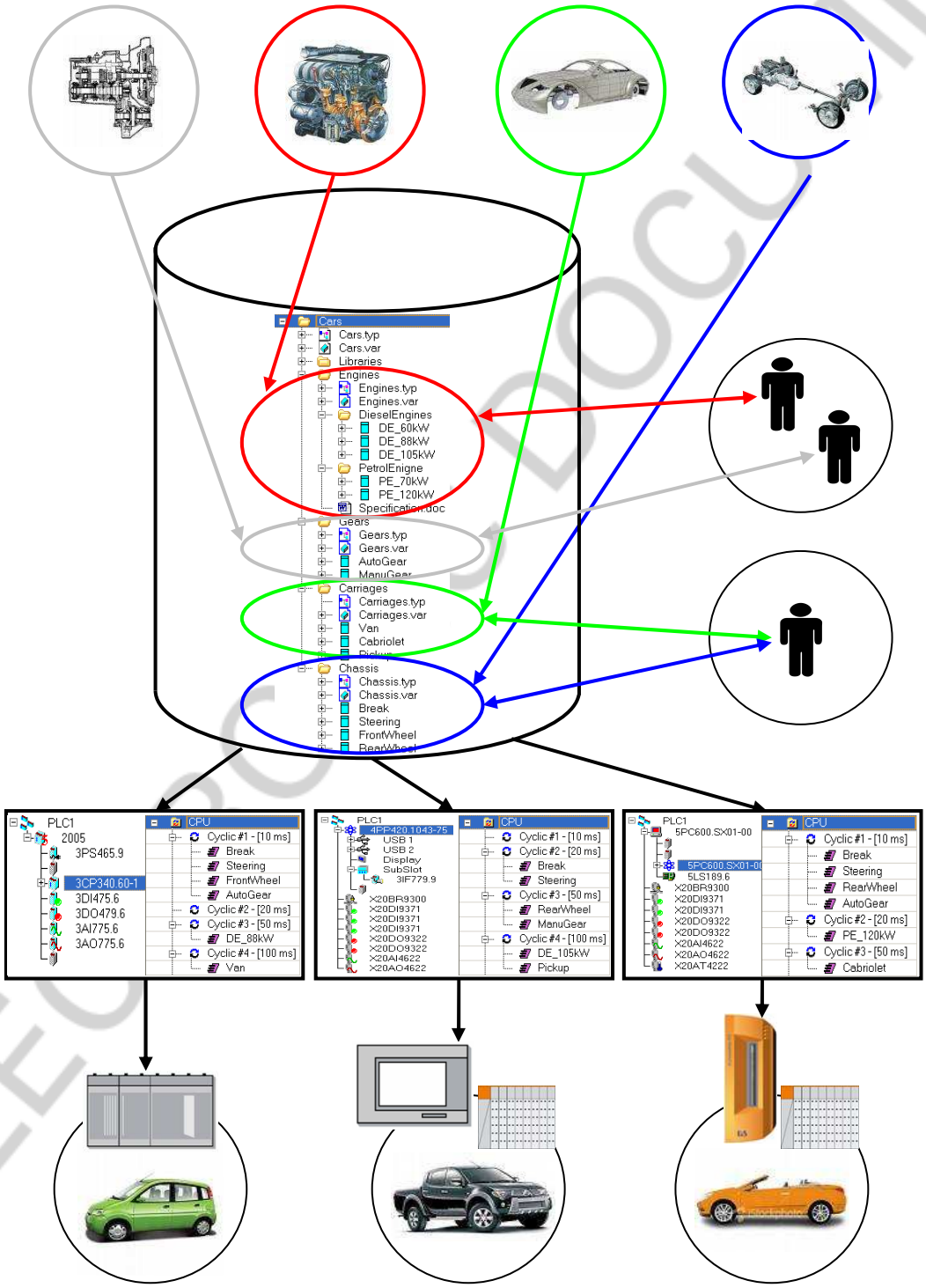
The programmed machine parts can be assigned different configurations. As a result, different delivery states for certain machine types, which vary in the software used and the hardware design, can be managed in a project.

This basic concept creates many configuration possibilities, which will be described in greater detailed for the following views.

5.3 The different views

In this section we will use a concrete example to describe the different views in Automation Studio.

The following image illustrates our example.



Let's assume someone wants to make three different vehicles, a van, a convertible and a pickup. All vehicles have certain features in common, but differ from one another in aspects such as engine, gears, chassis and carriage. Our different machine parts (vehicle parts) should represent these various components.

The four "machine parts", **engine, gearbox, carriage** and **chassis** are represented in the upper section. These are located one layer deeper in the Automation Studio software. These software packages are now used to put together different vehicle models.

Let's take a look at the tree structure with the higher-order name **Cars**. A lower-level hierachy contains three definitions (Cars.typ, Cars.var, Libraries) that pertain to all "car parts".
These are followed by the four sub-trees **Engines, Gears, Carriage** and **Chassis**.

The **Engines** sub-tree consists of definitions (Engines.typ, Engines.var), which are valid for both motor types and the two additional sub-categories, **DieselEngines** and **PetrolEngines** each with their own different performance types.

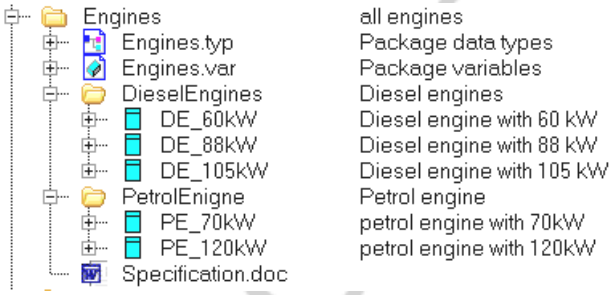


Fig. 38 Engines sub-tree

In the **Gears** sub-tree we find again general definitions (Gears.typ, Gears.var) that are valid for both gear types and sub-categories for the two special gear types **AutoGear** and **ManuGear**.

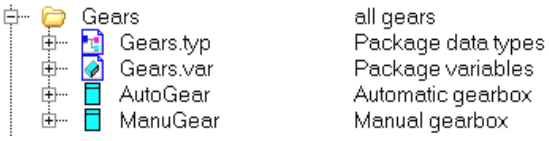


Fig. 39 Gears sub-tree

The **Carriages** sub-tree also contains definitions (Carriages.typ, Carriages.var) that are the same for all contained vehicle carriage types, but also a special function for each carriage type.

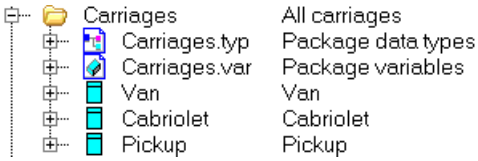


Fig. 40 Carriages sub-tree

The same principle applies to the **Chassis** sub-tree which consists of **Brake**, **Steering**, **Front Wheel Drive** and **Rear Wheel Drive**.

The previous described view / tree structure is called a **"Logical View"** in Automation Studio.

5.3.1 Logical view

All software elements are arranged in the **Logical View** in the form of a tree. The elements in this tree are folders and objects. The folders will also be referred to as packages.

Each package in the logical view represents e.g. the complete software and documentation for a specific machine part. This makes is possible to structure a project using the machine structure.

Each machine part can be configured individually. Any data type can be added to the respective package for the documentation.

Packages can be imported/exported individually, which makes it possible for each member of a team to work on one package or machine part.

There is no reference to hardware in this view. The focus here is on the structuring and arrangement of program sections.

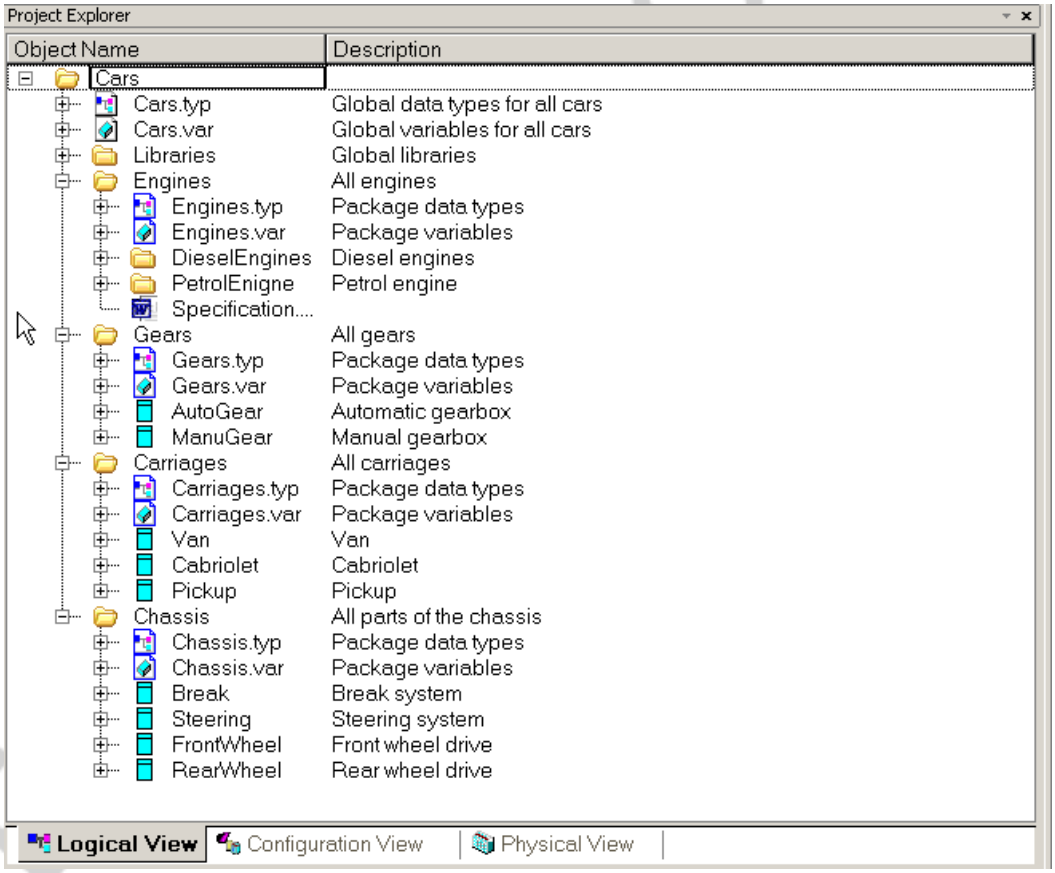


Fig. 41 Logical view

Let's go back to our **"car example"**. As we determined earlier, we now have all of the software components and definitions available that are needed to start making different vehicle models.

But how can we now allocate parts to a single vehicle from all of these sub-components?
Automotion Studio contains a **Configuration View** for this very purpose which makes it possible to create and manage different configurations.

5.3.2 Configuration view

The different configurations are managed in this view. Managing in this case stands for creating, changing, deleting and activating a configuration. Each of these configurations contains hardware and software.
Only one configuration may be active at a time. The active configuration is shown in **bold** and contains the add-on **[Active]**.
Depending on which configuration is activated, the hardware selected for the configuration is displayed in the **Physical View**.

All settings for the target system can be made in the **Configuration View**.

