

# Protection System Simulator SIM600

## Configuration Manual





**Contents:**

<b>1. About this manual .....</b>	<b>5</b>
1.1. Copyrights .....	5
1.2. Trademarks .....	5
1.3. Guarantee .....	5
1.4. General .....	5
1.5. Document conventions .....	6
1.6. Use of symbols .....	6
1.7. Terminology .....	7
1.8. Abbreviations .....	7
1.9. Related documents .....	8
1.10. Document history .....	8
<b>2. Product overview .....</b>	<b>9</b>
2.1. Protection System Simulator SIM600 overview .....	9
2.2. Hardware .....	11
2.3. SIM600 Editor .....	11
2.4. Software requirements .....	13
2.5. Supported data types .....	13
2.6. Supported objects .....	14
2.7. Supported functions and function blocks .....	14
<b>3. SIM600 Editor .....</b>	<b>15</b>
3.1. Opening SIM600 Editor .....	15
3.2. Using GUI Editor .....	15
3.2.1. Loading background image .....	15
3.2.2. Adding GUI objects .....	16
3.2.3. Modifying GUI object properties .....	16
3.2.4. Deleting GUI objects .....	19
3.3. Using IO Editor .....	19
3.3.1. Adding IO objects .....	19
3.3.2. Connecting IO objects .....	20
3.3.3. Modifying IO object properties .....	21
3.3.4. Mapping IO names .....	22
3.3.5. Inverting .....	22
3.3.6. Deleting IO objects and connections .....	23
3.3.7. Scaling of objects .....	23
3.3.7.1. Adding inputs .....	23
3.3.7.2. Removing extra inputs .....	24
<b>4. Compiling configuration .....</b>	<b>25</b>
4.1. Compiling configuration .....	25
<b>Appendix 1 .....</b>	<b>26</b>

GUI Tools .....	26
<b>Appendix 2 .....</b>	<b>30</b>
IO Tools .....	30
<b>Index .....</b>	<b>35</b>

## **1. About this manual**

### **1.1. Copyrights**

The information in this document is subject to change without notice and should not be construed as a commitment by ABB Oy. ABB Oy assumes no responsibility for any errors that may appear in this document.

In no event shall ABB Oy be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB Oy be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from ABB Oy, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license.

© Copyright 2008 ABB. All rights reserved.

### **1.2. Trademarks**

ABB is a registered trademark of ABB Group. All other brand or product names mentioned in this document may be trademarks or registered trademarks of their respective holders.

### **1.3. Guarantee**

Please inquire about the terms of guarantee from your nearest ABB representative. Factory warranty period of SIM600 is 1 year from product delivery.

### **1.4. General**

This manual provides thorough information on the configuration of SIM600 and the central concepts related to it. For more information on installing and using SIM600, refer to 1.9, Related documents.

Information in this user's guide is intended for trainers, product demonstrators, marketers, and sales personnel.

SIM600 Editor and its logical symbols are in some parts in compliance with the IEC 61131–3 standard. Knowledge of the standard is an advantage when you work with the editor and design the logic.

## 1.5. Document conventions

The following conventions are used for the presentation of material:

- The words in names of screen elements (for example, the title in the title bar of a window, the label for a field of a dialog box) are initially capitalized.
- Capital letters are used for the name of a keyboard key if it is labeled on the keyboard. For example, press the ENTER key.
- Lowercase letters are used for the name of a keyboard key that is not labeled on the keyboard. For example, the space bar, comma key, and so on.
- Press CTRL+C indicates that you must hold down the CTRL key while pressing the C key (to copy a selected object in this case).
- Press ESC E C indicates that you press and release each key in sequence (to copy a selected object in this case).
- The names of push and toggle buttons are boldfaced. For example, click **OK**.
- The names of menus and menu items are boldfaced. For example, the **File** menu.
  - The following convention is used for menu operations: **MenuName > Menu-Item**. For example: select **File > New**.
  - The **Start** menu name always refers to the **Start** menu on the Windows taskbar.

## 1.6. Use of symbols

This publication includes warning, caution, and information icons that point out safety related conditions or other important information. It also includes tip icons to point out useful information to the reader. The corresponding icons should be interpreted as follows.



The electrical warning icon indicates the presence of a hazard which could result in electrical shock.



The warning icon indicates the presence of a hazard which could result in personal injury.



The caution icon indicates important information or warning related to the concept discussed in the text. It might indicate the presence of a hazard which could result in corruption of software or damage to equipment or property.



The information icon alerts the reader to relevant facts and conditions.



The tip icon indicates advice on, for example, how to design your project or how to use a certain function.

## 1.7. Terminology

The following is a list of terms associated with SIM600 that you should be familiar with. The list contains terms that are unique to ABB or have a usage or definition that is different from standard industry usage. See also 1.8, Abbreviations

Term	Description
amplitude	Peak value of a sinusoidal quantity.
analogue output	A connector whose output signal is analogue.
binary output	Informs the IED of on/off states, such as disconnect and circuit breaker states, alarm signals or control functions.
connector	Device providing connection and disconnection to a suitable mating component.
digital output	See "binary output".
Input/Output	Connection points to a system. I/O can be digital (the signal is TRUE or FALSE, for example 0 or 10 V) or analogue (where the signal can vary from for example 0 to 10 V).
Intelligent Electronic Device	A physical IEC 61850 device that behaves as its own communication node in the IEC 61850 protocol.
Light Emitting Diode	Semiconductor that emits light.
protection relay	Measuring relay which, either solely or in combination with other relays, is a constituent of protection equipment.

## 1.8. Abbreviations

The following is a list of abbreviations associated with SIM600 that you should be familiar with. See also 1.7, Terminology.

