Mircom™

OpenBAS | BUILDING AUTOMATION SYSTEM

eZ HVAC and Building Automation Wizard

LT-6631 Rev. 0
June 2017

HVAC and building automation Wizard

The eZ App programing style for HVAC and building automation is a step forward in the way a solution to automate buildings is done.

It is designed from the ground up with the application in mind, reliving the user from the tedious job of having to know the details to accomplish the task of automating mechanical processes that are common for buildings.

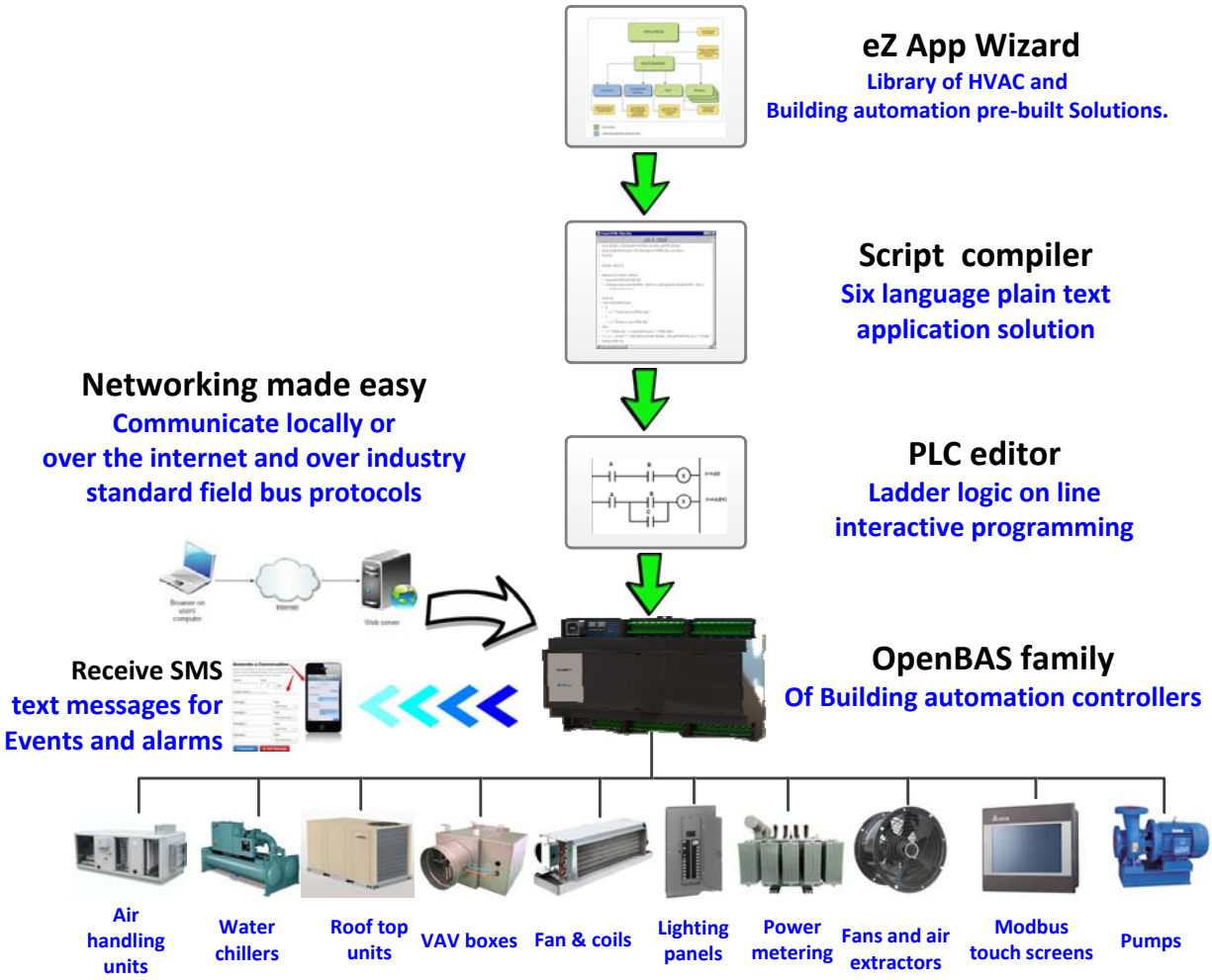
At the bottom of the solution is a rock solid **OpenBAS** hardware that is field proven for years that has a rich set of peripherals to get tasks in building automation done in a fast, secure and economical way.

Layered on top of a family of controllers, that runs interactively ladder PLC instructions. This new layer of multi-language script compiler gets its input from a pre-built and tested solution templates library that is automatically generated by the App Wizard.

An eZ one step communication port and solution configurator eases the tedious task of having to start from scratch every time.

As shown on the image on next page the solution provides an entry level building automation designers with little experience to become proficient and be able to finish an automation solution in minutes, rather days.

With the added feature that experienced users will not lose control of a powerful set of tools that have been used for years.

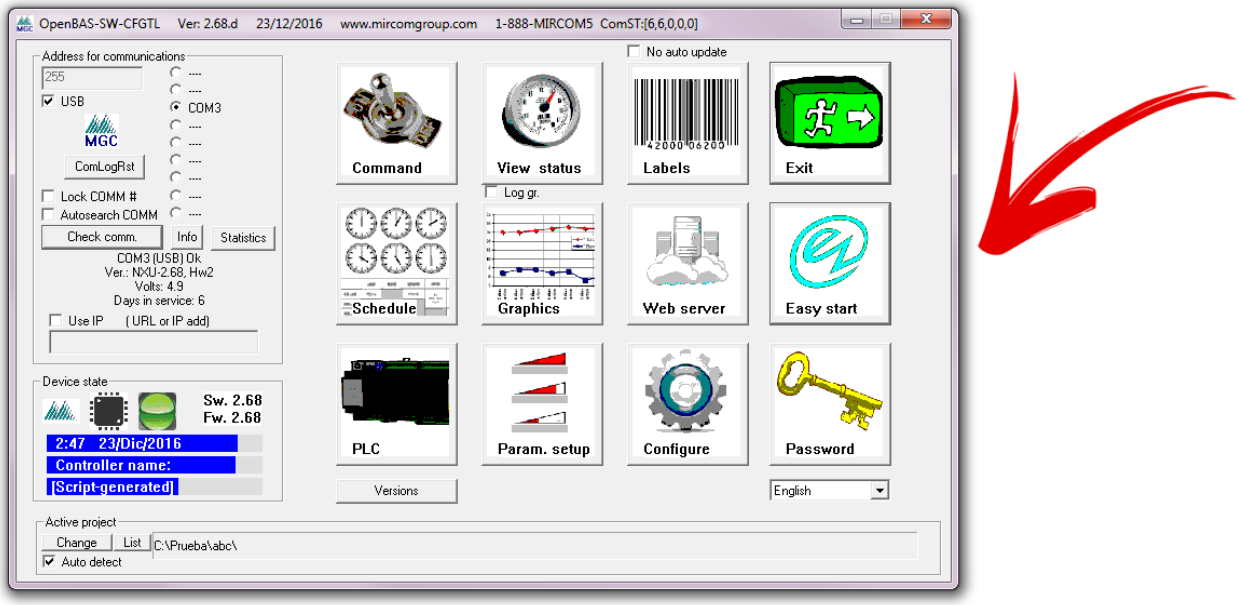


As seen on the image above, the layered programming interface takes the solution from concept to final application in a matter of minutes.

Everything starts at the top, in the **eZ** Application Wizard, where choosing from a library of prebuilt solutions takes the user by the hand with a series of questions. When everything is answered with the single push of a button a script is generated in the language preferred by the user.

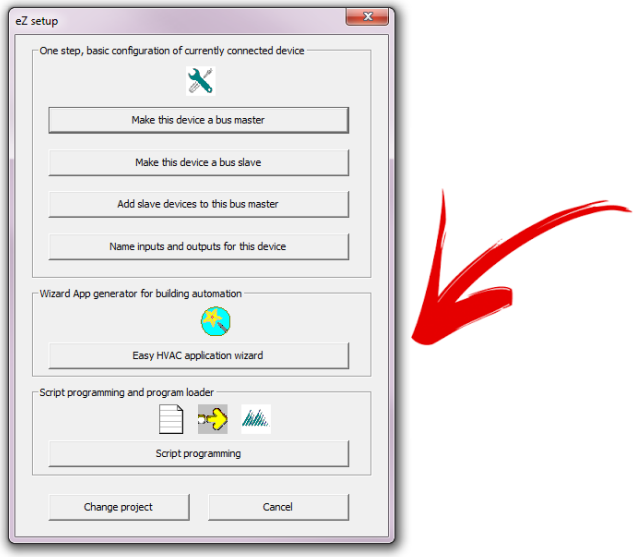
Next it automatically jumps to the **Script compiler,** where the user can optionally hand tune the script, or simply compile and load the solution.

As easy as it sounds, just like a piece of hot apple pie.