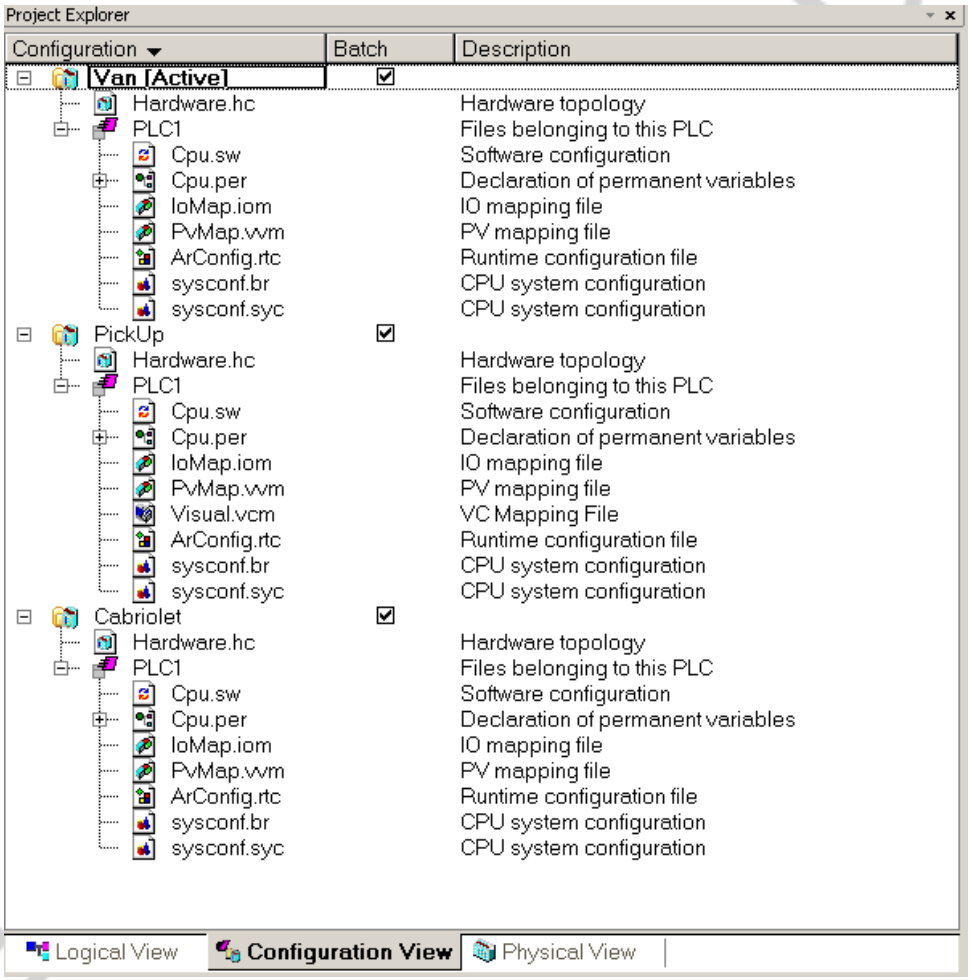


Fig. 42 Configuration view

5.3.3 Physical view

The hardware tree of the configuration selected in the **Configuration View** is shown in the **Physical View**.
The hardware for the corrersponding configuration can be defined and adjusted here.

The following actions can be performed here:

- Set the interface cards (e.g. for the online connection)
- Set I/O modules
- Assign I/O data points
- Open the software configuration

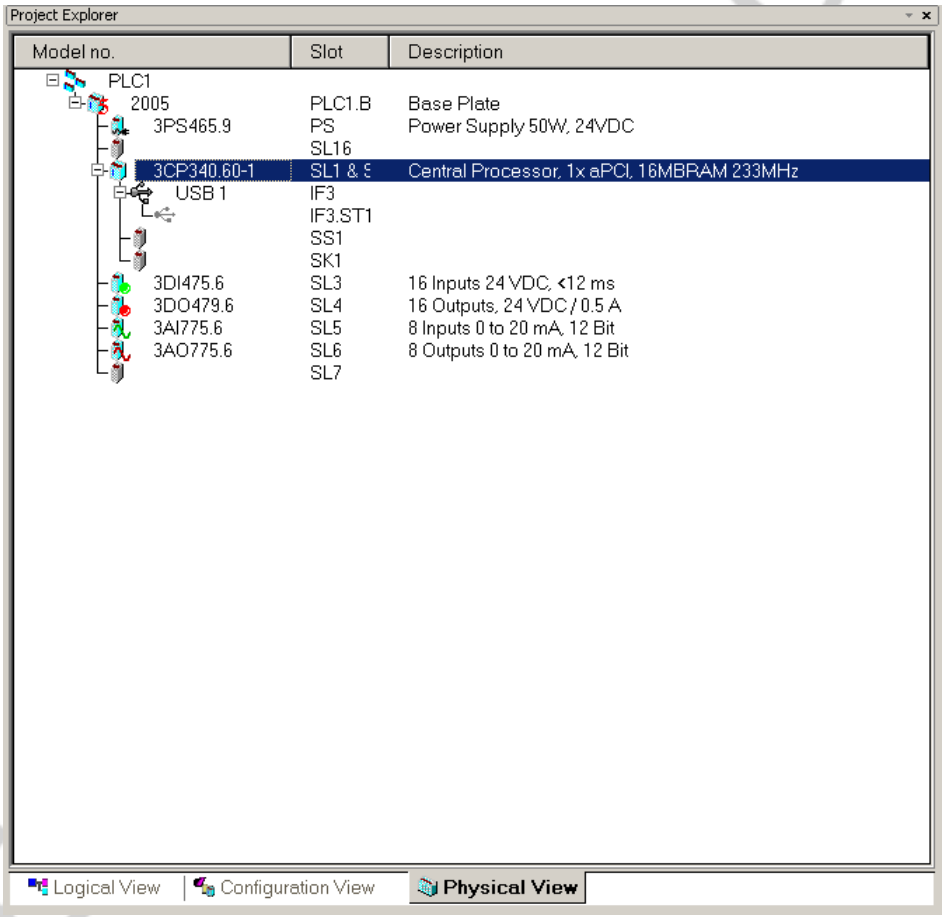


Fig. 43 Physical view

5.3.4 Output window

The output window displays warnings with green text, errors with red text and information with normal text. This is important information when solving errors during compilation.

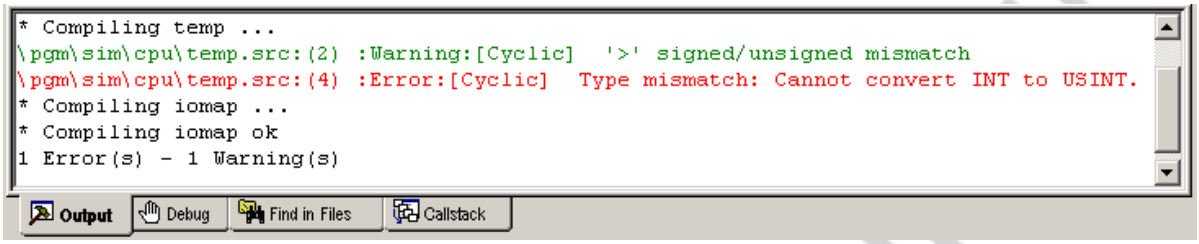


Fig. 44: Automation Studio output window

The output window combines the following:

- Compiler warning and error messages
  Double-clicking on a message brings you to the program line that caused the error.
- Progress and status display when downloading a project
- Message display when inserting and deleting objects in the project or on the target system
- Output window for debugger messages

Output of results for the "Find in Files" function that searches all the files in the project

5.4 Relationship between functionality and task

The logical view shows us the available software components which we can now use for creating our project.

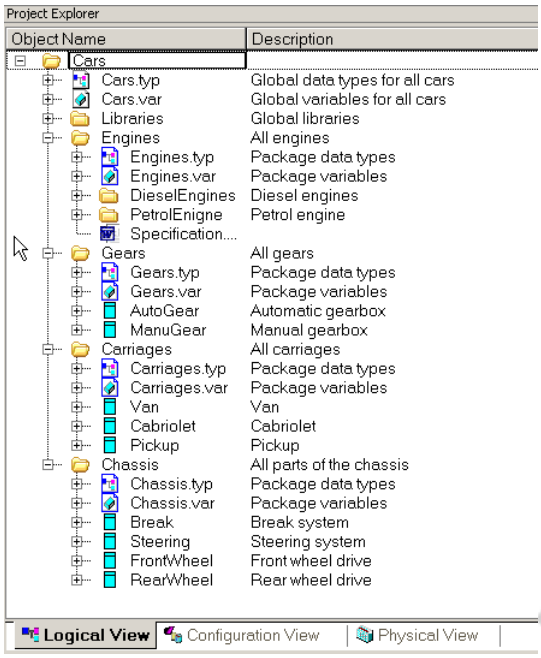
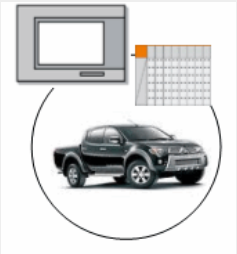


Fig. 45 Logical view

Configuration example - "Pickup"



Now we're going to make the configuration for the pickup in accordance with our "car example".

Solution approach:

- Create a new configuration called "Pickup"
- Add the required hardware
- Open the software configuration
- Assign the software

5.4.1 Creating a new configuration

The **Insert Configuration** shortcut menu in the configuration view can be used to insert a new configuration.

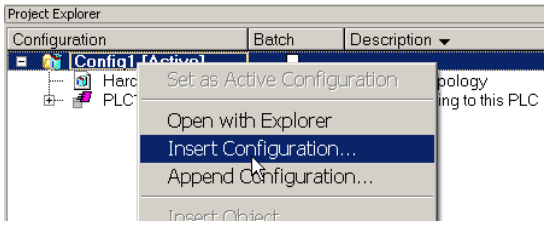


Fig. 46 Inserting a new configuration

The name of the new configuration is defined in the following dialog box. We will choose **Define a new hardware configuration** and continue the setup wizard by clicking on **Next**.

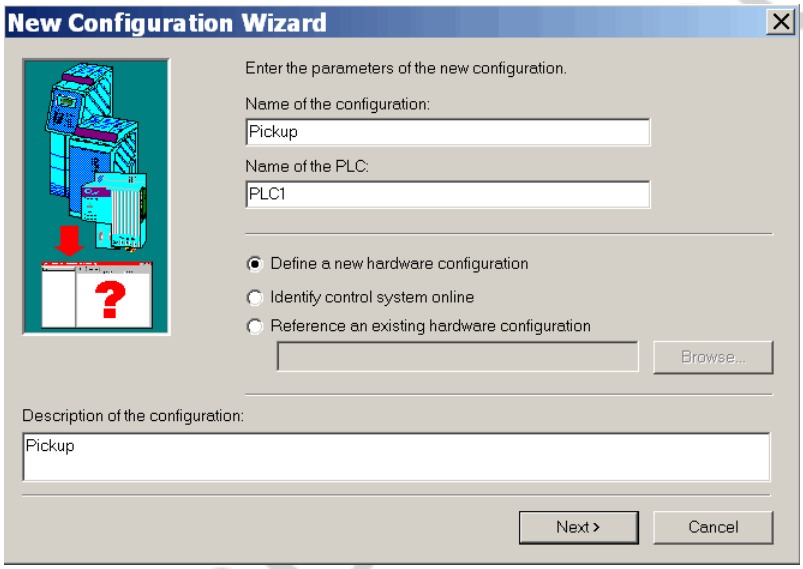


Fig. 47 New configuration – Define settings

We will now select the CPU for the pickup in the following dialog box. In our example we will use a 4PP420.1043-75. Clicking on **Next** will continue with the wizard.