Abbreviation	Description
AO	Analogue output

Abbreviation	Description
DO	Digital output
GUI	Graphical User Interface
HMI	Human Machine Interface
HSPO	High-Speed Power Output
IED	Intelligent Electronic Device
I/O	Input/Output
LED	Light Emitting Diode
PC	Personal Computer
PO	Power output
USB	Universal Serial Bus

## 1.9. Related documents

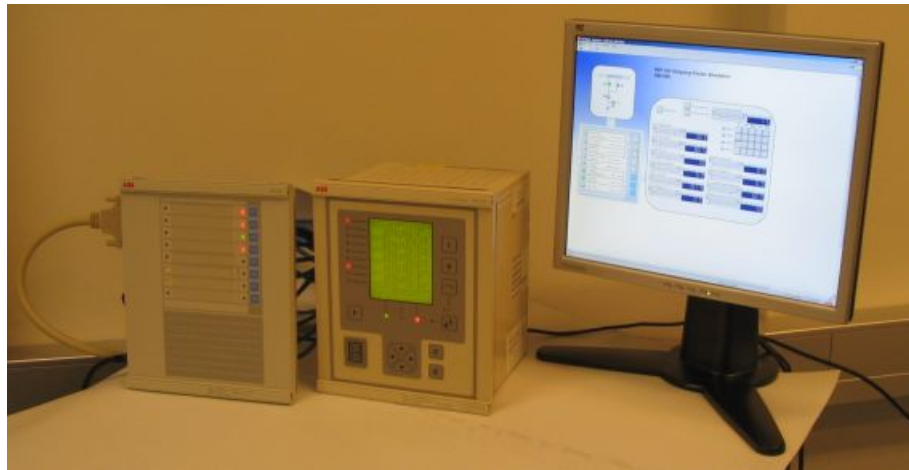
Name of the manual	MRS number
Protection System Simulator SIM600 Installation Manual	1MRS756102
Protection System Simulator SIM600 Operator's Manual	1MRS756103

## 1.10. Document history

Document version/date	Product revision	History
A/28.9.2007	1.0	Document created
B/08.12.2008	1.0	Document updated







SIM600\_connected\_to\_PC.JPG

*Figure 2.1-2 SIM600 connected to a PC and an IED*

SIM600 Editor is used for creating the configuration for Protection System Simulator SIM600. The editor is not mandatory for the operation of the simulator, if the required configuration is already available.

SIM600 Runtime is used as the user interface of Protection System Simulator SIM600.

## Modules included in SIM600

Protection System Simulator unit:

- Device

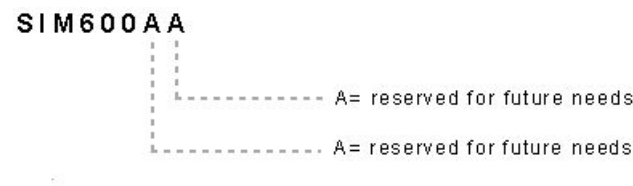
External power supply:

- Single output AC/DC
  - 90-130VAC or 180-260VAC (auto-ranging)
  - 48Vdc/80 W

Cables:

- Electric cable for an external power supply
- 48 Vdc power supply cable to a protection relay
- Binary input and output cables (2 cables)
- Analogue output cables (5 cables)
- Analogue mA output cable (1 cable)
- USB cable

## Ordering information



SIM600AA.jpg

Figure 2.1-3 Ordering code for SIM600

Option available:

- Package of cables: 5 Analogue + 1 mA cables
- Ordering code: 1MRS090070

Mounting:

- Ordering code: 1MRS050993

Commission order by ABB:

- Current amplifiers (Tillquist)
  - CXV30 Rogowski-Voltage to AC Current amplifier
- Voltage amplifiers (Tillquist)
  - Three-phase RHVD-voltage power-amplifier, type: PXV33-X-I/O V

## 2.2.

## Hardware

Protection System Simulator SIM600 is a microprocessor-based device that can be used as a stand-alone device or connected to a PC. It can be configured in any way required. Protection System Simulator SIM600 consists of a central unit, housed in a metallic case. The central unit contains a USB-connection to a PC, eight pushbuttons, and 16 LED indicators. For more information on connectors for SIM600, see Protection System Simulator SIM600 Installation Manual.

## 2.3.

## SIM600 Editor

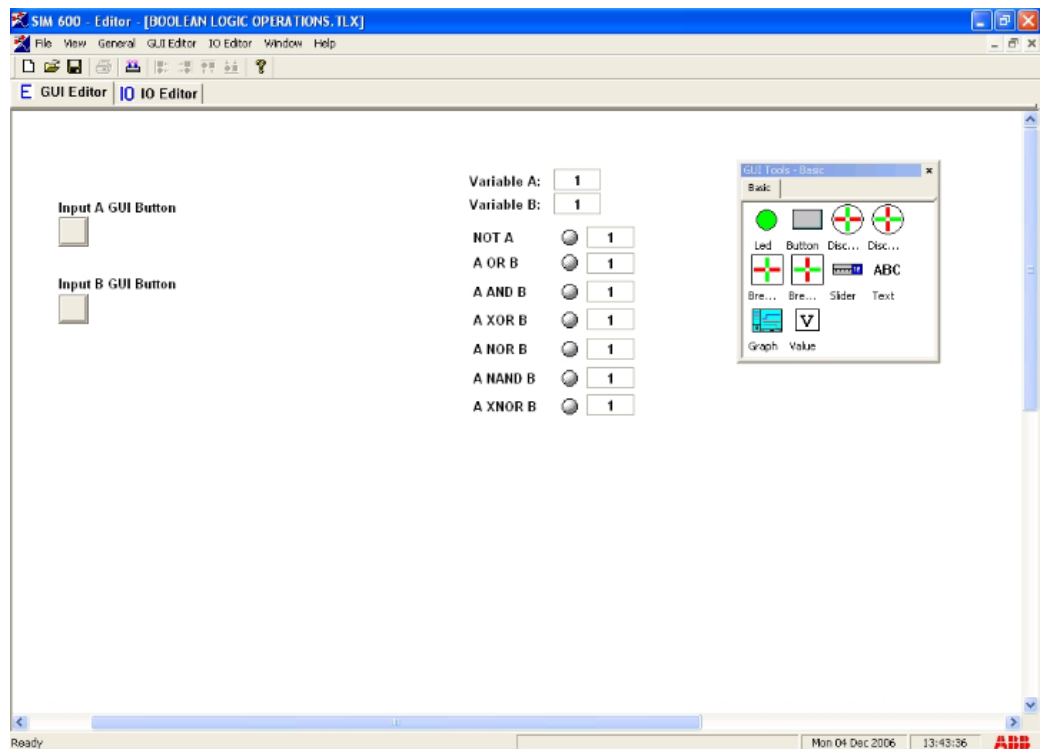
SIM600 Editor is used for configuring Protection System Simulator SIM600 for the required task. SIM600 Editor is not mandatory for the operation of SIM600, if the required configuration is already available.

SIM600 Editor has two views, the GUI Editor and the IO Editor view.

### GUI Editor

GUI Editor is used for drawing the user interface that includes all the functional objects required by the application. The objects available are shown in the GUI Tools window

on the right of the GUI Editor view. Objects can be added to the workspace and they can be freely moved within the area.