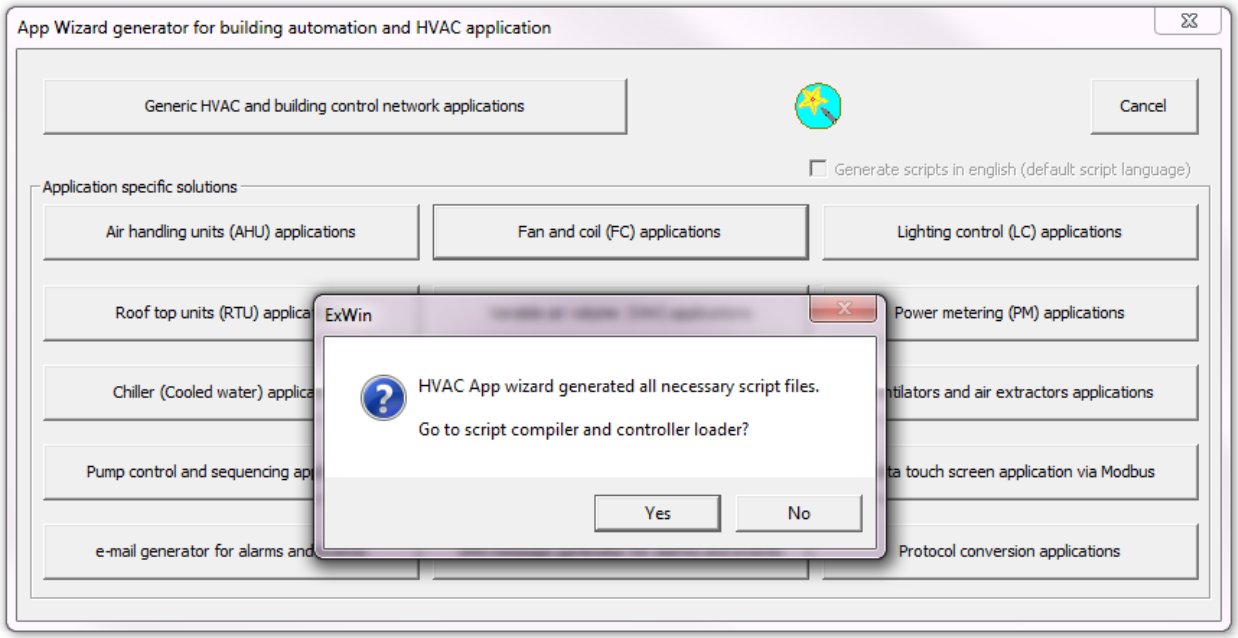
Everything starts here by selecting the **eZ** Start button; this will take you to the newly added screens that let you choose between a one button communication port setup for industry standard protocols as either a bus master or slave.

Next is the **eZ** Application Wizard button that will take us to the library template builder shown on the next screen.

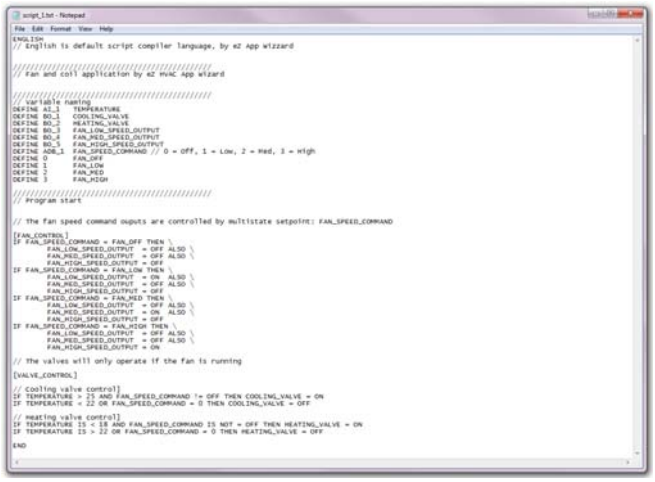


Finally at the end is a direct access to the Script compiler where the programmer can edit, compile and download the application.

So now, because this should be as easy as *1, 2, and 3* let's get started by building a typical application using the **eZ** Application Wizard.

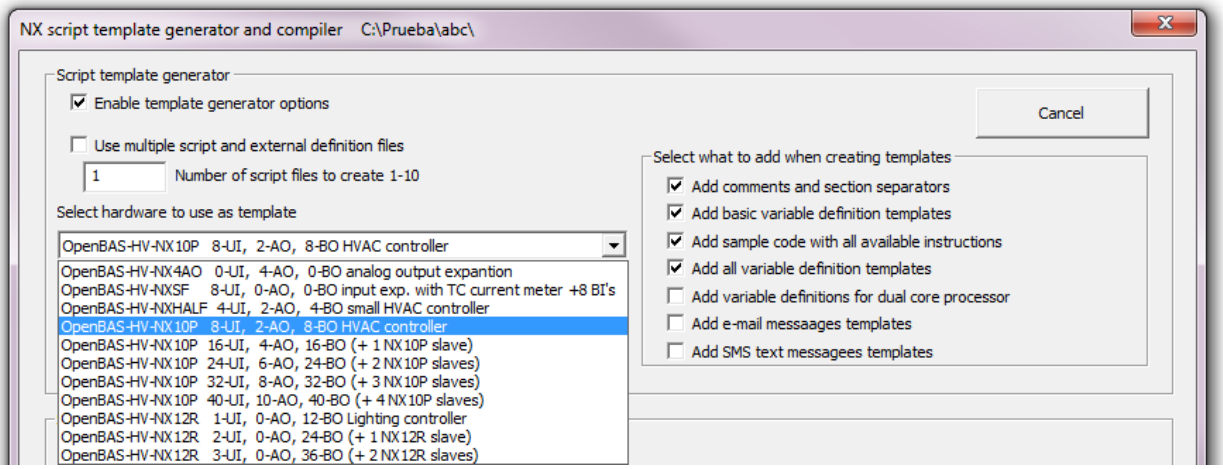


By selecting from the pre-built libraries **(Step 1)** and answering some questions, now with this information the **eZ** Application Wizard generates all the necessary files **(Step 2)** and asks whether you want to compile and load the solution to the controller **(Step 3)**.



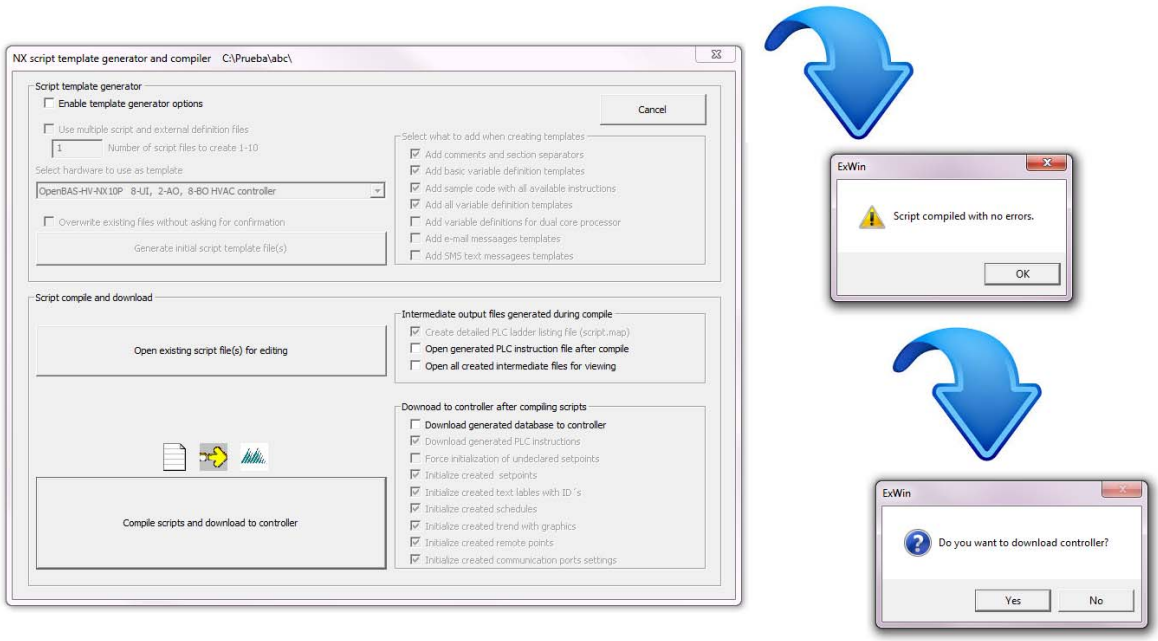
It opens the generated solution for you to examine and optionally hand tune names, logic et. Here is an example of a typically generated solution.

By following the **1, 2, 3** step solution, we ended up in the **"Script compiler"**. For experienced programmers this is heaven, as you can use the integrated **"Script template generator"** for the **OpenBAS** family of controllers and use pre-built templates that already include all the features that each controller has to offer with a single **"push and create"** feature as a good starting point. Call it a handicap if you want.