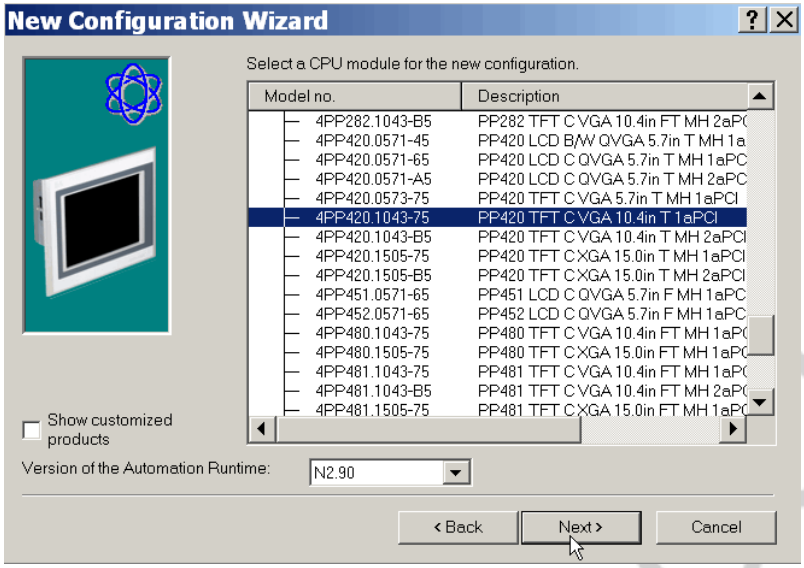


Fig. 48 Selecting the CPU for the new configuration

We will now complete the insertion of the configuration by clicking **Finish**.

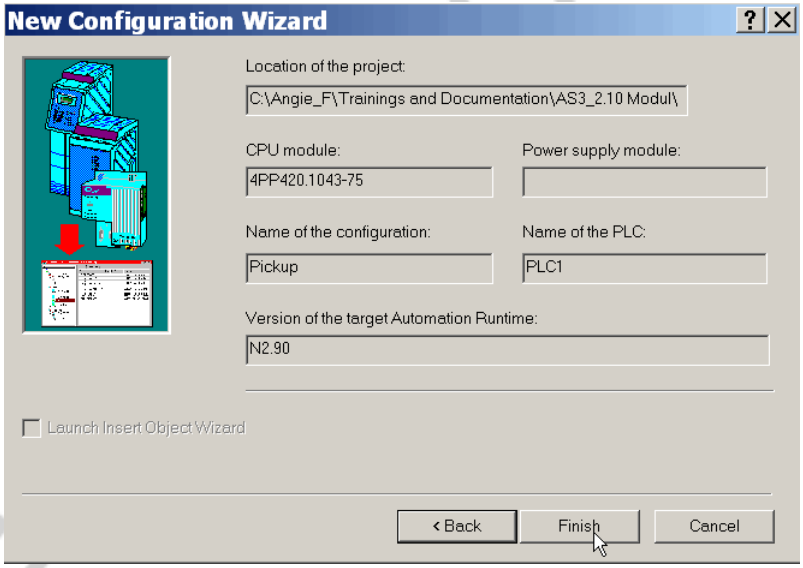


Fig. 49 Finished inserting the new configuration

The newly created configuration "Pickup" has not been successfully inserted. You can change between individual configurations by **double-clicking** on the configuration names.

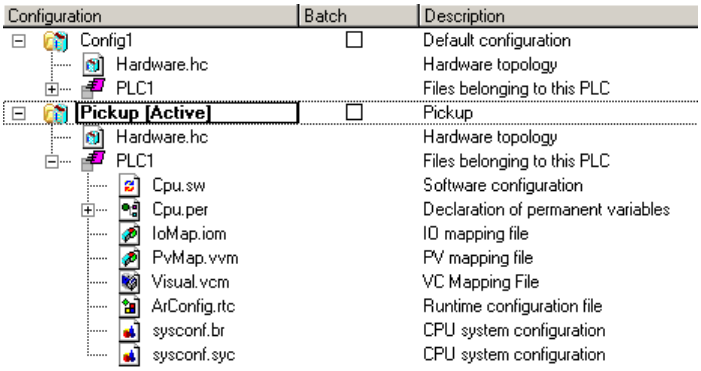


Fig. 50 New configuration "Pickup" successfully created

5.4.2 Adding the required hardware

The corresponding hardware tree of the configuration activated in the **Configuration View** is shown in the **Physical View**. The required hardware can now be entered here for the pickup.

The following hardware should be added:

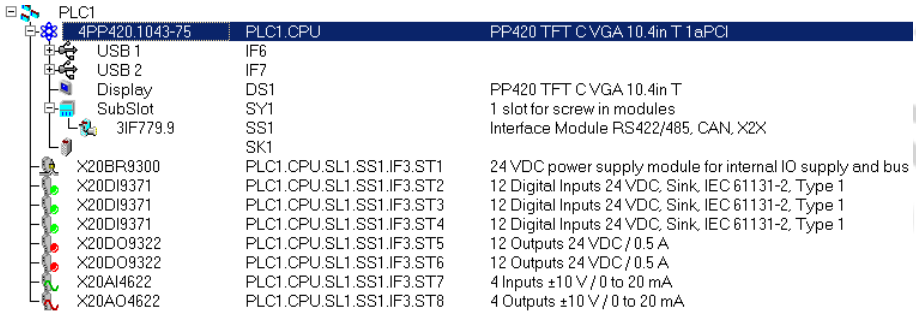


Fig. 51 List of the hardware required for the pickup

The corresponding interface module can be inserted by right-clicking on **SubSlot**.

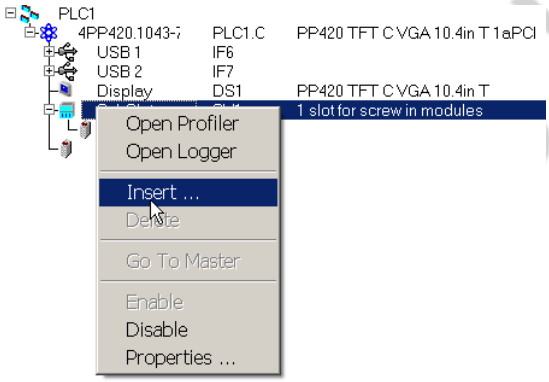


Fig. 52 Inserting the interface card

The desired interface module (with X2X) can be selected here. The selection is completed by clicking on **OK**.

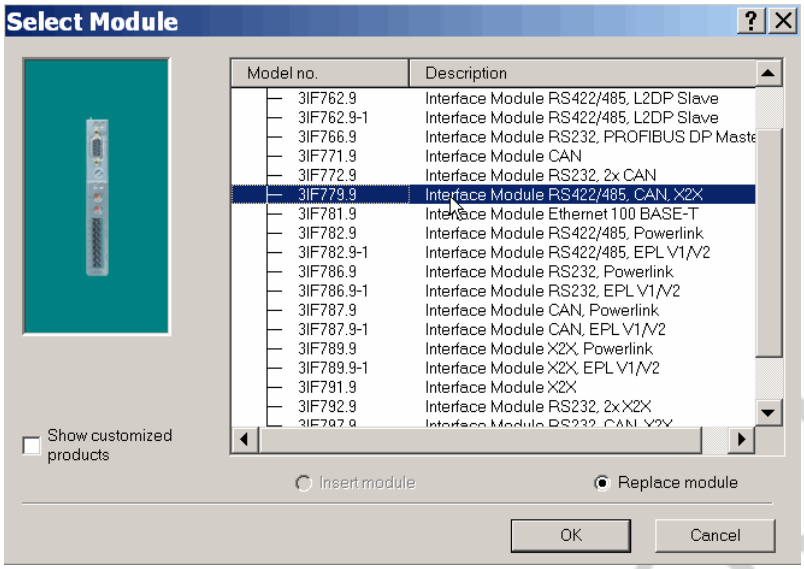


Fig. 53: Selecting the interface module

The configuration area for the I/O connection is opened on the right side of the screen by right-clicking on the interface card and selecting **Open X2X Link**.

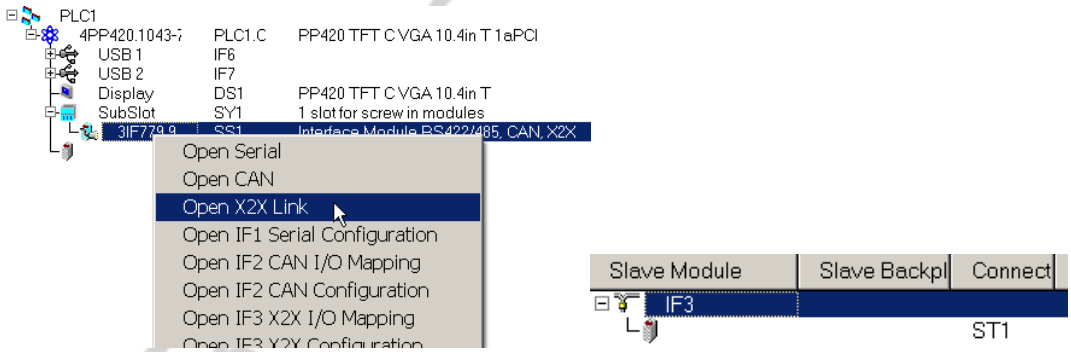


Fig. 54 Opening the X2X connection

All of the required I/O modules can be inserted at the same time by right-clicking the selected interface. This procedure can be repeated as many times needed until all specified I/O modules have been added to the hardware tree.

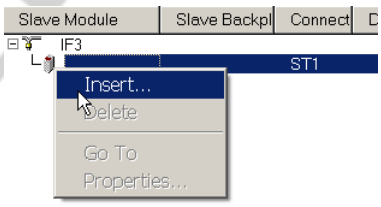


Fig. 55: Inserting I/O modules

5.4.3 Open the software configuration

Now that the hardware has been configured for the pickup, we can get started assigning software.

The software configuration for our pickup is opened by **double-clicking** on the CPU entry in the **Physical View**.

Fig. 56 Opening the software configuration

The corresponding software configuration appears on the right side of the screen. The software elements in the cyclie system are managed here.

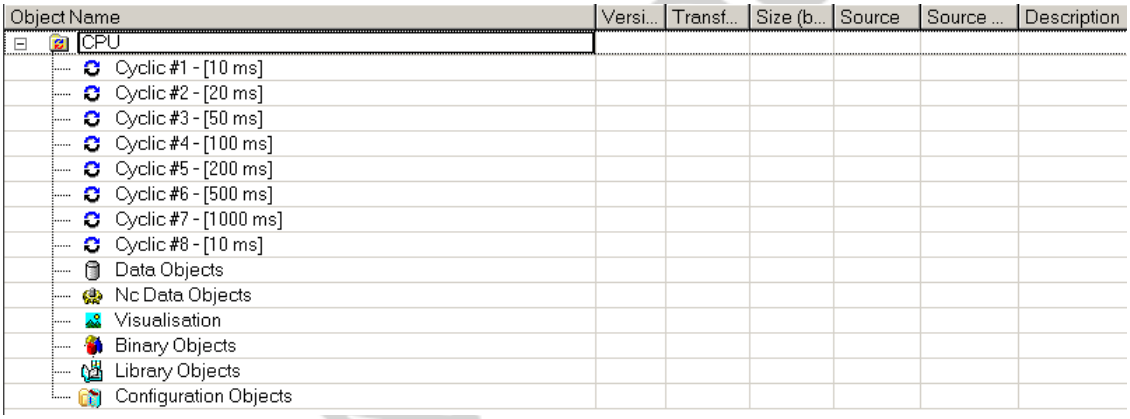


Fig. 57 Software configuration

5.4.4 Assigning the software

Programs from the logical view can be added to the **software configuration** at any time when the software configuration is opened on the right side of the screen and the **logical view** is active on the left side.