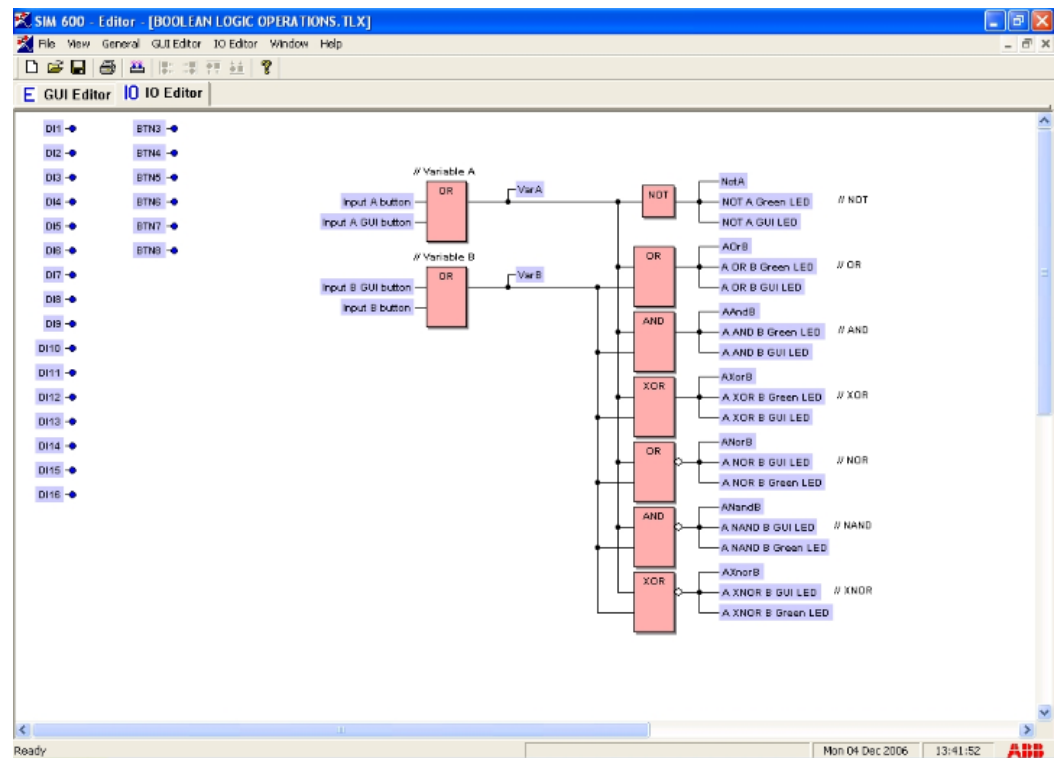


Editor\_GUI\_example.jpg

Figure 2.3-1 An example view of GUI Editor

## IO Editor

IO Editor is used to connect the objects displayed in the user interface and to specify the necessary operation for SIM600. The workspace of IO Editor contains a list of input and output signals of SIM600. The IO Tools window on the right of IO Editor view contains logic symbols. Logic symbols are used in the same way as the objects in GUI Editor, that is, they can be added to the workspace as required.



Editor\_IO\_example.jpg

Figure 2.3-2 An example view of IO Editor

For information on installing SIM600 Editor, see Protection System Simulator SIM600 Installation Manual.

## 2.4. Software requirements

SIM600 Editor and SIM600 Runtime are compatible with MS Windows 2000 or newer.

## 2.5. Supported data types

SIM600 configurations support three different data types. These data types are used for calculations and evaluations in the configuration logic. The types are listed below.

### BOOL

A **BOOL** corresponds to either a logical 0 (FALSE) or a logical 1 (TRUE). A **BOOL** always contains either of these values (but not both) and thus represents a truth-value of for example a condition or a logical function.

## UINT

The abbreviation **UINT** stands for unsigned integer, in other words a positive integer number. In the SIM600 device the UINT value is an integer ranging from 0 to +655.

## DECIM

The **DECIM** data type represents decimal numbers. In the SIM600 device **DECIM** has exactly two decimal numbers and ranges, similarly to the UINT type, from 0 to +655.35.

## 2.6. Supported objects

There are eight different types of objects available in the GUI Editor view: LED, BUTTON, DISCONNECTOR, BREAKER, SLIDER, TEXT GRAPH and VALUE. They are positioned in the GUI Editor view and logically connected in the IO Editor view. For a description of the different GUI objects, see Appendix 1, GUI Tools.

## 2.7. Supported functions and function blocks

There is a number of function blocks available for programming and configuring SIM600. The functions are divided into four categories based on their purpose of use:

- the BASIC function block group
- the MATH function block group
- the TIMERS function block group and the WAVE function block group.

For a list and description of the different function blocks, see Appendix 2, IO Tools.

## 3. SIM600 Editor

### 3.1. Opening SIM600 Editor

SIM600 Editor can be operated directly from the installation CD or it can be installed on your PC. Open SIM600 Editor by double clicking the PRSEditor.exe. SIM600 Editor has two views, the IO Editor view and the GUI Editor view. For more information on SIM600 Editor, see 2.3, SIM600 Editor.

### 3.2. Using GUI Editor

#### 3.2.1. Loading background image

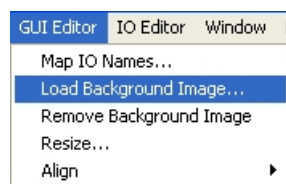
You can add a background image into GUI Editor. A background image provides space for graphics or information that cannot be added using the standard GUI objects. The background image can be made using any image-processing tool, and then loaded into GUI Editor. Make sure that the image size is suitable, because the image is shown in GUI Editor in its actual size that cannot be modified.

SIM600 supports the following image formats:

- Windows Metafile (.wmv)
- Windows Bitmap (.bmp)
- Joint Photographic Experts Group format, JPEG (.jpg, .jpeg)
- Graphic Interchange Format, GIF (.gif).

To load a background image in GUI Editor:

1. Open SIM600 Editor.
2. Create a new configuration by selecting **File > New** or open an existing configuration by selecting **File > Open**.
3. Select **GUI Editor > Load Background Image**.



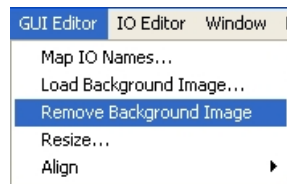
loading\_background\_image.jpg

*Figure 3.2.1-1 Loading a background image*

4. Browse for the desired file and click **Open** to load the image. The image is inserted in the top-left corner of the GUI workspace.