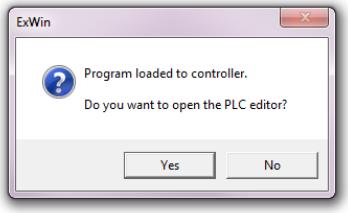


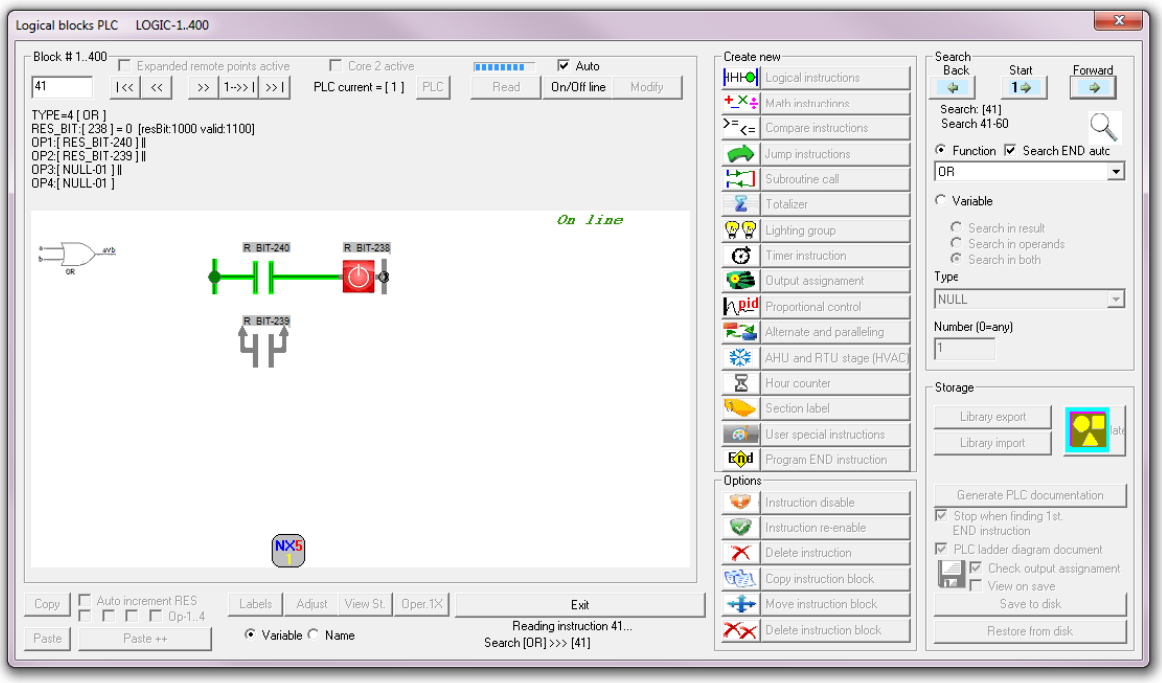
But because I don´t want to bore you with **"advanced stuff"** right now, let's focus on the 1, 2, 3 solution.

1. **Select from library of pre-built applications** **DONE !**
2. **Generate the solution** **DONE !**
3. **Compile and download the solution** **DONE !**



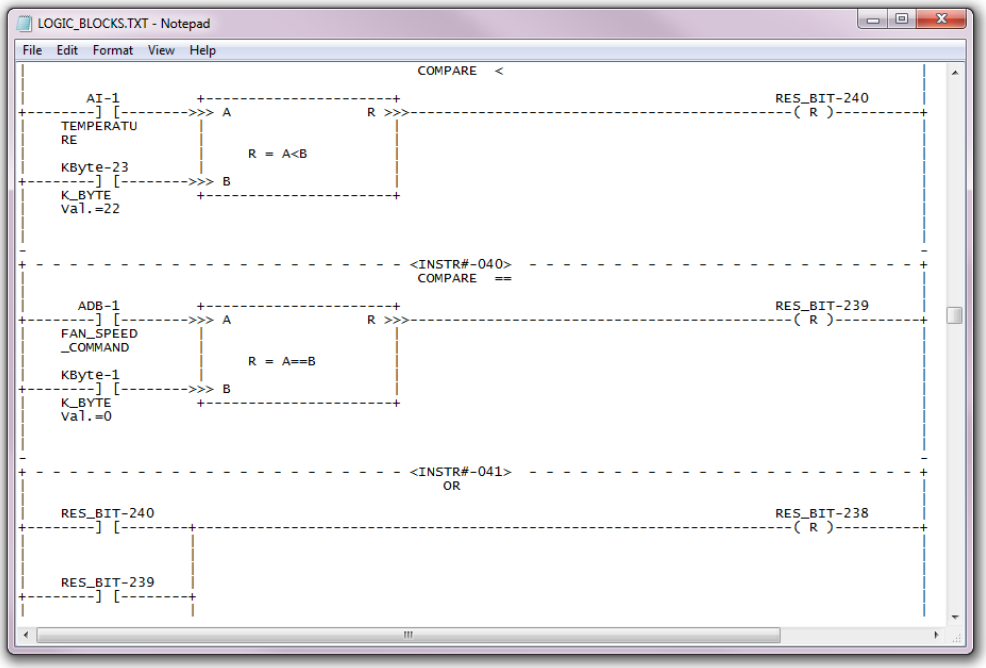
So now we are done, loading the application takes between a couple of seconds to a pair of minutes depending on the length of the program and the generated database objects, and finally a last question, the **eZ** Application Wizard asks if we want to view the loaded solution in the PLC editor and takes us directly there.





As I promised on the beginning, is like a slice of fresh backed apple pie.

Finally in the **"Ladder diagram PLC editor"** you can use the standard tools that have been available for more than half a decade to edit, document and fine tune the solution.



As you can see the newly added tools simplify building automation. Just around the corner is the **"Network builder"** that with "drag and drop" will allow you to **"sketch or draw"** your networking solution and with the push of a button create a full networking project with ready to download solution.

This is how typically a **"Script"** looks like, remember it can be written in English, French, Spanish, Italian, German and Dutch, and the integrated Tower of Babel language converter will take the tedious and nauseous work of language translation away from you.

script_1.txt - Notepad

File Edit Format View Help

```
ENGLISH
// English is default script compiler language, by eZ App Wizzard


////////////////////////////////////////////////
// Fan and coil application by eZ HVAC App Wizard


//////////////////////////////////////////////
// Variable naming
DEFINE AI_1    TEMPERATURE
DEFINE BO_1    COOLING_VALVE
DEFINE BO_2    HEATING_VALVE
DEFINE BO_3    FAN_LOW_SPEED_OUTPUT
DEFINE BO_4    FAN_MED_SPEED_OUTPUT
DEFINE BO_5    FAN_HIGH_SPEED_OUTPUT
DEFINE ADB_1   FAN_SPEED_COMMAND // 0 = Off, 1 = Low, 2 = Med, 3 = High
DEFINE 0       FAN_OFF
DEFINE 1       FAN_LOW
DEFINE 2       FAN_MED
DEFINE 3       FAN_HIGH

//////////////////////////////////////////////
// Program start


// The fan speed command ouputs are controlled by multistate setpoint: FAN_SPEED_COMMAND

[FAN_CONTROL]
IF FAN_SPEED_COMMAND = FAN_OFF THEN \
        FAN_LOW_SPEED_OUTPUT  = OFF ALSO \
        FAN_MED_SPEED_OUTPUT  = OFF ALSO \
        FAN_HIGH_SPEED_OUTPUT = OFF
IF FAN_SPEED_COMMAND = FAN_LOW THEN \
        FAN_LOW_SPEED_OUTPUT  = ON  ALSO \
        FAN_MED_SPEED_OUTPUT  = OFF ALSO \
        FAN_HIGH_SPEED_OUTPUT = OFF
IF FAN_SPEED_COMMAND = FAN_MED THEN \
        FAN_LOW_SPEED_OUTPUT  = OFF ALSO \
        FAN_MED_SPEED_OUTPUT  = ON  ALSO \
        FAN_HIGH_SPEED_OUTPUT = OFF
IF FAN_SPEED_COMMAND = FAN_HIGH THEN \
        FAN_LOW_SPEED_OUTPUT  = OFF ALSO \
        FAN_MED_SPEED_OUTPUT  = OFF ALSO \
        FAN_HIGH_SPEED_OUTPUT = ON

// The valves will only operate if the fan is running

[VALVE_CONTROL]

// Cooling valve control]
IF TEMPERATURE > 25 AND FAN_SPEED_COMMAND != OFF THEN COOLING_VALVE = ON
IF TEMPERATURE < 22 OR FAN_SPEED_COMMAND = 0 THEN COOLING_VALVE = OFF

// Heating valve control]
IF TEMPERATURE IS < 18 AND FAN_SPEED_COMMAND IS NOT = OFF THEN HEATING_VALVE = ON
IF TEMPERATURE IS > 22 OR FAN_SPEED_COMMAND = 0 THEN HEATING_VALVE = OFF

END
```