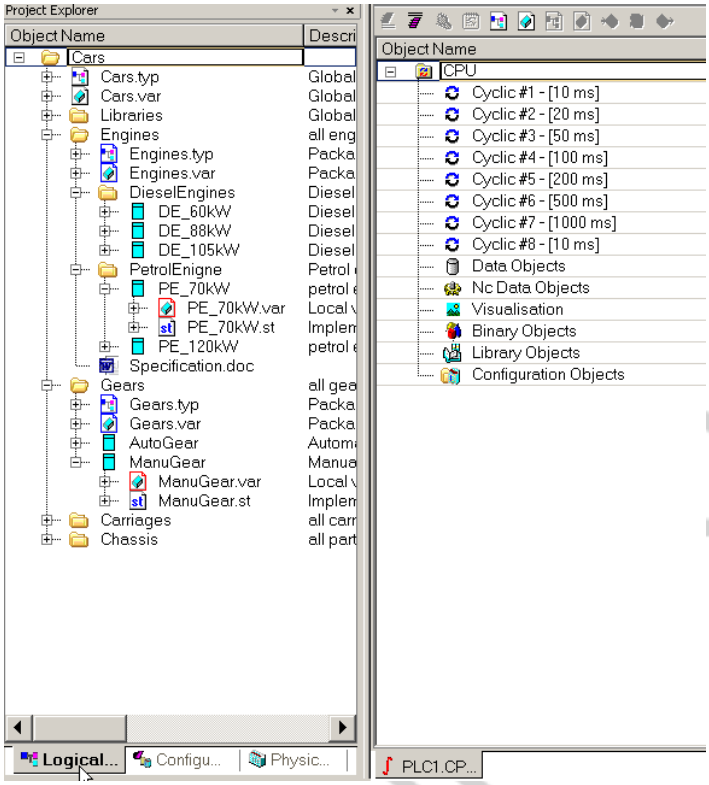


Fig. 58 Relationship between software configuration and logical view

You can assign an object by dragging it from the **Logical View** into the desired task class of the **Software Configuration**.

This is how all software components required for the pickup can be assigned.

Our software configuration then looks like this.

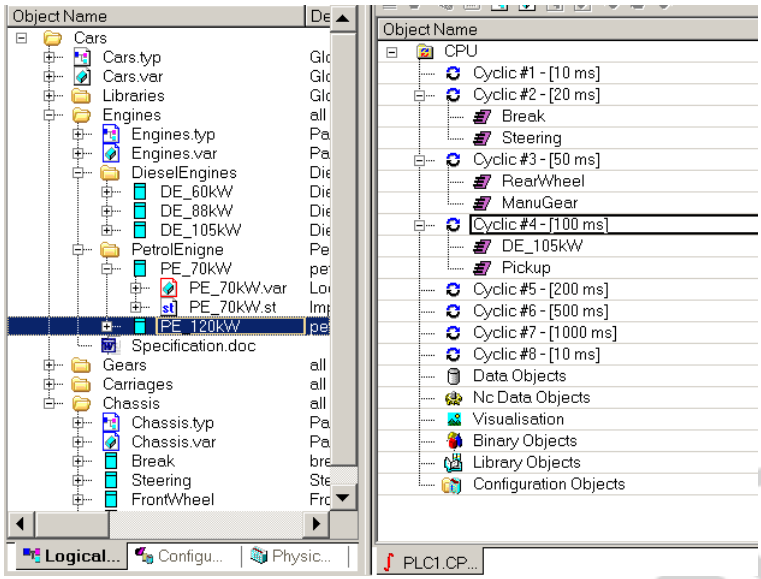


Fig. 59: Finished software configuration for the pickup

Note:

Only programs from the active configuration can be carried over to the software configuration (selected in the configuration view by double clicking).
Programs (logical view) become control tasks when transferred to the software configuration.

Task: "Integrating your own configuration"

With the help of the **"Pickup" configuration example**, try to create your own configuration.

Solution approach:

- Create the project structure
- Create a new configuration
- Add the required hardware
- Open the software configuration
- Assign the software

5.4.5 Directory structure of a project

A project is divided up between several folders. This division is based on the project views.

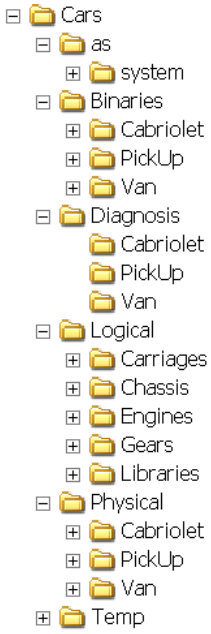


Fig. 60 Project structure

- AS
  The operating systems and libraries used in the project
- Binaries
  Compiled programs separated by configuration
- Diagnosis
  Diagnostics information (e.g. saved watch window) separated by configuration
- Logical
  Packets and files, which can be found in the logical view
- Physical
  Configurations, which can be found in the physical view.

5.5 Teamwork

In a project team, responsibilities must be split and assigned to team members. Automation Studio supports working in a project team with dedicated functionalities. It provides functionalities for efficient project data exchange with minimum data sizes.

5.5.1 Project Export

When selecting the point **File – Export Project** from the main menu it is possible to export any desired subset of objects in your project. First, select all objects to be exported from the logical project view and the configurations view.

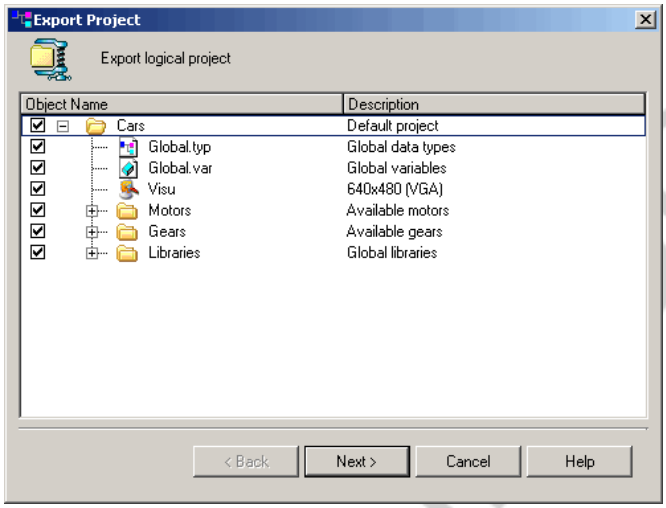


Fig. 61: Selecting software objects from the Logical View

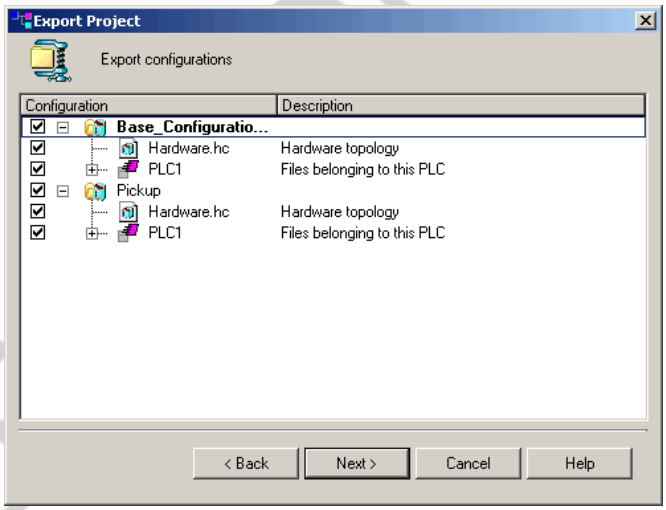


Fig. 62: Selecting configurations from the Configuration View

The ZIP-archive will be generated with the selected compression level in the selected directory.

5.5.2 Project Import

Exported projects may be imported by selecting the point **File – Import Project** from the main menu.

If objects with identical names exist in your project you will be prompted for confirmation of replacement:

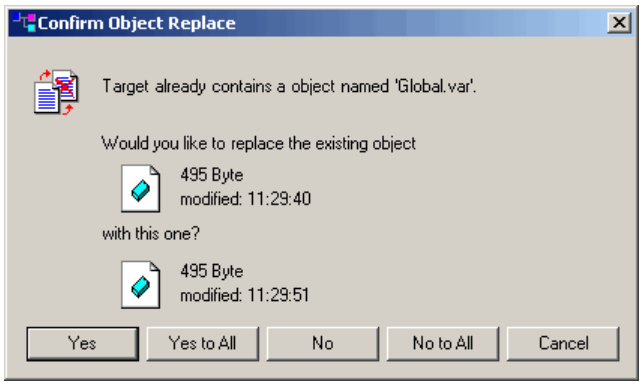


Fig. 63: Confirm Object Replace dialogue

5.5.3 Clean Project

With the point **Project – Clean Project** from the main menu the size of a project can be substantially reduced. This is especially convenient if the project is sent to other team members by email.

If you choose the options in the clean dialogue the project will be reduced to a minimum size. In any case no source information will be lost.

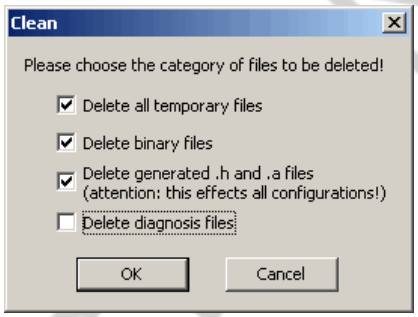


Fig. 64: Clean project dialogue

Note that deleting all temporary files will destroy information necessary for debug purposes.

6. OPERATING COMFORT

6.1 Editor Views

The division of the Automation Studio workspaces into different views makes it possible to simultaneously open multiple working environments.

6.1.1 Switching via menu or shortcut key

Switching between windows can be done either from the **Window** menu or using the <**CTRL** + **TAB**> key combination.

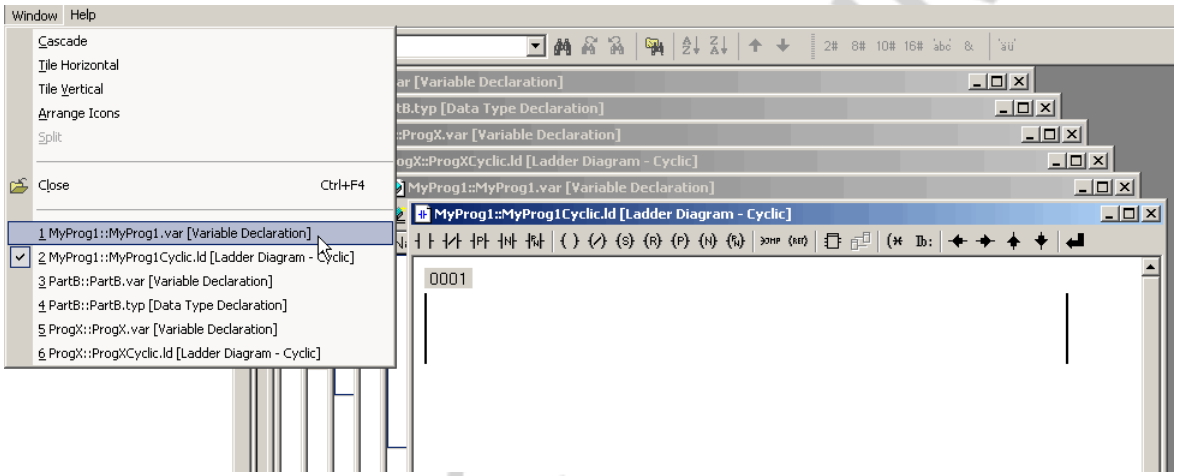


Fig. 65 Switching between the individual windows

Note:

It is helpful to close windows that are not needed (if finished editing for quite a while). This leaves only those windows open that are actually needed.
The result is a cleaner, more efficient method of working.