You can add objects on top of the background image in GUI Editor, see 3.2.2, Adding GUI objects. When you have loaded the configuration into SIM600 Runtime, you can use it as a user interface for SIM600.

To remove the background image from the configuration, select **GUI Editor > Remove Background Image**.



removing\_background\_image.jpg

Figure 3.2.1-2 Removing a background image

### 3.2.2. Adding GUI objects

You can add GUI objects to the workspace of GUI Editor.

To add GUI objects:

1. Open the GUI Editor view.
2. Create a new configuration by selecting **File > New**, or modify an existing configuration by selecting **File > Open** and by opening the desired configuration.
3. Open the GUI Tool Palette by selecting **View > Tool Palettes > New Tool Palette**. The Tool Palette has one tab page that displays all available GUI objects.
4. Drag and drop the desired GUI objects to the workspace. You can also drag and drop objects to a new position in the workspace.



To select several objects, right-click on the workspace and drag the mouse to outline the objects to be selected. When the mouse button is released, all objects within the outline area are selected.

Objects that have been added to GUI Editor appear as input or output components in IO Editor. The names of the components are the same as the name defined in the object properties dialog in the GUI Editor view. Use logical names to enable easy identification of objects, their position, purpose and functionality.

After you have added the GUI objects in GUI Editor, you have to connect the objects in IO Editor. For more information, see 3.3.2, Connecting IO objects.

### 3.2.3. Modifying GUI object properties

You can modify GUI object properties in GUI Editor to change their function in the configuration.



## Modifying led and switch properties

You can modify the name of the led and switch objects. The name of the object is shown in IO Editor, but not in the user interface.

To modify the name of an object:

1. Select the object whose properties you want to modify in GUI Editor and right-click it.
2. Select **Properties**.
3. Insert a new name for the object.
4. Click **OK**.



Give the objects unique, descriptive names that help to locate them in IO Editor.

## Modifying button properties

You can modify the name and auto-reset setting properties of the button. The button object has an auto-reset setting, which affects the click behaviour of the button. A button that is not auto-resetting stays down when pressed, returning only when pressed again. Buttons are, by default, auto-resetting, meaning that they return to their original position when released.

To modify button properties:

1. Select the button object in GUI Editor and right-click it.
2. Select **Properties**.
3. Insert a new name for the object and deselect or select the check box to change the auto-reset setting.
4. Click **OK**.

## Modifying slider properties

You can modify the name of the slider object, as well as two groups of settings, **User range** and **Internal range** settings.

The name of the slider is shown in IO Editor, but not in the user interface.

The user range settings affect the values that are displayed graphically to the user. The Min and Max user range settings determine the numbers written on the slider. The Initial Value setting determines the position of the slider when the configuration is loaded and its value must be between the Min and Max user range settings. The Decimals value sets the number of decimals shown in the slider display.

The internal range settings affect the values that are output in the configuration logic. The Type settings determine the data type of the slider output, while the Min and Max

values determine the values output when the slider is changed to its minimum or maximum position. The output value is automatically calculated based on the position of the slider and the values entered into the internal range settings.

To modify the properties:

1. Select the slider object in GUI Editor and right-click it.
2. Select **Properties**.
3. Insert a new name and new user range and internal range values for the slider. Select the decimals settings and data type from the drop-down menus.
4. Click **OK**.

### Modifying text properties

The text object is used to add text to the user interface and it is not shown in IO Editor. You can modify the displayed text, font and font size of the text object.

To modify text properties:

1. Select the text object in GUI Editor and right-click it.
2. Select **Properties**.
3. Insert the desired text and select the font type and size from the drop-down menu.
4. Click **OK**.

### Modifying graph properties

The graph object is mainly configured in the IO Editor view, but the name of its IO component can be set in the GUI properties. For more information on graph properties, see Appendix , GUI Tools.

To modify the name of the graph:

1. Select the graph object in GUI Editor and right-click it.
2. Select **Properties**.
3. Insert a new name for the graph.
4. Click **OK**.

### Modifying value properties

You can modify the name, value and font type of the value object. The value object is used as a text field for outputting values from the configuration to the GUI. The settings provided are the IO component name, the data type of the input signal, and the font and font size used for displaying the reading.

To modify the value properties:

1. Select the value object in GUI Editor and right-click it.
2. Select **Properties**.

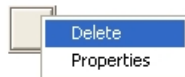
3. Insert a new name for the value object. Select a value type, font type and size from the drop-down menus.
4. Click **OK**.

### 3.2.4. Deleting GUI objects

You can delete GUI objects only in the GUI Editor view, not in the IO Editor view. Before deleting GUI objects, all connections to their IO components must be deleted, see 3.3.6, Deleting IO objects and connections.

To delete objects in the GUI Editor:

1. Select the object you want to delete in the workspace of GUI Editor.
2. Right-click the object and select **Delete** or select the object and press the **Delete** key. The object is deleted from both GUI Editor and IO Editor.



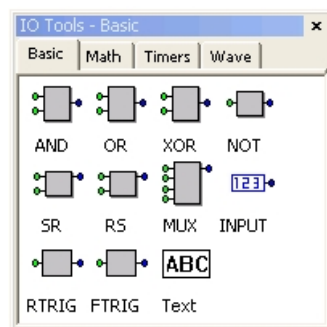
deleting\_GUI\_objects.jpg

Figure 3.2.4-1 Deleting GUI objects

## 3.3. Using IO Editor

### 3.3.1. Adding IO objects

The Tool Palette in IO Editor view contains all function blocks that can be added to a configuration.



IO\_tools\_palette.jpg

Figure 3.3.1-1 IO Tools

To add IO objects:

1. Open the IO Editor view.
2. Create a new configuration by selecting **File > New** or modify an existing configuration by selecting **File > Open** and by opening the desired configuration.

3. Open the IO Tool Palette by selecting **View > Tool Palettes > New Tool Palette**. The Tool Palette has four tab pages that contain all function blocks that can be added to a configuration.



You can have several Tool Palettes open at the same time.

4. Select the desired function block in the Tool Palette.
5. Drag and drop the function block to the IO Editor workspace. You can also move the IO object by dragging and dropping it in the IO Editor view.



To select several objects, right-click on the workspace and drag the mouse to outline the objects to be selected. When the mouse button is released, all objects within the outline area are selected.