OpenBAS

Script compiler

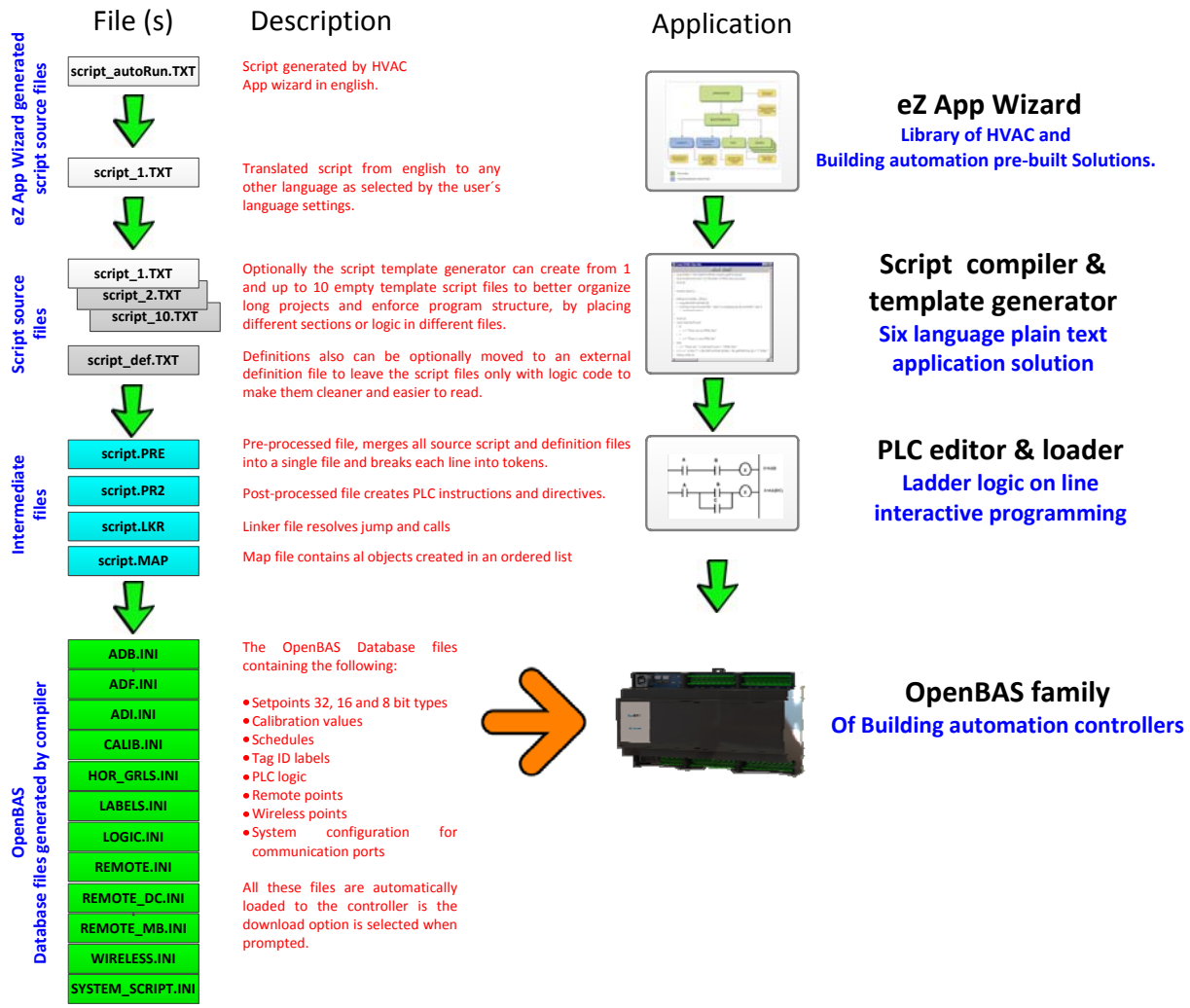
User's guide

Script compiler user's guide

The script compiler is a language that converts plain text into instructions that can be executed by the OpenBAS controllers. This language follows a syntax that will be described in the following sections and can be entered in the following ways:

- Manually following the syntax described in this document.
- Automatically generated by the HVAC AppWizard.
- Using templates generated by the script compiler template generation.

Regardless of the way the script was initially generated it can always be edited using any standard text editor as long as it is saved as PLAIN TEXT with no formatting such as added in Word or other rich text editors. Below is a diagram describing the creation of the script files and the compilation process.

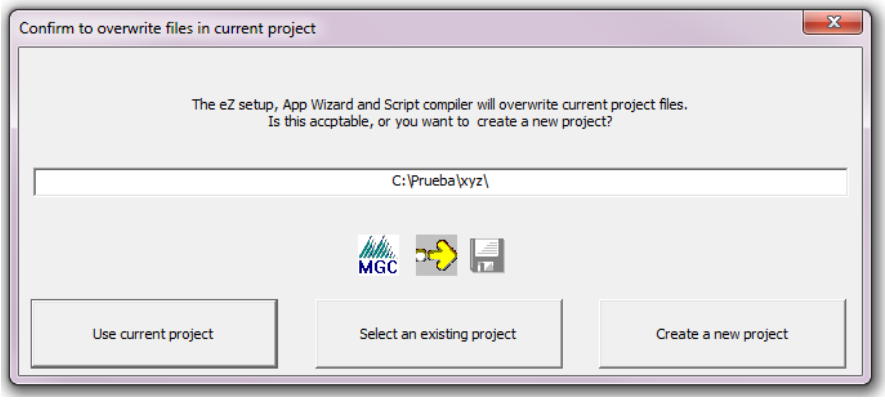


As can be seen, the flow is as follows; the **eZ App wizard** generates based on the user preferences a pre-built and pre-tested solution that can be directly compiled and loaded to the controller. It does so by generating a pair of files:

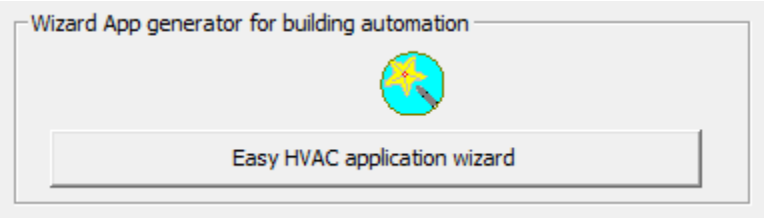
script_autoRun.txt which is a script file with the user selected options that is generated in the script´s native language which is English.
From here the language translator generates the **script_1.txt** file that if the user selects will be translated to the user´s preferred language. Only keywords are translated, the names of the variables are presented to the user prior to generating the scripts in English as he or she can manually translate their names to something meaningful for the application in the desired language.

When the eZ wizard is invoked for the first time, will ask if we want to use:

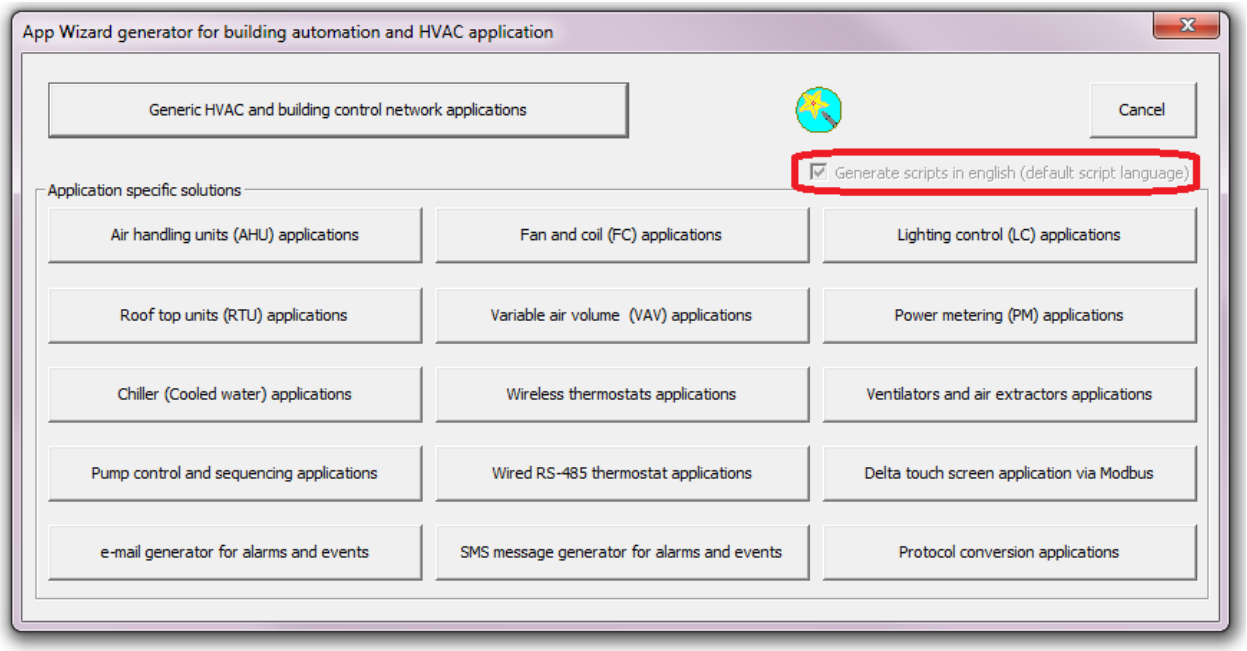
- The current project
- An already existing project
- A new project