6.1.2 Workbook mode

The individual windows can also be managed as workbooks. This option can be switched on using the setting in the main menu **View:Workbook**.

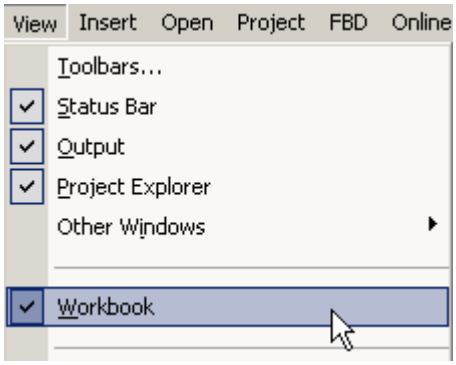


Fig. 66 Workbook view

The windows can now be switched using tabs in the workbook.

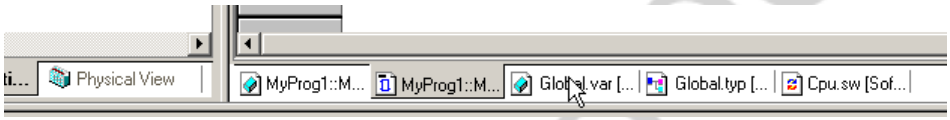


Fig. 67 Using the workbook mode

6.1.3 Open as text or table

All declaration tables can be displayed as either text or table in Automation Studio using the open data storage as IEC files.

The option **Open as Text** or **Open as Table** can be selected from the shortcut menu by right-clicking on the declaration object.

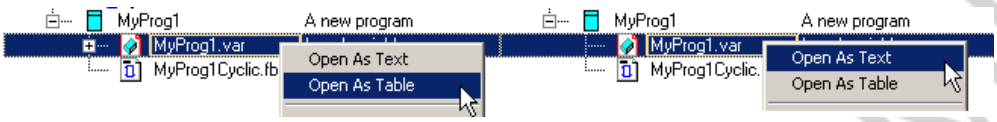


Fig. 68 Open as table or text

This causes the declaration to be displayed as a table or as text, in accordance to the **IEC-Norm 61131-3**.

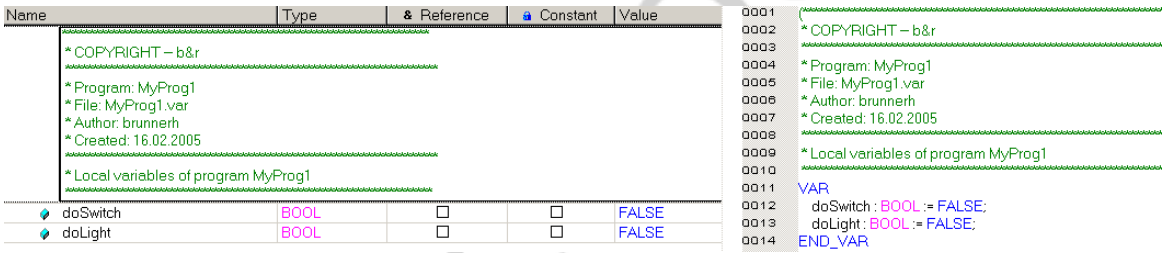


Fig. 69 Open as table or text

6.2 Smart Edit

The following language elements can be **automatically complemented** using the CTRL-Space shortcut:

- Variable names
- Structure member
- Function name
- Language constructs (IF THEN, CASE,..)

The following navigation aids are available:

- Goto
- Locating the variable declaration
- Locating the variable declaration
- Use in the source code
- Corresponding brackets

Move the mouse pointer over objects to display the following **tooltips**:

- Parameter lists for functions and function blocks
- Variable data type

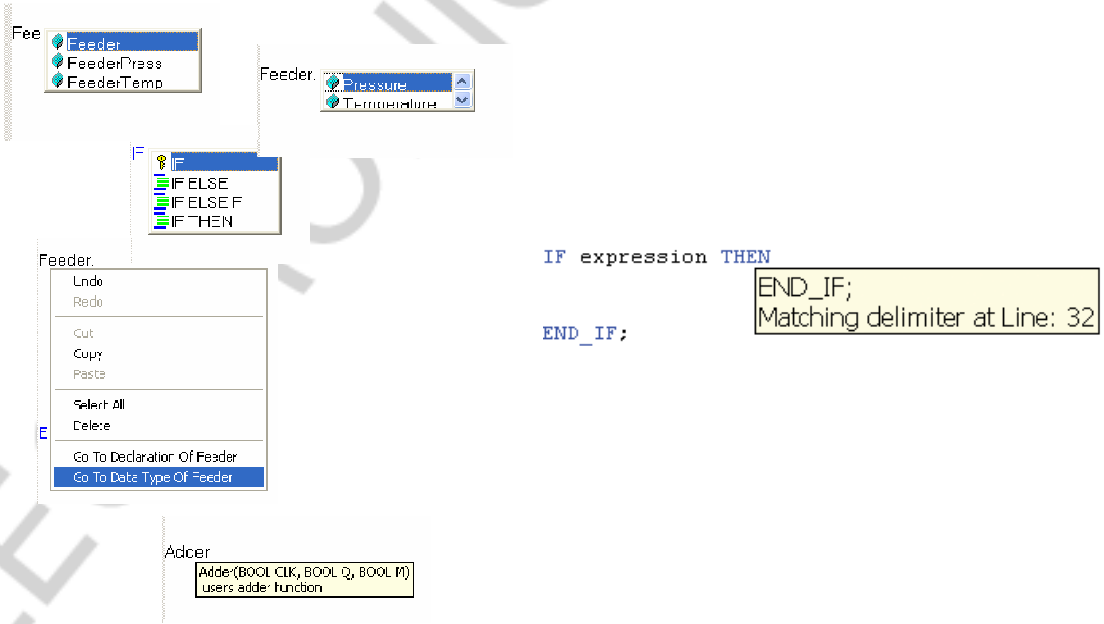


Fig. 70: Smart Edit

The **Zoom-In** option offers more operating convenience when editing project sections.

6.3 Open data storage

The data storage in Automation Studio offers an open entrance. The following possibilities result:

- All project data stored in ASCII files
- External generation possible
- Separation of source data and compiled data
- Use of version checking systems possible
- Zip export/import

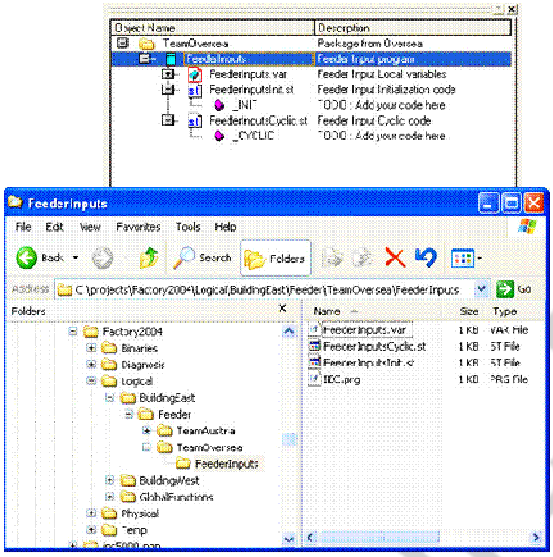


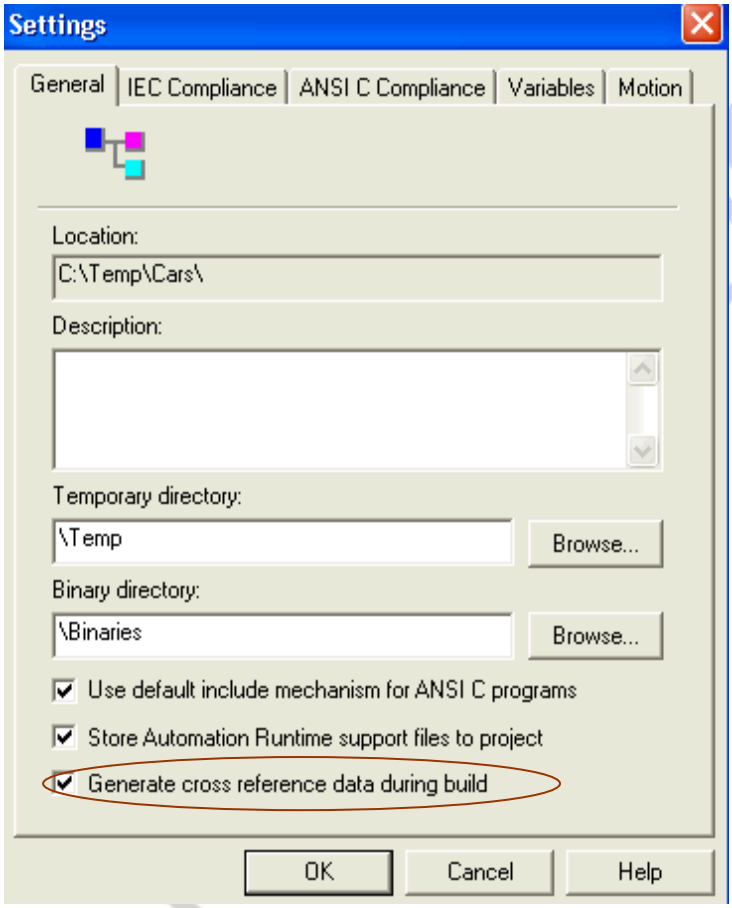
Fig. 71: Open data storage

6.4 Cross reference

Common search tasks can be handled easily with the help of the cross reference list.

For example, all variables that are used in a program can be listed. Additionally, information about **where and how** a variable is used is made available (read or write access).

To use the cross reference list, go to the menu item **Project: Settings** in the **General** tab and activate the option **"Generate cross reference data during build"**.



After compiling, cross reference list features are available.

Detailed information about the cross reference list can be found in the online help.

7. VARIABLES

Variables are symbolic elements that are used in programming.
They represent memory positions that can be either read or written by accessing a variable.
Using these symbolic elements allows the user to not worry so much about memory management since this is handled by the programming task.

Constants are much like variables. Unlike variables, however, constants can only be assigned an initial value when the software is being created. Constants can no longer be written during runtime.

7.1 Data types

Data types describe the properties of a variable. For example, these can include the possible range of the number stored in the variable, its accuracy, or which operations are possible with it.

7.1.1 Basic data types

The following data types are among what are called basic data types. They can be used in all programming languages.

Binary	Unsigned	Signed	Floating point	Time, date, string
BOOL	USINT	SINT	REAL	TIME
	UINT	INT	LREAL	DATE_AND_TIME
	UDINT	DINT		STRING

Data type	Memory requirements [bytes]	Value range
BOOL	1	TRUE (1), FALSE (0) Digital inputs and outputs
SINT	1	-128 ... +127
INT	2	-32768 ... +32767 Analog inputs and outputs
DINT	4	-2147483648 ... +2147483647
USINT	1	0 ... 255
UINT	2	0 ... 65535
UDINT	4	0 ... 4294967295

REAL	4	-3.4E38 ... +3.4E38
LREAL	8	-1.79769313486231E308 ... +1.79769313486231E308
TIME	4	T#-24d_20h_31m_23s_648ms ...T#24d_20h_31m_23s_647ms
DATE_AND_TIME	4	DT#1970-01-01-00:00:00 ... DT#2106-02-07-06:28:15
STRING	Variable	Character string display

7.2 Declaring variables and constants

Variables and constants are declared in Automation Studio as follows.