Objects that have been added in the GUI Editor appear as input or output components in the logic. The names of the components are the same as the name defined in the object properties dialog in the GUI Editor view. For more information, see 3.2.2, Adding GUI objects.

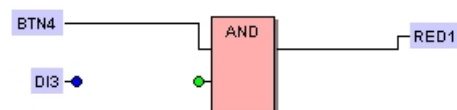
After you have added the IO objects in IO Editor, you have to connect them. For more information, see 3.3.2, Connecting IO objects.

### 3.3.2.

### Connecting IO objects

Function blocks have to be connected in a configuration to allow them to communicate with each other. The input and output pins of function blocks are connected to other function blocks so that the desired operation of the blocks together corresponds to the operation of the simulation. For information on adding IO objects, see 3.3.1, Adding IO objects.

Unconnected pins are displayed as coloured circles. Green circles represent input pins and blue circles represent output pins. When a pin is connected, the circle disappears to indicate a connection.



IO\_object\_example.jpg

Figure 3.3.2-1 An example of connected and unconnected pins of a function block

To connect two pins together:

1. Click on the circle of either one of the pins to be connected. The connection lines can be drawn starting from inputs or outputs.
2. Move the cursor towards the pin to be connected to and click on the circle. A line between the two pins indicates that the pins have been connected.
3. To draw a connection line with additional angles, click on the work space to make an angle to the connection line. Continue moving the cursor to the pin or to another a line to connect the pin.



If an invalid attempt to connect lines is made, an error message is displayed.

The following rules have to be followed when connecting IO objects:

- Output pins, as well as input pins, cannot be connected together.
- A line cannot be connected to itself.
- An output can be connected to any number of inputs, while an input can only be connected to one output.
- An output cannot be connected to an existing connection line. If two signals need to be applied to the same line, a function block or several blocks must be used to merge the two signals into a single signal, using for example a logical or a mathematical function.
- A connection line cannot be connected to another connection line.
- An input or output pin that has been connected cannot accept further connections. If such are needed, the connections must be made to the connecting line of the pin.

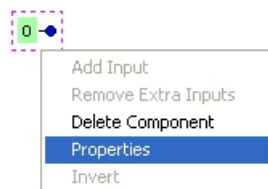
### 3.3.3.

### Modifying IO object properties

The values and types of the input signals to an IO function block often internally determine the properties of the block and cannot be configured by the user. However, the properties of the INPUT component and the TEXT component can be configured without editing the configuration logic.

To modify INPUT and TEXT object properties:

1. Select the object whose properties you want to modify.
2. Right-click the object and select **Properties**.



modifying\_IO\_properties.jpg

Figure 3.3.3-1 Modifying IO object properties

3. Select the desired input type and insert input value for the INPUT object. Properties of the INPUT object include data type and value of the output signal of the component. The data type can be chosen from any of the three available types (UINT, DECIM, BOOL) and the data value can be assigned to any valid value for the chosen data type.
4. Modify the TEXT object properties. The TEXT object properties enable customization of the component text, the font type and font size, similarly to the TEXT component of the GUI view.
5. Click **OK**.

### 3.3.4. Mapping IO names

Mapping IO names means, that the names of the hardware IO components, such as AO, DI and REL components, are renamed for a more precise description of their function. IO names are mapped in the IO Editor view.

To map IO names:

1. Open the IO Editor view in SIM600 Editor.
2. Select **IO Editor > Map IO Names...**  
A dialog window displaying a list of all the hardware components and the names of their corresponding IO components opens. The hardware name is listed in the left column, while the displayed name is listed in the right column.
3. Change the name of the component by clicking the name in the User name column and by entering the desired name. The hardware name should always be included in the display name. Suitable names are for example AO1: Residual voltage  $U_0$ , REL1: Close Q1 or DI1: Q0 Trip signal.



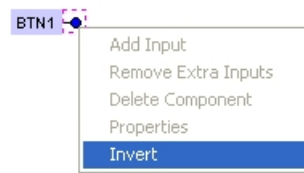
After renaming a hardware component, its name cannot be reset to its original hardware name.

### 3.3.5. Inverting

Inverting corresponds to applying the logical NOT function to a signal. This means that a logical 1 (TRUE) signal becomes a logical 0 (FALSE) signal when inverted, and vice versa. Invert only signals with BOOL type data, that is, logical signals. Application of other data on pins using numerical values may result in unpredictable and/or illogical configuration behaviour.

Inversion is applied to input or output pins of function blocks and components in the IO Editor view. Inversion is displayed as a white circle attached to the body of the component, and also attached to the pin that is inverted.

To invert a pin, right click the pin and select **Invert**. Note that double inversion, that is, the inversion of a signal twice, is equivalent to the original signal value.



inverting.jpg

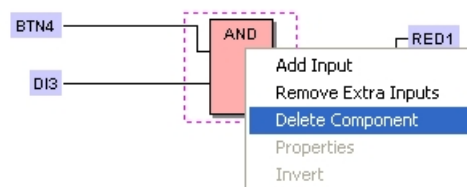
Figure 3.3.5-1 Inverting a pin

### 3.3.6. Deleting IO objects and connections

IO objects and connections can be deleted in the IO view.

To delete objects and connections in IO Editor:

1. Select the object or connection line you want to delete in the workspace of IO Editor.
2. Right-click the object or connection line and select **Delete** or select the object or connection line and press **Delete** on your keyboard.



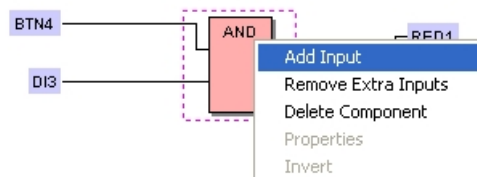
deleting\_IO\_objects.jpg

Figure 3.3.6-1 Deleting IO objects

### 3.3.7. Scaling of objects

#### 3.3.7.1. Adding inputs

Some IO components support the addition of inputs, thus increasing the number of input pins in a component and possibly decreasing the number of function blocks in a configuration. An input is added to a function block by right-clicking it and selecting **Add Input**. The extra input is appended to the bottom of the component.