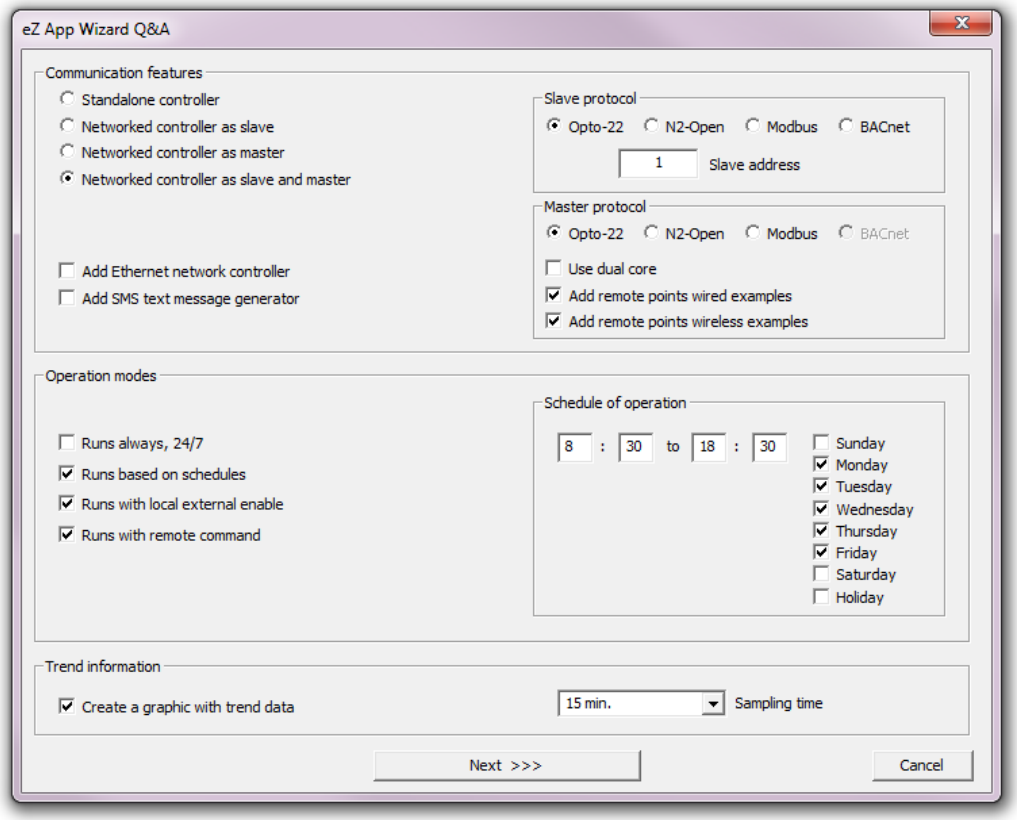
Once the selection is done, select the HVAC application wizard to start creating the application.

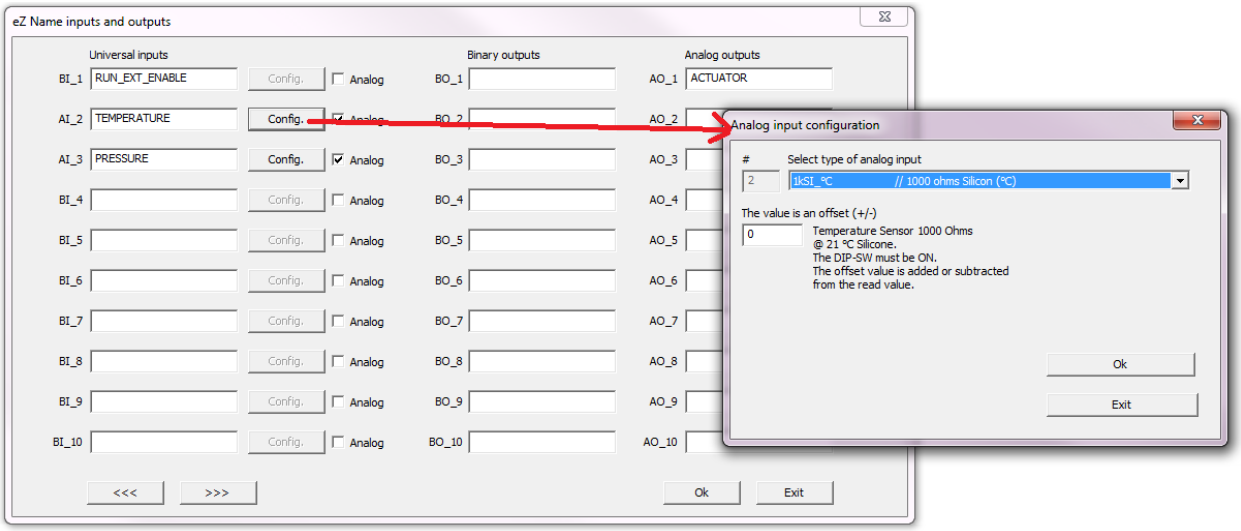


Here a screen showing the different type of pre-built applications will appear. From this screen the user can select if the languages to be created will be translated by checking the language select checkbox, if English is selected the checkbox is disabled and no translation will take place.

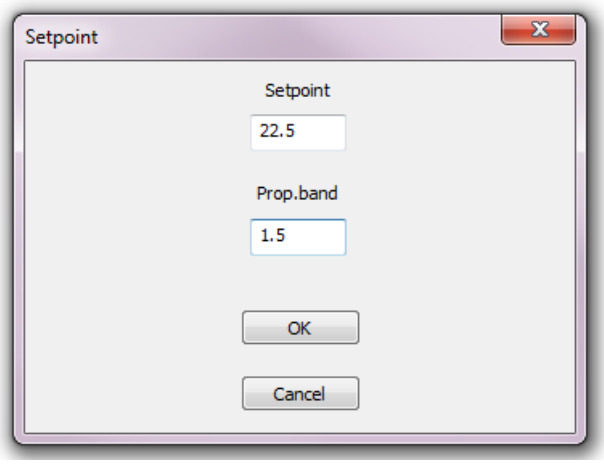


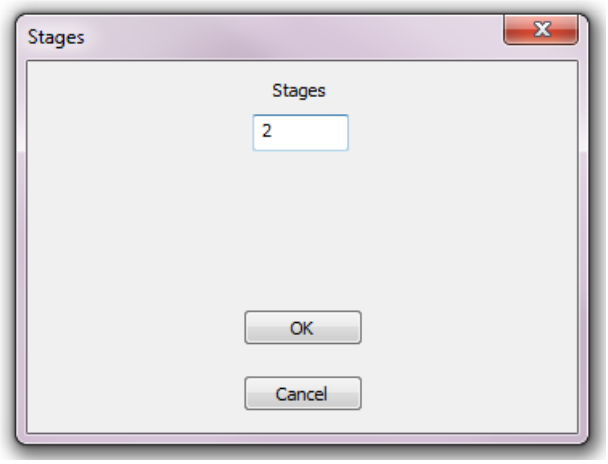
Once an application is selected, a couple of screens will sequentially pop up showing first, the communication features, then the operation modes and finally the trend information.





In this dialog box the I/O names can be edited, as well as the universal inputs can be selected as analog or digital, and their default configuration can be set.





Optionally depending on the application selected, more screens to fill parameters such as; set points, number of stages etc., will keep showing until all required data is gathered, at which point the script will be created and optionally translated.

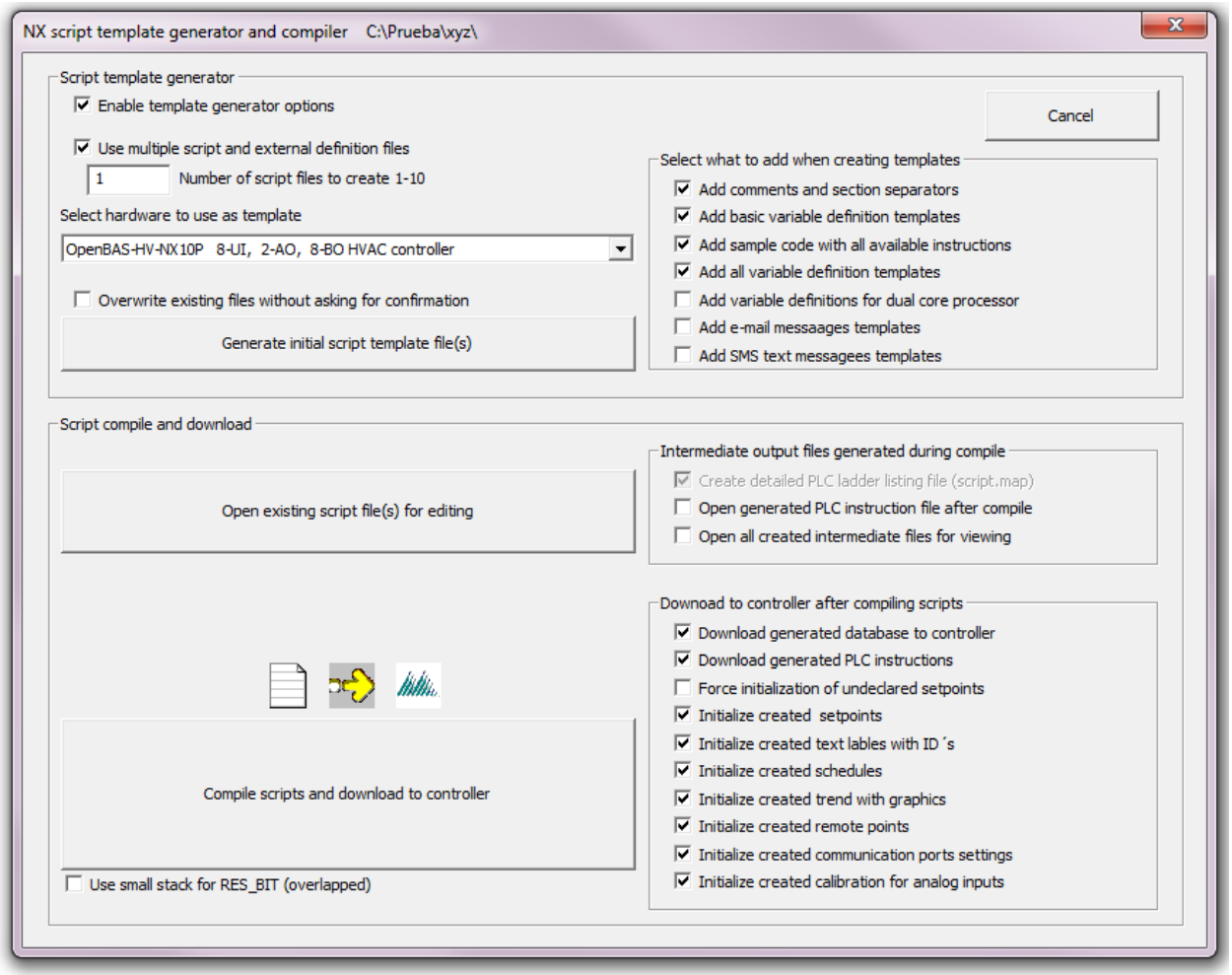
After this step, the **script compiler** is automatically invoked. All the previous steps of the **eZ App wizard** can be skipped if the user wants to manually create the script files or copy them from a previous project and manually modify them. Optionally also the script compiler has a script template generator that can generate one or multiple template files to better organize the whole project. Also an external definition file can be added to the project, so definitions common to all script files can be located there. This way the script files will be less cluttered and contain only logic sentences describing the program.