Open the variable declaration in the desired packet by **double clicking**. Variable declarations use the file extension ***.var**.

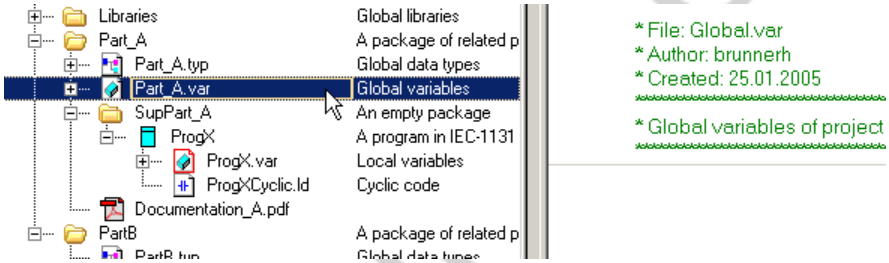


Fig. 72 Opening the variable declaration

The declaration window is opened on the right side of the screen. Select **shortcut menu: Insert Variable** to add variables.

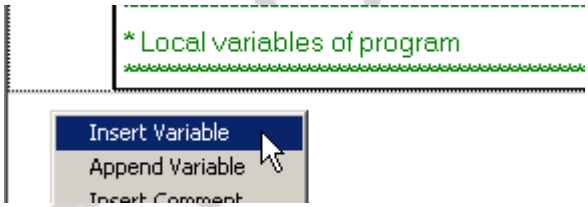


Fig. 73 Adding variables

Choose a name for the variable and the desired data types, then save the declaration. The variable / constants can now be used in your programs.

The variable can be declared as a constant by selecting the checkbox. An initial value can also be determined for a variable or constant.

Fig. 74 Variable

A variable's data type can be changed as follows: Either write the data types directly in the Type column, or **double click** on the variable and then click on the ... icon.

The following window is opened.

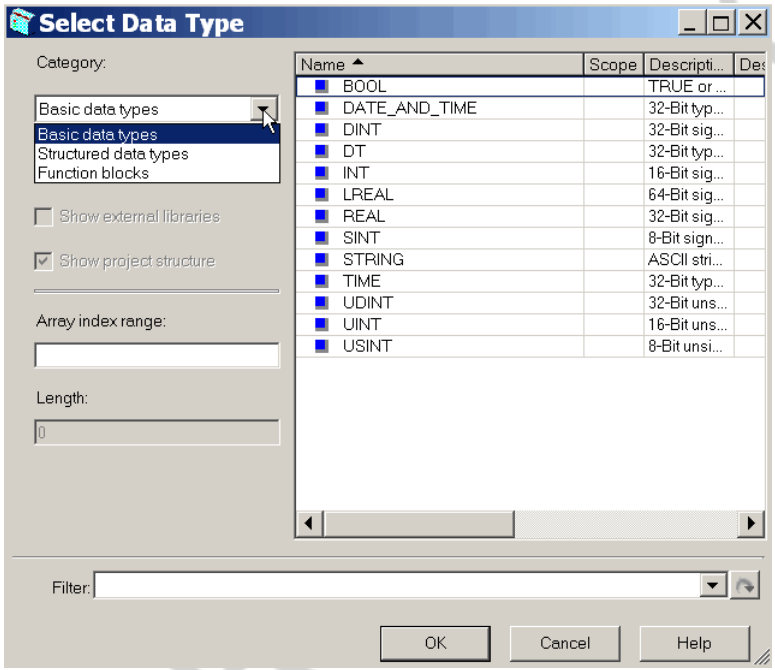


Fig. 75 Selecting the data type

You can choose from **Basic data types**, **Structured data types** and **Function blocks** in the **Category** pull-down menu. This enables you to assign your variables the suitable data type.

The size of fields is determined by the **Array index range** text field. For example the entry for a size **4** field could be **0..3**.

7.2.1 Structures (user data types)

The user can group a collection of variables in a structure. This allows individual values that would otherwise be scattered around to be grouped together to form structures that reflect a certain function or task.

Example: User data type

You have been given an assignment in which you have to create a program that can bake two types of bread.
One type of bread is defined using the variables Water, Flour, Salt and Yeast. The bread data type could consist of the following elements:

```
Water
Flour
Salt
Yeast
```

You need the bread types mixed and homemade. One advantage of the structure is that you only have one variable "mixed_bread" and one variable "homemade_bread" in your software. These variables each contain the elements water, flour, salt and yeast.

To expand your program to include an extra type of bread, you only have to create an additional variable (e.g. "white_bread") and have all of the respective data. If you notice later that you also have to specify the baking time for each type of bread, then you can simply expand the structure to include the "baking time" element. As a result, you immediately have a "backing_time" for all bread types.

In this example, you have three variables with the bread data type instead of 15 individual variables.

Creating a user data type:

To create data types in Automation Studio, the data type declaration of the desired packet must be opened by **double-clicking**. Data type declarations use the file extension ***.typ**.

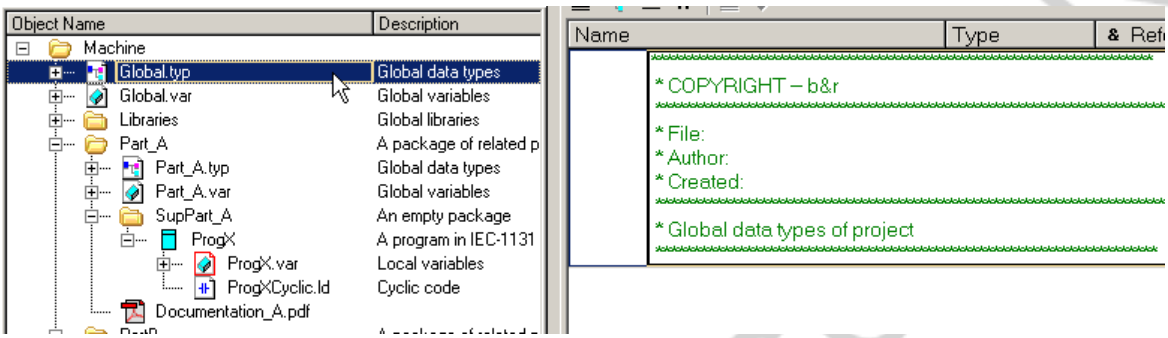


Fig. 76 Opening the data type declaration

The declaration window is located on the right half of the screen. A new data type can be inserted by selecting **Insert Datatype** from the **shortcut menu**. Enter a name.

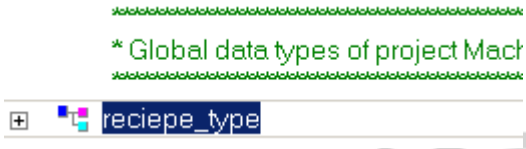


Fig. 77 Adding and naming a data type

The individual elements, which the datatype will contain, are added by selecting **Insert Datatype Member** from the **shortcut menu**.

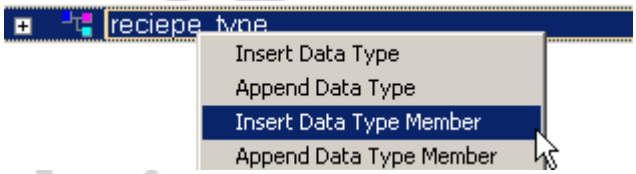


Fig. 78 Adding elements

Data types and their elements can also be inserted using the following toolbar.

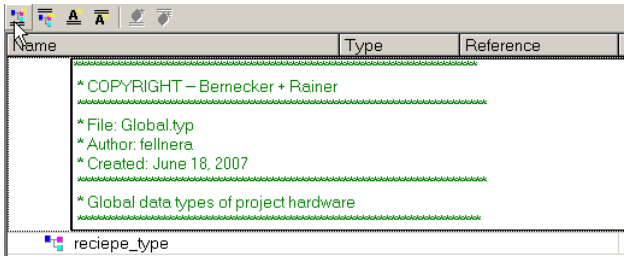


Fig. 79 Toolbar for creating data types

A finished data type might look something like this:

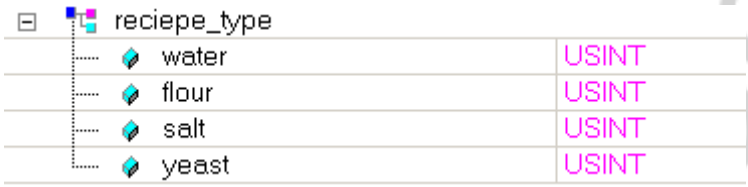


Fig. 80 Data type

After being saved, this can be used immediately in your programs.

7.2.2 Function block data types

Each function block has inputs and outputs that are grouped together in the form of a structure. When the function block is called, the actual program behind the function block receives this data structure.
In the Watch window, you can clearly see that a function block consists of individual elements when it is added.

7.2.3 Arrays

Arrays are variables that contain several elements with the same data type. These elements are accessed using an index. These elements can be declared either as basic data types (simple array) or as a user data type (array of structures).

The array index always begins with 0. This means that the array index can only receive values **0 to (size of the array - 1)** when accessing the individual variables.

Accessing an element in a simple array looks like this:

```
ArrayVariable[ArrayIndex]
```

Arrays of structures would look like this:

```
ArrayVariable[ArrayIndex].Element
```

In Automation Studio, a variable can be declared as an array in the variable declaration window when selecting a data type.

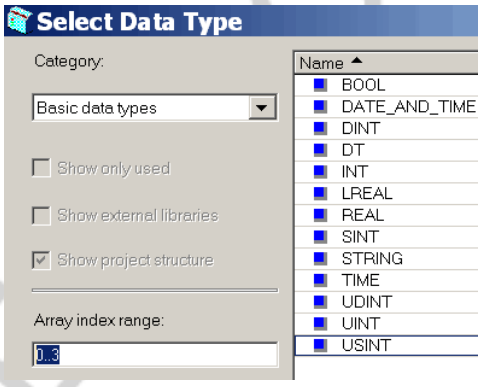


Fig. 81 Setting the size of an array

Arrays are used when variables of the same data type are needed (array of base data type or structure).

7.2.4 Variable scope

A project's packages can be as deeply nested / structured in the logical view as needed. This enables the encapsulation of data.

This structure determines the scope / visibility of the declared variables and data types. This allows us to define variables "logically" at a suitable location in the project.

This results in the following difference regarding the visibility of variables in Automation Studio:

- **Global variables** at the highest level are visible in the entire project. These are also global from the standpoint of the controller.
- **Package-local variables**, declared within a package are valid in the respective package and all subordinate package and programs. However, the validity of these variables is global from the standpoint of the controller.
- **Local variables**, declared in a program and only visible in this program. These are also local from the standpoint of the controller.