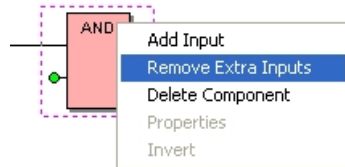


adding\_inputs.jpg

Figure 3.3.7.1-1 Adding inputs

### 3.3.7.2. Removing extra inputs

You can remove extra inputs from an IO function block. Right-click the function block and select **Remove Extra Inputs** to remove unnecessary inputs. Only the lowest unconnected inputs will be removed.



removing\_inputs.jpg

*Figure 3.3.7.2-1 Removing extra inputs*



## 4. Compiling configuration

### 4.1. Compiling configuration

When you create or modify a configuration with SIM600 Editor, you have to compile it before you can load it to SIM600. The compilation transforms the configuration file to a form, that the SIM600 device can read.

To compile a configuration:




1. Save a new configuration by selecting **File > Save As** or save modifications to an existing configuration by selecting **File > Save**.
2. Select **IO Editor > Compile** or click the **Compile** quick button in the toolbar to start the compilation.


A compilation message is displayed in the lower part of SIM600 Editor. The message contains an analysis of the configuration and informs the user about possible errors or warnings in the configuration. The last line of the message should display **Compilation was successful**, which indicates that the configuration does not contain errors. If the compilation message contains errors or warnings, there might be logical errors in the configuration and it might not work properly.

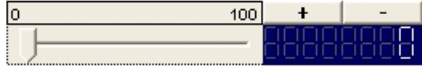
3. Correct all possible errors in the configuration and compile it again.

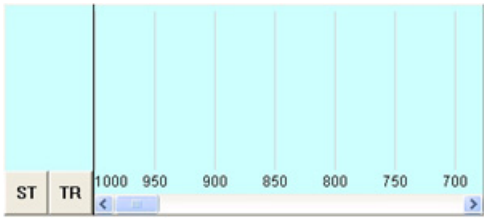
## Appendix 1

### GUI Tools

Icon in GUI Editor	Description
	A LED object has two states; on or off. It is displayed as a circle that has a grey-gradient colour in the off state, and a light green colour in the on state. A LED object represented in IO Editor has the same functionality as the LED objects in SIM600, and it operates and is connected similarly. The LED object properties contain the name of the object that is shown in the IO Editor view.
	A GUI BUTTON object is displayed as a grey box. It operates as any other BUTTON, but can be configured as stay-down. When a BUTTON object is pressed, it will set its output state to a logical 1 (TRUE). When it returns, the output is reset to a logical 0 (FALSE). A BUTTON that is configured as auto-resetting will generate a TRUE signal only for as long as it is held pressed. The IO Editor representation of a GUI BUTTON object is the same as for the hardware BTN objects (1-output object). The GUI BUTTON is connected similarly. The BUTTON object properties contain the name of the object that is shown in the IO Editor view, and a checkbox for altering the auto-reset setting of the button object.
	A BREAKER object represents a physical circuit breaker in a circuit. It indicates the state of a circuit breaker in the GUI, and it cannot be used to control or affect the simulation configuration logic. The BREAKER object is displayed as a box, with a red and green cross in the center. The IO Editor representation of a BREAKER object is a 2-input red block, with a picture of the GUI BREAKER object inside the block and the block name at the top. The two inputs are labelled "R" and "G". When the inputs are activated with a logical 1 (TRUE) signal, they set the breaker state to set-red or set-green. The BREAKER object properties contain the name of the block that is shown in the IO Editor view.

Icon in GUI Editor	Description
	<p>A DISCONNECTOR object represents a physical disconnector in a circuit. It indicates the state of a disconnector in the GUI, and it cannot be used to control or affect the simulation configuration logic. The DISCONNECTOR object is displayed as a circle, with a red and green cross in the centre. The IO Editor representation of a DISCONNECTOR object is a 2-input red block, with a picture of the GUI DISCONNECTOR object inside the block and the block name at the top. The two inputs are labelled “R” and “G”. When the inputs are activated with a logical 1 (TRUE) signal, they set the disconnector state to set-red or set-green. The DISCONNECTOR object properties contain the name of the block that is shown in the IO Editor view.</p>
<p>ABC</p>	<p>A TEXT object can be used to add informative text to the GUI. It is completely GUI-based and does not affect the configuration logic. The properties of the TEXT object contain the text to be displayed, a pull-down box listing available fonts, and a pull-down box for selecting the desired font size.</p>
<p>123.45</p>	<p>A VALUE object is used for visualizing data. The VALUE object can display information of any type (see 2.5, Supported data types) numerically in a text box. The VALUE object is displayed as a grey outline box with a number, written using a user-defined font and font size, inside it. The IO representation of the VALUE object is a 1-input object, similar to LED objects. The input pin is connected to the section in the configuration logic whose value is to be displayed. The VALUE object properties contain the name of the block that is shown in the IO Editor view, and pull-down boxes for selecting the data type of the input signal, and the desired text font and font size.</p>

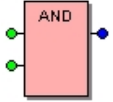
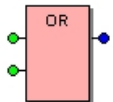
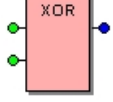
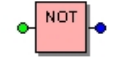

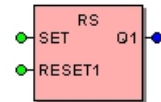
Icon in GUI Editor	Description
	<p>A SLIDER object allows the user to adjust certain values in the simulation, without having to edit and recompile the configuration. SLIDER objects can be connected to parts of the simulation where adjusting numerical values may be useful or desired. The SLIDER object is displayed as a slider with minimum and maximum values, a value display and two buttons that allow zooming the scale in and out. The IO Editor representation of a SLIDER object is a 1-output object, similar to button objects. The slider value is sent to the output as an analogue signal in the data type selected in the SLIDER object properties. The properties of the SLIDER object contain the name of the IO Editor object and additional values that affect the behaviour of the object.</p>

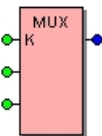
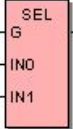



Icon in GUI Editor	Description
	<p>A GRAPH object can be used to visualize data. The GRAPH object can contain several graphs displaying different kinds of information of any of the three supported data types, see 2.5, Supported data types.</p> <p>The GRAPH object is displayed with a milli-second scale for time reading, two buttons that affect the graphing process, and a scrollbar. The names of the graphs drawn are listed in the left part of the GRAPH object. The GRAPH object is entirely configured in the IO Editor, apart from the properties that contain the name of the IO Editor object shown for the graph. The GRAPH IO object only has inputs (initially none), one for each graph it draws.</p> <p>The values entered in the dialog:</p> <ul style="list-style-type: none"> <li>• Name: The name of the graph that is displayed in the GUI graph list.</li> <li>• Range: The Min value represents the input pin value that will cause the graph to reach its lowest level. Correspondingly, the Max value represents the input pin value that will cause the graph to reach its highest level. The Type pull-down box is used to select the data type of the input pin signal and can be BOOL, UINT or DECIM.</li> <li>• Graph position and size: The Offset from top value represents how much space (in pixels) will be left between the top of the GUI Graph object and the highest level of the drawn graph. The Height value represents how many pixels of space should be left between the lowest and the highest levels of the graph. This attribute affects the height of the GRAPH object in the GUI view. The Colour pull-down box allows switching of the graph colour.</li> </ul>