The script compiler needs at least one source file that must be named **script_1.txt.** If additional files are added to the project they must be named sequentially **script_2.txt** and up to **script_10.txt**. If the external definition file is included to the project, it must be named **script_def.txt**.

If the script files are in a language other than English, **the first line** of either **script_def.txt** if it exists or **script_1.txt** must contain the language directive or an error will be generated because the keyword´s language will be undefined. These language directives are as listed in the following table:

	GB	FR	ES	DE	IT	NL
Language selector	**ENGLISH**	**FRENCH**	**SPANISH**	**GERMAN**	**ITALIAN**	**DUTCH**
Alternate		**FRANÇAIS**	**ESPAÑOL**	**DEUTSCH**	**ITALIANO**	**NEDERLANDS**

Both the script compiler section as well as the template generator section are depicted on the picture below. By default the template generator section is disabled, but can be enabled by checking the "Enable template generator" checkbox.



If the user wants to edit the existing file(s) there is a button that opens all the existing script and the optionally definition files at once.

The checkboxes to the right of this button allow the intermediate generated files to be opened after the compile process for viewing. They usually are not shown if compile goes without errors, if however any syntax of declaration error exists, the intermediate files that have already been processed and created will be opened.

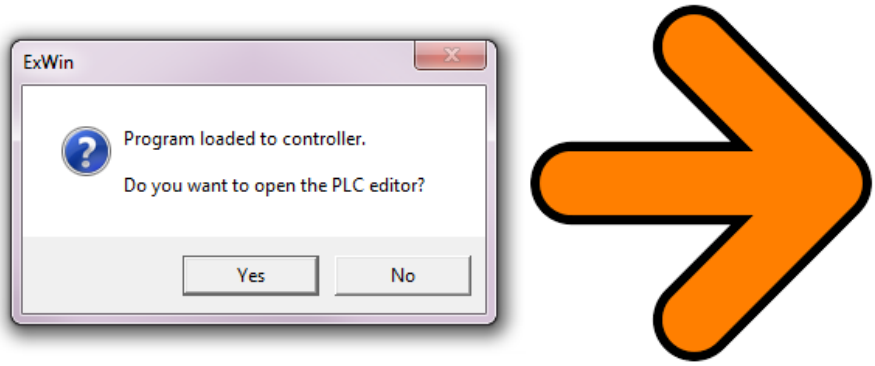
At the point where an error is found during the compiling process, an error text will be added to the generated intermediate file(s) giving an error code and a brief explanation of the kind of error found. At the end of this document, in the appendix named "**compiler errors**", a comprehensive list with all the errors and their possible solutions can be found.

The "**Compile scripts and download to controller**" button does just what its name implies, compiles the script, and by generating the intermediate files one after the other at the end creates an OpenBAS database that will be loaded to the controller automatically.

To the right of this compile button, a group of checkboxes allows the user to manually select what is to be downloaded. By default all created database objects by the compile process are downloaded to the controller.

This compile and download process takes between a couple of seconds to a couple of minutes depending on the size of the program and the created database objects.

Once the whole process is completed, the script compiler asks if you want to proceed to the **PLC editor** to view and optionally fine tune and debug the program online.



The intermediate **LINK** and **MAP** files contain the source code as well as the generated PLC instructions so the user can refer to each section of his or her source file and see what was created. The next section of this manual will explain the script compiler syntax.

Before proceeding to the script compiler in detail the following tables describe the database objects and the keywords that can be used in each of the supported languages.

Database objects table.

Database object	Ranges	GB	FR	ES	DE	IT	NL
Analog inputs	1-40	AI_x	EA_x	EA_x	AE_x	IA_x	AI_x
Binary inputs	1-40	BI_x	EB_x	EB_x	BE_x	IB_x	BI_x
Analog outputs	1-10	AO_x	SA_x	SA_x	AA_x	UA_x	AU_x
Binary outputs	1-60	BO_x	SB_x	SB_x	BA_x	UB_x	BU_x
EEprom 32 bits	1-100	ADF_x					
EEprom 16 bits	1-100	ADI_x					
EEprom 8 bits	1-100	ADB_x					
Lighting groups	1-20	LG_x					
RAM result 1 bit	1-255	RES_BIT_x					
RAM result 32 bit	´1-40	RES_FLT_x					
Timers	´1-16	TMR_x TIMER_x					
Remote registers	1-50	RMT_x REMOTE_x					
Remote reg. exp	51-255	RMT_RES_x					
Integer constants	0-253	KBYT_x KBYTE_x					
Float constants	any float value	K_FLT_x K_FLOAT_x					

Note that the English versions of the hardware I/O will always be available, so are the keywords, so a user writing for example the script in French can freely use English and French keywords and database objects intermixed.

On the next six pages the keyword list for the different languages is shown. Note that accents and other nonstandard characters have been replaced with standard non accented ASCII characters to make language translation easy to work with.

All words are case insensitive so the user can write in capital or non-capital or mixed characters. For some keywords and database objects there are shorthand versions of some words to make script programming easier.

Keywords table 1 of 6

Keyword	GB	FR	ES
KWD_NULL	NULL	NULL	NULL
KWD_IF	IF	SI	SI
KWD_ELSE	ELSE	AUTRE	CASO_CONTRARIO
KWD_THEN	THEN	PUIS	ENTONCES
KWD_JUMP	JUMP	SAUT	BRINCA
KWD_CALL	CALL	APPEL	LLAMA
KWD_SCRIPT_1	SCRIPT_1	SCRIPT_1	SCRIPT_1
KWD_SCRIPT_2	SCRIPT_2	SCRIPT_2	SCRIPT_2
KWD_SCRIPT_3	SCRIPT_3	SCRIPT_3	SCRIPT_3
KWD_SCRIPT_4	SCRIPT_4	SCRIPT_4	SCRIPT_4
KWD_SCRIPT_5	SCRIPT_5	SCRIPT_5	SCRIPT_5
KWD_SCRIPT_6	SCRIPT_6	SCRIPT_6	SCRIPT_6
KWD_SCRIPT_7	SCRIPT_7	SCRIPT_7	SCRIPT_7
KWD_SCRIPT_8	SCRIPT_8	SCRIPT_8	SCRIPT_8
KWD_SCRIPT_9	SCRIPT_9	SCRIPT_9	SCRIPT_9
KWD_SCRIPT_10	SCRIPT_10	SCRIPT_10	SCRIPT_10
KWD_SEC	SECONDS	SECONDES	SEGUNDOS
KWD_SEC_1_10	SEC_1_10	SEC_1_10	SEG_1_10
KWD_RUN	RUN	COURIR	CORRE
KWD_LOAD	LOAD	CHARGER	CARGA
KWD_IS	IS	ETRE	ES
KWD_NOT	NOT	NON	NO
KWD_EMRG_STOP	EMRG_STOP	ARRET_D_URGENCE	PARO_EMERG
KWD_CMP_LT	<	<	<
KWD_CMP_LET	<=	<=	<=
KWD_CMP_GT	>	>	>
KWD_CMP_GET	>=	>=	>=
KWD_ASSIGN	=	=	=
KWD_CMP_NOT_EQUAL	!=	!=	!=
KWD_CMP_EQUAL	==	==	==
KWD_ADD	+	+	+
KWD_SUBSTRACT	-	-	-
KWD_MULTIPLY	*	*	*
KWD_DIVIDE	/	/	/
KWD_MIN	MIN	MIN	MIN
KWD_MAX	MAX	MAX	MAX
KWD_AVG	AVG	MOYENNE	PROMEDIO
KWD_ADD_ASGN	+=	+=	+=
KWD_SUB_ASGN	-=	-=	-=
KWD_DIV_ASGN	/=	/=	/=
KWD_MUL_ASGN	*=	*=	*=
KWD_ON	ON	ALLUME	ENCENDIDO
KWD_OFF	OFF	ETEINDRE	APAGADO
KWD_OPEN	OPEN	OUVERT	ABIERTO
KWD_CLOSED	CLOSED	FERME	CERRADO
KWD_TRUE	TRUE	VRAI	VERDADERO
KWD_FALSE	FALSE	FAUX	FALSO
KWD_DEFINE	DEFINE	DEFINIR	DEFINE
KWD_DEF	DEF	DEF	DEF
KWD_TIMER	TIMER	MINUTEUR	TEMPORIZADOR
KWD_OSCILATOR	OSCILATOR	OSCILLATEUR	OSCILADOR
KWD_FREQUENCY	FREQUENCY	FREQUENCE	FRECUENCIA
KWD_AND	AND	ET	Y
KWD_NAND	NAND	ET_INVERSE	Y_NEGADO
KWD_OR	OR	OU	O
KWD_NOR	NOR	OU_INVERSE	O_NEGADO
KWD_XOR	XOR	XOR	O_EXCLUSIVO
KWD_NXOR	NXOR	XOR_INVERSE	O_EXCLUSIVO_NEGADO
KWD_INVERT	INVERT	INVERSER	INVERTIR
KWD_AND_OR	AND_OR	ET_OU	Y_O