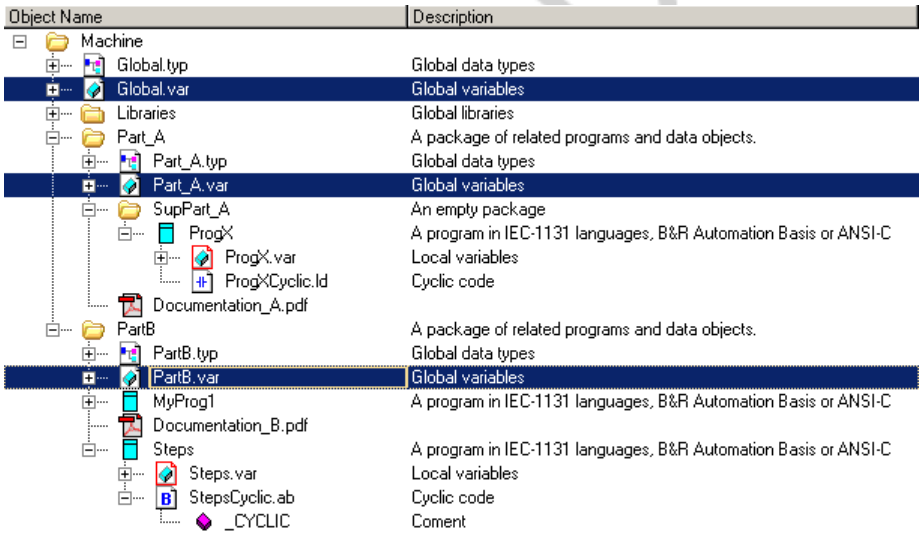


Fig. 82 Variables declared within packets

For example, we have two equal packages, **package A** and **package B**. We define the variable "**MachineType**" in both packages. What would happen if we were to compile the project?

8. INITIALIZATION

Initializing data – in this case, variables and constants – is an important topic.
Variables should have defined values at all times. There are several ways for variables to be initialized – either by the system or by the user.

The initializations become accomplished in this order:

- Variable declaration window
- Task initialization
- Cyclic task section

Variable declaration: Initialization values can be entered for variables and constants in the variable declaration window.
The **Value** column is used to set the initialization value. There are two possibilities:

- Variables can be initialized with a **fixed** value (numeric value within the value range of the variable).
- Variables can be identified as **remanent** (RETAIN). These values are **backed up** in a buffered memory area before a system restart and reloaded during the restart (remain after a warm restart).

Fig. 83 Declarations

Task initialization: If available, each task cycles through its initialization subprogram (Init-Sp) when the cyclic system starts (this occurs before the cyclic part of the program is executed).
This Init-Sp can contain program code that defines variable values.

Cyclic task section: The cyclic part of the program starts after the variable declaration and the task initialization. Variables that are assigned values there retain them until they receive new ones or the system is restarted (see the sections on variable declarations and remanent variables).

Remanent / RETAIN and permanent variables: As mentioned above, remanent variables are stored in a secure memory area during a system restart (warm restart of power loss) where they can be read back once the system is finished restarted. Permanent variables are handled in much the same way, except they can withstand cold restarts, too. In both cases, the buffering (battery, rechargeable battery) in the CPU or backplane is responsible for holding on to the data.

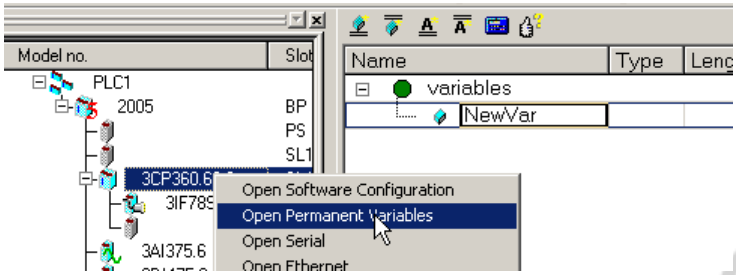


Fig. 84 Inserting a permanent variable

For variables to be created in the permanent area, they have to be defined as **RETAIN** in the variable declaration window.

9. PROGRAMMING LANGUAGES

9.1 Overview

Programs can be created in several different programming languages in Automation Studio. For this reason, mixing several programming languages together within a project is both allowed and desired as long as it gets you to your goal.

The following programming languages are available:

Programming language	Comment
Ladder diagram (LD)	Graphical
Function Block Diagram (FBD)	Graphical
Continuous Function Chart (CFC)	Graphical
Sequential Function Chart (SFC)	Graphical & textual
Instruction List (IL)	Textual
Structured Text (ST)	Textual
Automation Basic (AB)	Textual
ANSI C (C)	Textual

In Automation Studio, all textual programming languages use the same editor. Diagnostic tools are therefore always the same and are operated in the same way. This uniformity makes it easier to work and increases productivity.
The Watch window for checking and setting values is operated the same way regardless of whether the programming language is textual or graphical.

Note:
Function blocks from B&R standard libraries can be called and used in all programming languages.

9.1.1 Possibilities

It is possible to set the desired application with any programming language. Each language has its special strengths.

The following table lists the programming languages in the header columns. The rows represent different function groups.

	LAD	FBD	CFC	SFC	IL	ST	AB	C
Logic	√	√	√	√	√	√	√	√
Arithmetic					√	√	√	√
Decisions	√	√	√	√	√	√	√	√
Loops						√	√	√
Step sequencers				√		√	√	√
Dyn. variables						(√)	√	√
Function blocks	√	√	√	√	√	√	√	√

Note:
Using function blocks allows functions that are not supported by a programming language to be expanded.

A graphic editor is used to create the logic for ladder diagram programming.

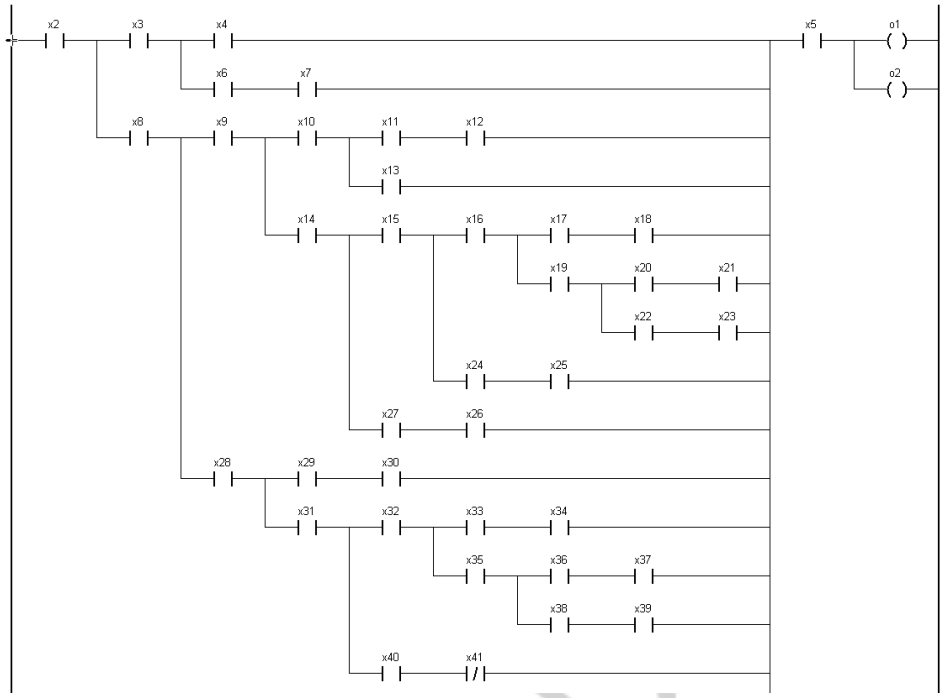


Fig. 85 Ladder diagram programming

The function chart editor offers another possibility for graphical programming.

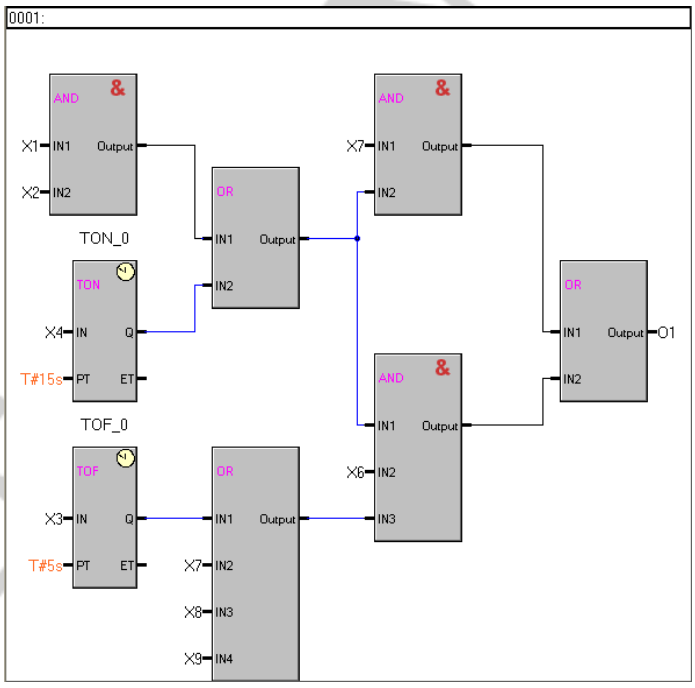


Fig. 86: Function chart programming

Sructured text is a type of textual high-level language programming.

```
PROGRAM _INIT

  (* calculate max. number of stations *)
  ModGrpNrMax:= UDINT_TO_UINT(SIZEOF(Station)/SIZEOF(Station[0]));
  (* calculate max. number of modules per station *)
  ModNrMax:= UDINT_TO_UINT(SIZEOF(Station[0].Modul)/SIZEOF(Station[0].Modul[0]));

  ModGrpNrError:= READY_FOR_NEW_ENTRY;            (* reset module group number for errorhandling *)
  ModNrError:= READY_FOR_NEW_ENTRY;               (* reset module number for errorhandling *)

END_PROGRAM

PROGRAM _CYCLIC

  tmpModOK:= TRUE;
  FOR ModGrpNr:= 0 TO (ModGrpNrMax - 1) DO
    FOR ModNr:= 0 TO (ModNrMax - 1) DO
      (* check only configured modules *)
      IF (Station[ModGrpNr].Modul[ModNr].Used = TRUE) THEN
        (* check only those configured modules with a connected PV on ModuleOK-Flag *)
        IF (Station[ModGrpNr].Modul[ModNr].pModulOkPv <> 0) THEN
          dModulOK ACCESS Station[ModGrpNr].Modul[ModNr].pModulOkPv;
          Station[ModGrpNr].Modul[ModNr].OK:= dModulOK;
          IF (Station[ModGrpNr].Modul[ModNr].OK = FALSE) THEN
            tmpModOK:= FALSE;                     (* set error *)
            (* error not recognized yet and logger ready for a new entry *)
            IF (BIT_TST(Station[ModGrpNr].Modul[ModNr].StatusInfo, 0) = FALSE) AND (ModGrpNrError = READY_FOR_NEW_ENTRY) THEN
              Station[ModGrpNr].Modul[ModNr].StatusInfo:= BIT_SET(Station[ModGrpNr].Modul[ModNr].StatusInfo, 0);
              ModGrpNrError:= ModGrpNr;           (* set module group number for errorhandling *)
              ModNrError:= ModNr;                 (* set module number for errorhandling *)
            END_IF;
          ELSE
            Station[ModGrpNr].Modul[ModNr].StatusInfo:= BIT_CLR(Station[ModGrpNr].Modul[ModNr].StatusInfo, 0);
          END_IF;
        ELSE
          (* no PV connected on ModuleOK-flag *)
          Station[ModGrpNr].Modul[ModNr].OK:= FALSE;
        END_IF;
      ELSE
        (* module not configured *)
        Station[ModGrpNr].Modul[ModNr].OK:= FALSE;
      END_IF;
    END_FOR;
  END_FOR;
  AllModulesOK:= tmpModOK;

END_PROGRAM
```

Fig. 87 Structured Text programming

ANSI C is also a text-based high-level language. It has a different notation and syntax than B&R Automation Basic.

```
/*-----------------------------------------------------------*/
int  _CYCLIC main( void ) /* Zyklischer Teil des C-Objektes */
{
  int i;
  int NachfOk;

  if(GrpSelected) /* Weg in Listbox ausgewählt */
  {
    WegPtr[1].Bez=&WegBez1;
    strncpy(BezEdit, WegPtr[GrpSelected].Bez->Bez[WegSelected[GrpSelected]], 21);
    SaveGrpSelected=GrpSelected;
    GrpSelected=FALSE;
/*
    */
  }

  if(TimerHupe) TimerHupe--;
  for(i=1;i<=ANZ_GRP;i++)
  { /* Abschaltverzögerung bei Verlust des Nachfolgers
       in Betriebsart verriegelt */
    if(StoerTimer[i])   StoerTimer[i]--;
    /* Verriegelung ein bei Automatik */
    if(GrpStatus[i].autom) GrpStatus[i].verriegelt=TRUE;
  }

 /*  for(i=1;i<=7;i++) nur für Test */
  for(i=1;i<=ANZ_M;i++)
  {
    /* Tastenfarben zuweisen */
    Color[i]=0;  /* alle Bits rücksetzen */
    if(Taste[i])        Color[i]=TASTE_EIN;
    else                Color[i]=TASTE_AUS;
```

Fig. 88 ANSI C programming

10. SUMMARY

Automation Studio makes it possible to program all of the automation components provided by B&R. The ability to clearly structure the software based on machine parts and to work with different configurations makes it possible to manage multiple machine variations in one project and allows a whole team to work on the same project.