## Appendix 2

### IO Tools

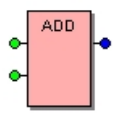
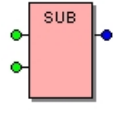
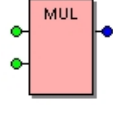
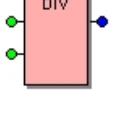
**Table A2-1 Basic IO Tools**

Icon in IO Editor	Description
	The AND function block models the logical function AND. The block has at least two, but possibly more, inputs of type BOOL, and one output of type BOOL. The block is scalable, that is, the number of inputs can be configured by the user. The output state of the AND function block depends on the state of its inputs, so that the output value is a logical 1 (TRUE) only if all of its inputs are in a logical 1 (TRUE) state. Otherwise, if even one input is in a logical 0 (FALSE) state, the output value is a logical 0 (FALSE).
	The OR function block models the logical function OR. The block is scalable, with inputs and output of type BOOL. The output state of the OR function block is a logical 1 (TRUE) if one (or more) of its inputs are in a logical 1 (TRUE) state. Otherwise, if all the inputs are in a logical 0 (FALSE) state, the output value is a logical 0 (FALSE).
	The XOR (exclusive-or) function block models the logical function OR. The block is not scalable, and thus has exactly two inputs and one output of type BOOL. The output state of the XOR function block is a logical 1 (TRUE) if one (but not both) of its inputs is in a logical 1 (TRUE) state. Otherwise, if all the inputs are in a logical 1 (TRUE) or logical 0 (FALSE) state, the output value is a logical 0 (FALSE).
	The NOT function block models the logical function NOT. The block is not scalable, and thus has exactly one input and one output of type BOOL. The output state of the NOT function block is the opposite of the state of its input. Thus, if the state of the input is a logical 0 (FALSE), the output state is a logical 1 (TRUE) and vice versa. The NOT function block can also be implemented by inverting inputs or outputs. For more information on inversion, see 3.3.5, Inverting.
	The SR function block is a memory block. The SR function block has two inputs and one output, all of type BOOL. The output state of the SR function is determined by the state of its inputs. If a logical 1 (TRUE) signal is (even momentarily) applied to the SET1 input, the output moves to a logical 1 (TRUE) state. Correspondingly, if a logical 1 (TRUE) signal is (even momentarily) applied to the RESET input, the output moves to a logical 0 (FALSE) state. If both inputs receive a logical 1 (TRUE) signal at the same time, the SR function, being set-dominant, prioritizes the signal of the SET1 input and moves the output to a logical 1 (TRUE) state. If the SET1 input changes to a logical 0 (FALSE) state while the RESET input remains a logical 1 (TRUE) signal, the output is reset to a logical 0 (FALSE) state.
	The RS function block is a memory block. The RS function block is identical to the SR function block, except that the RS function is reset-dominant. The reset-dominance translates to the RESET1 input signal being prioritized. If both inputs are initially at a logical 1 (TRUE) level, the output is reset to a logical 0 (FALSE) level. If the RESET1 input moves to a logical 0 (FALSE) state, the output switches to a logical 1 (TRUE) state.

Icon in IO Editor	Description
	<p>The MUX function block is a multiplexing function block. It is scalable, but has at least two inputs and one output. The K input is of type UINT, the other inputs and the output can be of any type. The switched input signals and the output signal are referred to as being loadable since they can be of multiple data types. The MUX function block provides a way of switching signals. A UINT number n is applied to the K input, making the MUX forward the signal from the (n+1):th input from the top to the output.</p> <p>For instance, if a MUX has three inputs (K plus two others) and a UINT 0 is applied to the K input, the output state is assigned the signal at the input directly below the K input. Correspondingly, if a UINT value of 1 is applied to the K input, the MUX switches to forward the signal of the input two steps below the K input. The output signal data type automatically switches to the data type of the connected input.</p> <p>Because the MUX is scalable, any number of inputs can be added to the function block. This means that a bigger number must be assigned to the K input in order to switch the bottom-most inputs.</p>
	<p>The SEL function block is a multiplexing function block, similar to the MUX. It has three inputs and one output. The G input is of type BOOL, the other inputs and the output can be of any type. The switched input signals and the output signal are loadable. The SEL function block cannot be extended, because the input BOOL signal has only two possible values and thus enables only the selection of two different signals.</p> <p>The SEL function block provides a means of switching the input signals. When a BOOL value is applied to the G input, either the IN0 or the IN1 signal is forwarded to the output, depending on the value of the input BOOL signal. If the BOOL value is FALSE, IN0 is forwarded to the output, whereas a TRUE input forwards the IN1 signal.</p> <p>The SEL function block should be used for input signals of type BOOL, for example buttons or logic functions. The MUX function block should be used for input signals with numerical values, for example the results of mathematical functions.</p>
	<p>The INPUT function block represents a constant value. It is loadable and always outputs the value assigned to it. The output of the block is connected to the input of the context it should be used in.</p>
	<p>The RTRIG function block reacts to changes in a signal. It is non-scalable and has one input and one output of types BOOL. The RTRIG function block detects rising edges, that is, a signal moving from a logical 0 (FALSE) state to a logical 1 (TRUE) state. In such a situation, the state of the output is switched, going to logical 1 (TRUE) if the output was at a logical 0 (FALSE) level when the input event occurred, and vice versa.</p>
	<p>The FTRIG function block reacts to changes in a signal. It is non-scalable and has one input and one output of types BOOL. THE FTRIG function block operates similarly to the RTRIG function block, differing in that the FTRIG detects falling edges, that is, a signal moving from a logical 1 (TRUE) state to a logical 0 (FALSE) state. The output state is switched as with the RTRIG function block.</p>

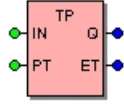
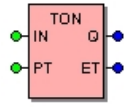
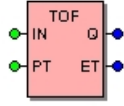
Icon in IO Editor	Description
Text object	The TEXT object can be used to add static text to the IO Editor view. The TEXT object does not affect the operation of the simulation configuration. It provides a tool for documenting and explaining the components that the configuration consists of. The TEXT object font and font size can be adjusted in the properties of the TEXT object.