Keywords table 2 of 6

Keyword	GB	FR	ES
KWD_AND_NOR	AND_NOR	ET_OU_INVERSE	Y_O_NEGADO
KWD_SET	SET	SET	FIJAR_A_1
KWD_RESET	RESET	RESET	FIJAR_A_0
KWD_INSIDE	INSIDE	INTERIEUR	DENTRO_DE
KWD_OUTSIDE	OUTSIDE	EXTERIEUR	FUERA_DE
KWD_LT_GROUP	LT_GROUP	GROUPE_D_ECLAIRAGE	GRUPO_ILUM
KWD_PROP_CTRL	PROP_CTRL	CONTROLE_PROPORTIONNEL	CONTROL_PROP
KWD_TOTALIZE	TOTALIZE	TOTALISATEUR	TOTALIZADOR
KWD_HOUR_CNT	HOUR_COUNTER	COMPTEUR_D_HEURES	HOROMETRO
KWD_HVAC_STAGE	HVAC_STAGE	ETAPE_HVAC	ETAPA_HVAC
KWD_ALTERNATE	ALTERNATE	ALTERNER	ALTERNADO
KWD_SCHEDULE	SCHEDULE	CALENDRIER	HORARIO
KWD_TREND	TREND	TENDANCE	TENDENCIA
KWD_REMOTE	REMOTE	ELOIGNE	REMOTO
KWD_WIRELESS_LINK	WIRELESS_LINK	LIAISON_SANS_FIL	ENLACE_INALAMBRICO
KWD_COMM	COMM	COMM	COMM
KWD_E_MAIL	E_MAIL	E_MAIL	E_MAIL
KWD_SMS_TEXT	SMS_TEXT	TEXTE_SMS	SMS_TEXTO
KWD_ALSO	ALSO	AUSSI	TAMBIEN
KWD_WITH	WITH	AVEC	CON
KWD_ADD_T	ADD	ADDITION	SUMA
KWD_SUBSTRACT_T	SUBSTRACT	SOUSTRACTION	RESTA
KWD_MULTIPLY_T	MULTIPLY	MULTIPLICATION	MULTIPLICACION
KWD_DIVIDE_T	DIVIDE	DIVIDE	DIVISION
KWD_START	START	DEMARRER	ARRANQUE
KWD_STOP	STOP	ARRETEZ	PARO
KWD_SUB_BEGIN	SUB_BEGIN	SUB_DEMARRER	SUB_INICIO
KWD_SUB_END	SUB_END	SUB_FIN	SUB_FIN
KWD_END	END	FIN	FIN
KWD_HYSTERESIS	HYSTERESIS	HYSTERESE	HISTERESIS
KWD_START_PLC	START_PLC	DEPART_PLC	INICIO_PLC
KWD_DAY	DAY	JOUR	DIA
KWD_WEEK	WEEK	SEMAINE	SEMANA
KWD_MONTH	MONTH	MOIS	MES
KWD_PERIOD	PERIOD	PERIODE	PERIODO
KWD_VALUE	VALUE	VALEUR	VALOR
KWD_INITIALIZE	INITIALIZE	INITIALISER	INICIALIZA
KWD_PV	PV	VP	VP
KWD_SP	SP	CONSIGNE	P_AJ
KWD_PB	PB	B_PROP	B_PROP
KWD_INTEG	INTEG	INTEG	INTEG
KWD_EEPROM	EEPROM	EEPROM	EEPROM
KWD_PLC_COUNTER	PLC_COUNTER	AJUSTER_INSTRUCTION_PLC	AJUSTE_INSTRUCCION_PLC
KWD_ON_CHANGE	ON_CHANGE	SUR_LE_CHANGEMENT	AL_CAMBIAR
KWD_LAST_PERIOD	LAST_PERIOD	DERNIERE_PERIODE	ULTIMO_PERIODO
KWD_PARTIAL_KW	PARTIAL_KW	PARTIELLE_KW	PARCIAL_KW
KWD_PARTIAL_ACC	PARTIAL_ACC	PARTIELLE_ACCUM	PARCIAL_ACUM
KWD_SAMPLE_COUNT	SAMPLE_COUNT	NUMERO_ECHANTILLON	NUMERO_MUESTRAS
KWD_ALT_OUTPUT	ALT_OUTPUT	ALT_SORTIE	ALT_SALIDA
KWD_ALT_STAGES	ALT_STAGES	ALT_ETAPES	ALT_ETAPAS
KWD_ALT_LEADER	ALT_LEADER	ALT_LEADER	ALT_LIDER
KWD_ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC
KWD_ALT_TMR_NEXT_STAGE	ALT_TMR_NEXT_STAGE	ALT_TM_SUIV_ETAPE	ALT_TM_SIG_ETAPA
KWD_ALT_TMR_ALARM	ALT_TMR_ALARM	ALT_TM_ALARME	ALT_TM_ALARMA
KWD_ALT_EXT_ENABLE	ALT_EXT_ENABLE	ALT_ACTIV_EXT	ALT_HABIL_EXT
KWD_ALT_FEEDBACK	ALT_FEEDBACK	ALT_RETOUR	ALT_RETROALIM
KWD_ALT_PARALLEL	ALT_PARALLEL	ALT_PARALLELE	ALT_SIMULTANEO
KWD_PARTIAL_COUNT	PARTIAL_COUNT	COMPTE_PARTIEL	CUENTA_PARCIAL
KWD_COOLING	COOLING	REFROIDISSEMENT	ENFRIAMIENTO
KWD_HEATING	HEATING	CHAUFFAGE	CALEFACCION
KWD_TMR_MINIMUM_ON	TMR_MINIMUM_ON	TMR_MIN_ALLUMER	TMR_MINIMO_ENC

Keywords table 3 of 6

Keyword	GB	FR	ES
KWD_TMR_MINIMUM_OFF	TMR_MINIMUM_OFF	TMR_MIN_ARRET	TMR_MINIMO_APAG
KWD_TIMER_INTERSTAGE	TMR_INTERSTAGE	TMR_INTERETAPES	TMR_INTERETAPAS
KWD_STAGE_RUN	STAGE_RUN	VALIDACION_ETAPE	PERMISIVO_ETAPA
KWD_INTERVAL_MINUTES	INTERVAL_MINUTES	INTERVALLE_MINUTES	INTERVALO_MINUTOS
KWD_READ_COIL	READ_COIL	READ_COIL	READ_COIL
KWD_READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS
KWD_READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER
KWD_READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER
KWD_ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE
KWD_BINARY_VALUE	BINARY_VALUE	BINARY_VALUE	BINARY_VALUE
KWD_NX_SLAVE	NX_SLAVE	NX_ESCLAVE	NX_ESCLAVO
KWD_WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C
KWD_WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F
KWD_WLS_REL_HUM	WLS_REL_HUM	WLS_HUM_REL	WLS_HUM_REL
KWD_WLS_MODE	WLS_MODE	WLS_MODE	WLS_MODO
KWD_WLS_FAN_SPEED	WLS_FAN_SPEED	WLS_VEL_VENTILATEUR	WLS_VEL_VENTILADOR
KWD_WLS_KEYBOARD	WLS_KEYBOARD	WLS_CLAVIER	WLS_TECLADO
KWD_WLS_SP_TEMP	WLS_SP_TEMP	WLS_PC_TEMP	WLS_PA_TEMP
KWD_WLS_SP_HUM	WLS_SP_HUM	WLS_PC_HUM	WLS_PA_HUM
KWD_WLS_SP_T1	WLS_SP_T1	WLS_PC_T1	WLS_PA_T1
KWD_WLS_SP_PB	WLS_SP_PB	WLS_PC_BP	WLS_PA_BP
KWD_WLS_SP_UNOCC	WLS_SP_UNOCC	WLS_PC_INOCCUPE	WLS_PA_DESOC
KWD_WLS_BATTERY_VOLTAGE	WLS_BATTERY_VOLTAGE	WLS_VOLTAGE_BATTERIE	WLS_VOLTAJE_BATERIA
KWD_WLS_AUX_INP	WLS_AUX_INP	WLS_ENT_AUX	WLS_ENT_AUX
KWD_WLS_LINK_TMR	WLS_LINK_TMR	WLS_TM_LIEN	WLS_TM_ENLACE
KWD_WLS_SEC_LINK_LOST	WLS_SEC_LINK_LOST	WLS_SEC_SANS_LIEN	WLS_SEC_SIN_ENLACE
KWD_CONTROLLER_NAME	CONTROLLER_NAME	NOM_CONTROLEUR	NOMBRE_CONTROLADOR
KWD_WLS_GROUP	WLS_GROUP	WLS_GRUPE	WLS_GRUPO
KWD_WLS_ADDRESS	WLS_ADDRESS	WLS_ADDRESSE	WLS_DIRECCION
KWD_DEVICE_ADDRESS	DEVICE_ADDRESS	ADDRESSE_COM	DIRECCION_COM
KWD_PARITY_NONE	PARITY_NONE	PARITE_SANS	PARIDAD_SIN
KWD_PARITY_ODD	PARITY_ODD	PARITE_IMPAIRE	PARIDAD_NON
KWD_PARITY_EVEN	PARITY_EVEN	PARITE_PAIRE	PARIDAD_PAR
KWD_STOP_BIT_0	STOP_BIT_0	BIT_D_ARRET_0	BIT_PARO_0
KWD_STOP_BIT_1	STOP_BIT_1	BIT_D_ARRET_1	BIT_PARO_1
KWD_PROTOCOL_ASCII_TERMINAL	PROTOCOL_ASCII_TERMINAL	PROTOCOLE_ASCII_TERMINAL	PROTOCOLO_ASCII_TERMINAL
KWD_PROTOCOL_OPTO_22_SLAVE	PROTOCOL_OPTO_22_SLAVE	PROTOCOLE_OPTO_22_ESCLAVE	PROTOCOLO_OPTO_22_ESCLAVO
KWD_PROTOCOL_N2_OPEN_SLAVE	PROTOCOL_N2_OPEN_SLAVE	PROTOCOLE_N2_OPEN_ESCLAVE	PROTOCOLO_N2_OPEN_ESCLAVO
KWD_PROTOCOL_MODBUS_SLAVE	PROTOCOL_MODBUS_SLAVE	PROTOCOLE_MODBUS_ESCLAVE	PROTOCOLO_MODBUS_ESCLAVO
KWD_PROTOCOL_BANCET_MSTP	PROTOCOL_BANCET_MSTP	PROTOCOLE_BANCET_MSTP	PROTOCOLO_BANCET_MSTP
KWD_PROTOCOL_O22_MASTER	PROTOCOL_OPTO22_MASTER	PROTOCOLE_OPTO22_MAITRE	PROTOCOLO_OPTO22_MAESTRO
KWD_PROTOCOL_MODBUS_MASTER	PROTOCOL_MODBUS_MASTER	PROTOCOLE_MODBUS_MAITRE	PROTOCOLO_MODBUS_MAESTRO
KWD_BAUD_RATE	BAUD_RATE	BAUD	BAUDIOS
KWD_AI_CONFIG	AI_CONFIGURATION	EA_CONFIGURATION	EA_CONFIGURACION
KWD_AI_CALIBRATION	AI_CALIBRATION	EA_ETALONNAGE	EA_CALIBRACION
KWD_SCRIPT_LAST	SCRIPT_END	SCRIPT_END	SCRIPT_END