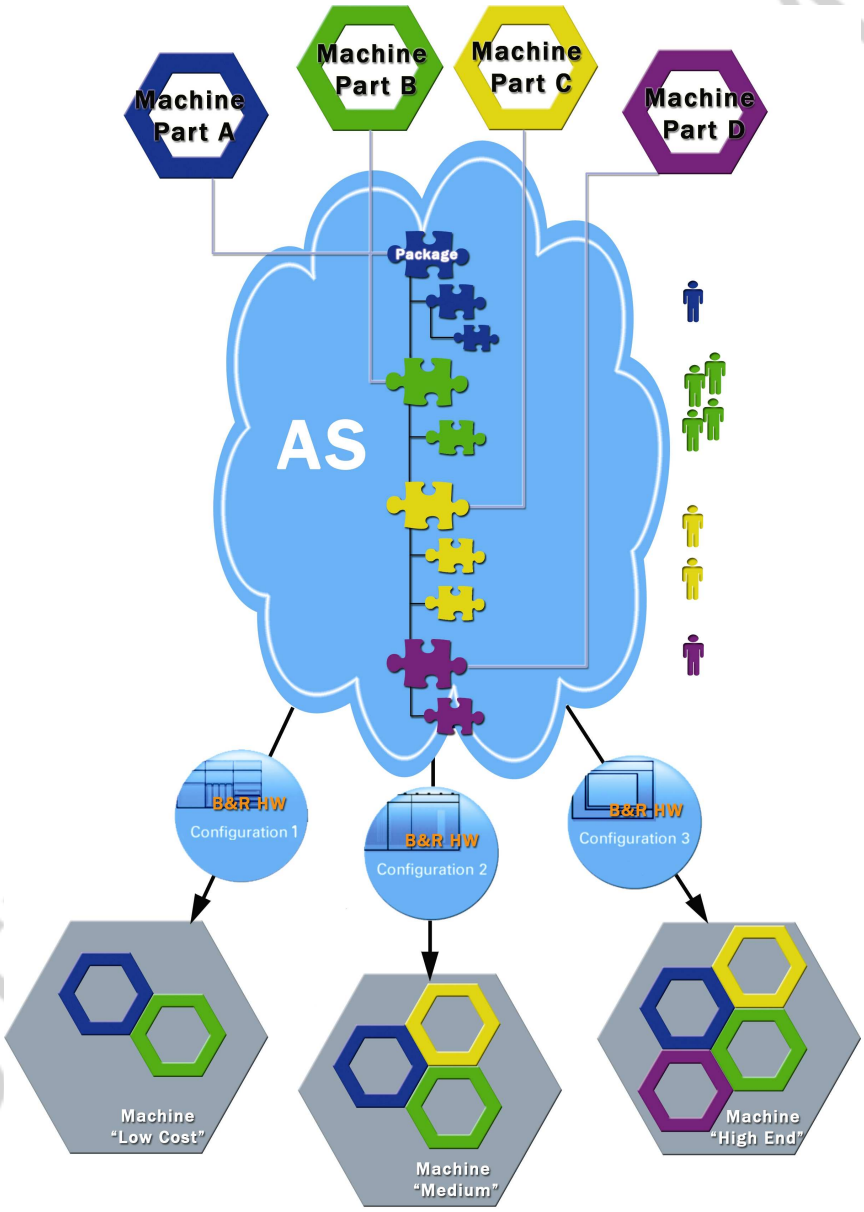


Fig. 89 Basic concept of Automation Studio

The structure of data types and variable declarations are always the same, which limits the number of different user interfaces. In turn, this makes it easier to find your way around and work with the system.

The scopes for variables, constants and data types are clearly structured using the logical view.

You are now familiar with Automation Studio and have gotten to know the Automation Studio online help system that will support you in the future while you are working.

The programming language overview allows you to select the language that is best suited to your application.

More details about them can be found in later training modules.

Notes

Overview of training modules

TM200 – B&R Company Presentation **
TM201 – B&R Product Spectrum **
TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM212 – Automation Target **
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram (LAD)
TM241 – Function Block Diagram (FBD)
TM246 – Structured Text (ST)
TM247 – Automation Basic (AB)
TM248 – ANSI C
TM250 – Memory Management and Data Storage
TM260 – Automation Studio Libraries I
TM261 – Closed Loop Control with LOOPCONR

TM400 – The Basics of Motion Control
TM410 – The Basics of ASiM
TM440 – ASiM Basic Functions
TM441 – ASiM Multi-Axis Functions
TM445 – ACOPOS ACP10 Software
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors

TM500 – The Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER

TM600 – The Basics of Visualization
TM610 – The Basics of ASiV
TM630 – Visualization Programming Guide
TM640 – ASiV Alarm System
TM650 – ASiV Internationalization
TM660 – ASiV Remote
TM670 – ASiV Advanced

TM700 – Automation Net PVI
TM710 – PVI Communication
TM711 – PVI DLL Programming
TM712 – PVIServices
TM730 – PVI OPC

TM800 – APROL System Concept
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM840 – APROL Parameter Management and Recipes
TM850 – APROL Controller Configuration and INA
TM860 – APROL Library Engineering
TM865 – APROL Library Guide Book
TM870 – APROL Python Programming
TM890 – The Basics of LINUX

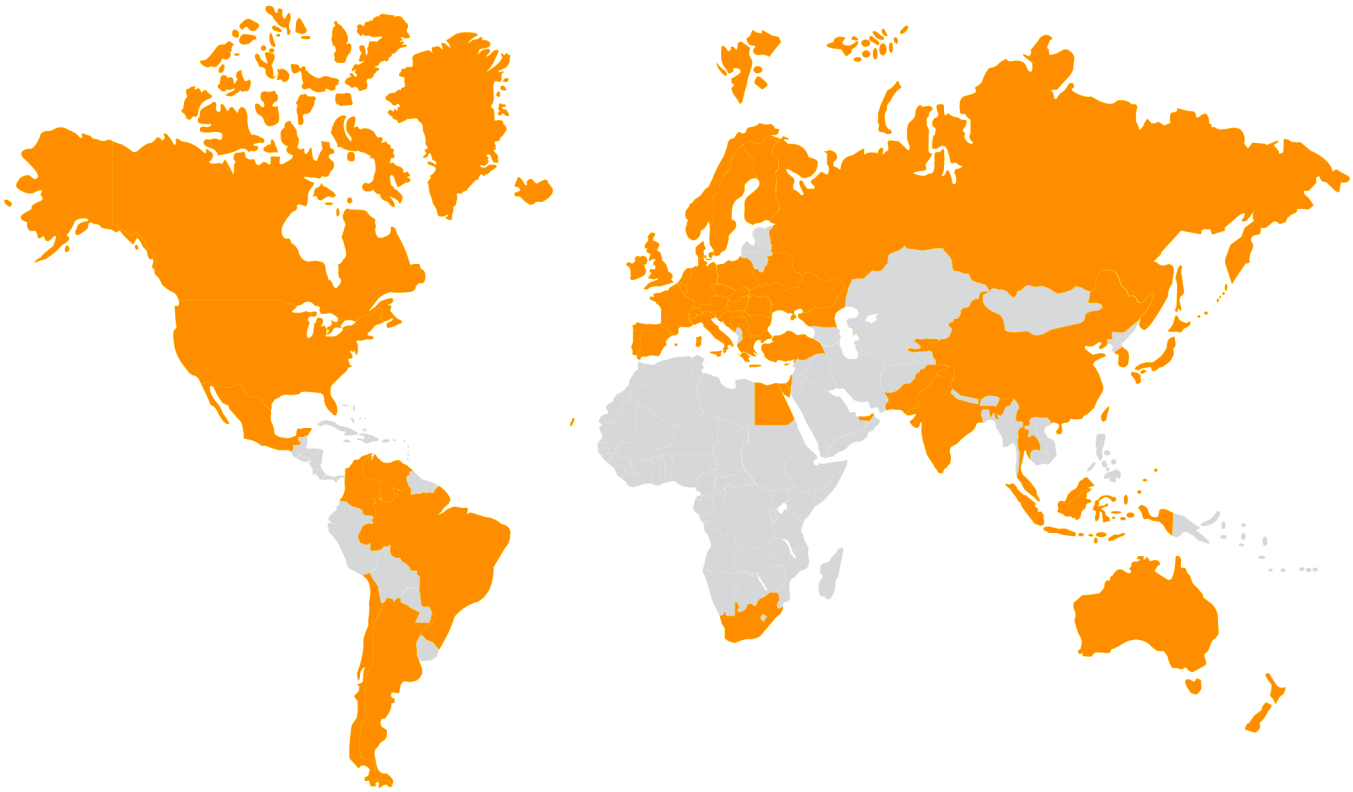
**) see Product Catalog

CORPORATE HEADQUARTERS

Bernecker + Rainer Industrie-Elektronik Ges.m.b.H.
B&R Strasse 1
5142 Eggelsberg
Austria
Tel.: +43 (0) 77 48 / 65 86 - 0
Fax: +43 (0) 77 48 / 65 86 - 26
info@br-automation.com
www.br-automation.com

TM210TRE.30-ENG 0907

140 offices in more than 55 countries - **www.br-automation.com/contact**



Australia • Argentina • Austria • Belarus • Belgium • Brazil • Bulgaria • Canada • Chile • China • Colombia • Croatia • Cyprus
Czech Republic • Denmark • Egypt • Emirates • Finland • France • Germany • Greece • Hungary • India • Indonesia
Ireland • Israel • Italy • Japan • Korea • Luxemburg • Kyrgyzstan • Malaysia • Mexico • The Netherlands • New Zealand
Norway • Pakistan • Poland • Portugal • Romania • Russia • Serbia • Singapore • Slovakia • Slovenia • South Africa
Spain • Sweden • Switzerland • Taiwan • Thailand • Turkey • Ukraine • United Kingdom • USA • Venezuela • Vietnam