**Table A2-2 Math IO Tools**

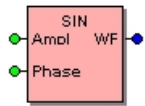
Icon in IO Editor	Description
	The ADD function block represents a mathematical addition operation. The block is scalable but always has one output and at least two inputs. The inputs are loadable and the output type adapts accordingly. If the inputs are of type BOOL, the output will be of type BOOL. Otherwise, if an input signal is analogue, the output will be of the same data type as the analogue signal. The signals can be connected to the function block inputs in any order. The ADD function block outputs the arithmetic sum of all the input signals.
	<p>The SUB function block represents a mathematical subtraction operation. The block is non-scalable and always has exactly two inputs and one output. The inputs are loadable and the output type adapts accordingly. If the inputs are of type BOOL, the output will be of type BOOL. Otherwise, if an input signal is analogue, the output will be of the same data type as the analogue signal. The inputs are connected so that the topmost input is the minuend and the bottom-most input is the subtrahend.</p> <p>For example, to perform the subtraction (a-b), a signal is connected to the topmost input and signal b to the bottom-most input. The SUB function block outputs the arithmetic difference of the two input signals.</p>
	The MUL function block represents a mathematical multiplication operation. The block is scalable, but always has one output and at least two inputs. The inputs are loadable, and the output type adapts accordingly. If the inputs are of type BOOL, the output will be of type BOOL. Otherwise, if an input signal is analogue, the output will be of the same data type as the analogue signal. The signals are connected to the function block inputs in any order. The MUL function block outputs the arithmetic product of the input signals.
	<p>The DIV function block represents a mathematical division operation. The block is non-scalable and always has exactly two inputs and one output. The inputs are loadable and the output type adapts accordingly. If the inputs are of type BOOL, the output will be of type BOOL. Otherwise, if an input signal is analogue, the output will be of the same data type as the analogue signal. The inputs are connected so that the topmost input is the dividend and the bottom-most input is the divisor.</p> <p>For example to perform the division a/b, signal a is connected to the topmost input and signal b to the bottom-most input. The DIV function block outputs the arithmetic quotient of the two input signals.</p>



**Table A2-3 Timer IO Tools**

Icon in IO Editor	Description
	<p>The TP function block is a timer function block. The block has two inputs and two outputs. The IN input should be of type BOOL, the PT input can be of either UINT or DECIM type, while the Q output is of type BOOL and ET is of type DECIM.</p> <p>The Timer function block:</p> <ul style="list-style-type: none"> <li>• IN starts the timer</li> <li>• PT sets the time in seconds</li> <li>• ET gives the time in seconds that has elapsed from the last start</li> <li>• Q is an output signal, that is TRUE when the timer is active</li> </ul>
	<p>The TON function block is a timer function block. The block has the same inputs as the TP function block. The difference between TON function block and the TP function block operation is that in the TP function block there is a delay before the Q output state moves to a logical 1 (TRUE) state. Assuming an IN signal of logical 1 (TRUE) level, the ET output counts from 0.00 to PT, upon which the Q output moves to a logical 1 (TRUE) state. When the IN signal drops to a logical 0 (FALSE) level, the Q and ET outputs return to their normal states. If the IN signal is not in a logical 1 (TRUE) state long enough before falling, the ET output does not finish counting and the Q output state does not rise.</p> <p>The Timer function block, ON-delay</p> <ul style="list-style-type: none"> <li>• IN starts the timer</li> <li>• PT sets the time in seconds</li> <li>• ET gives the time in seconds that has elapsed from the last start</li> <li>• Q is an output signal, that is FALSE until the timer has elapsed</li> </ul>
	<p>The TOF function block is a timer function block. The block has the same inputs as the TP function block. In the TOF function block a logical 0 (FALSE) signal applied to the IN pin activates the block. The Q output rises to a logical 1 (TRUE) state and the ET output counts.</p> <p>The Timer function block, OFF-delay, BOOL / TIME</p> <ul style="list-style-type: none"> <li>• IN starts the timer</li> <li>• PT sets the time in seconds</li> <li>• ET gives the time in seconds that has elapsed from the last start</li> <li>• Q is an output signal, that is TRUE until the timer has elapsed</li> </ul>

**Table A2-4 Wave IO Tools**

Icon in IO Editor	Description
	<p>The SIN function block is a sine-wave generator function block. The block has two inputs, whose input signals should preferably be of DECIM data type, and one output that is always connected to either an AO component or a MUX function block (in which case the MUX output must be connected to an AO component).</p> <p>The SIN function block generates the analogue sine-wave signals that are sent to the IED through the AO connectors. The Ampl input determines the amplitude of the signal, whereas the Phase input determines the phase angle. The frequency of all SIN components in the configuration is assigned to the function block named "Freq". A value of 100 applied to the amplitude input generates the maximum output voltage (<math>\pm 10</math> V) the SIM600 is capable of. The value applied to the phase input is interpreted as degrees. Please note that changing the amplitude or phase values may cause the signal to stray out of phase temporarily while the values are changed. This may be encountered as unwanted IED protection trips or alarms.</p>

## Index

### A

adding GUI objects .....	16
adding inputs .....	23
adding IO objects .....	19

### C

compiling a configuration .....	25
connecting IO objects .....	20

### D

deleting GUI objects .....	19
deleting IO objects and connections .....	23

### H

hardware .....	11
----------------	----

### I

inverting .....	22
-----------------	----

### L

loading background image .....	15
--------------------------------	----

### M

mapping IO names .....	22
modifying GUI object properties .....	16
modifying IO object properties .....	21

### R

removing background image .....	16
removing inputs .....	24

### S

SIM600 Editor .....	11
SIM600 overview .....	9

Configuration Manual

---

supported data types .....	13
supported function blocks .....	14
supported image formats .....	15
supported objects .....	14



**ABB Oy**

Distribution Automation  
P.O. Box 699  
FI-65101 VAASA  
FINLAND  
Tel. +358 10 22 11  
Fax. +358 10 22 1094

[www.abb.com/substationautomation](http://www.abb.com/substationautomation)

**ABB Inc.**

655 Century Point  
Lake Mary, Florida 32746  
USA  
Tel: +1 407 732 2000  
Fax: +1 407 732 2335