Keywords table 4 of 6

Keyword	DE	IT	NL
KWD_NULL	NULL	NULL	NULL
KWD_IF	OB	NEL_CASO_QUE	ALS
KWD_ELSE	SONST	ALTRIMENTI	ANDERS
KWD_THEN	DANN	ALORA	DAN
KWD_JUMP	SPRING	SALTARE	SPRING
KWD_CALL	ANRUF	CHIAMATA	OPROEP
KWD_SCRIPT_1	SCRIPT_1	SCRIPT_1	SCRIPT_1
KWD_SCRIPT_2	SCRIPT_2	SCRIPT_2	SCRIPT_2
KWD_SCRIPT_3	SCRIPT_3	SCRIPT_3	SCRIPT_3
KWD_SCRIPT_4	SCRIPT_4	SCRIPT_4	SCRIPT_4
KWD_SCRIPT_5	SCRIPT_5	SCRIPT_5	SCRIPT_5
KWD_SCRIPT_6	SCRIPT_6	SCRIPT_6	SCRIPT_6
KWD_SCRIPT_7	SCRIPT_7	SCRIPT_7	SCRIPT_7
KWD_SCRIPT_8	SCRIPT_8	SCRIPT_8	SCRIPT_8
KWD_SCRIPT_9	SCRIPT_9	SCRIPT_9	SCRIPT_9
KWD_SCRIPT_10	SCRIPT_10	SCRIPT_10	SCRIPT_10
KWD_SEC	SEKUNDEN	SECONDI	SECONDS
KWD_SEC_1_10	SEK_1_10	SEC_1_10	SEC_1_10
KWD_RUN	LAUF	CORRERE	LOPEN
KWD_LOAD	BELAST	CARICARE	LADEN
KWD_IS	IST	SEI	IS
KWD_NOT	NICHT	NO	NIET
KWD_EMRG_STOP	NOT_HALT	ARRESTO_EMERG	NOODSTOP
KWD_CMP_LT	<	<	<
KWD_CMP_LET	<=	<=	<=
KWD_CMP_GT	>	>	>
KWD_CMP_GET	>=	>=	>=
KWD_ASSIGN	=	=	=
KWD_CMP_NOT_EQUAL	!=	!=	!=
KWD_CMP_EQUAL	==	==	==
KWD_ADD	+	+	+
KWD_SUBSTRACT	-	-	-
KWD_MULTIPLY	*	*	*
KWD_DIVIDE	/	/	/
KWD_MIN	MIN	MIN	MIN
KWD_MAX	MAX	MASSIMO	MAX
KWD_AVG	DURCHSCHNITT	MEDIA	GEMIDDELDE
KWD_ADD_ASGN	+=	+=	+=
KWD_SUB_ASGN	-=	-=	-=
KWD_DIV_ASGN	/=	/=	/=
KWD_MUL_ASGN	*=	*=	*=
KWD_ON	EINSCHALT	ACCENDE	AANDOEN
KWD_OFF	AUSSCHALT	SPENGE	UITDOEN
KWD_OPEN	OFFEN	APERTO	OPEN
KWD_CLOSED	GESCHLOSSEN	CHIUSO	DICHT
KWD_TRUE	WAHR	VERO	WAAR
KWD_FALSE	VALS	FALSO	VALS
KWD_DEFINE	DEFINIEREN	DEFINIRE	BEPALEN
KWD_DEF	DEF	DEF	DEF
KWD_TIMER	TIMER	TIMER	TIMER
KWD_OSCILATOR	OSZILLATOR	OSCILLATORE	OSCILLATOR
KWD_FREQUENCY	FREQUENZ	FREQUENZA	FREQUENTIE
KWD_AND	UND	E	EN
KWD_NAND	UND_UMDREHT	E_ROVESCIATO	EN_OMZETTEND
KWD_OR	ODER	O	OF
KWD_NOR	ODER_UMDREHT	O_ROVESCIATO	OF_OMZETTEND
KWD_XOR	ODER_EXCLUSIF	O_EXCLUSIVO	OZ_EXCLUSIEF
KWD_NXOR	ODER_EXCLUSIF_UMDREHT	O_EXCLUSIVO_ROVESCIATO	OZ_EXCLUSIEF_OMZETTEND
KWD_INVERT	UMDREHT	ROVESCIATO	OMZETTEND
KWD_AND_OR	UND_ODER	E_O	EN_OF

Keywords table 5 of 6

Keyword	DE	IT	NL
KWD_AND_NOR	UND_ODER_UMDREHT	E_O_ROVESCIATO	EN_OF_OMZETTEND
KWD_SET	SETZEN	SET	SET
KWD_RESET	ZURUECKSETZEN	RESET	RESET
KWD_INSIDE	INNEN	DENTRO	BINNEN
KWD_OUTSIDE	AUSSEN	FUORI	BUITEN
KWD_LT_GROUP	BELEUCHTUNGSGRUPPE	GRUPO_ILUM	LICHTGROEP
KWD_PROP_CTRL	PROP_BEDIENUNG	CONTROLLO_PROP	PROP_BEDIENING
KWD_TOTALIZE	ZAEHLER	TOTALIZZATORE	TOTALIZATOR
KWD_HOUR_CNT	STUNDEN_ZAEHLER	CONTAORE	URENTELLER
KWD_HVAC_STAGE	HVAC_STUFE	STADIO_HVAC	HVAC_ETAPPE
KWD_ALTERNATE	ALTERNIEREN	ALTERNATO	AFWISSELEN
KWD_SCHEDULE	ZEITPLAN	ORARIO	DIENSTREGELING
KWD_TREND	TENDENZ	TENDENZA	TREND
KWD_REMOTE	FERNPUNKT	PUNTO_A_DISTANZA	AFGELEGEN
KWD_WIRELESS_LINK	DRAHTLOSE_VERBINDUNG	COLLEGAMENTO_SENZA_FILI	DRAADLOZE_VERBINDING
KWD_COMM	COMM	COMM	COMM
KWD_E_MAIL	E_MAIL	E_MAIL	E_MAIL
KWD_SMS_TEXT	SMS_TEXT	SMS_TESTO	SMS_TEKST
KWD_ALSO	AUCH	ANCHE	OOK
KWD_WITH	MIT	CON	MET
KWD_ADD_T	ADDITION	AGGIUNGERE	TOEVOEGEN
KWD_SUBSTRACT_T	SUBSTRAKTION	SOTTRAZIONE	AFTREKKEN
KWD_MULTIPLY_T	MULTIPLIZIEREN	MOLTIPLICARE	VERMENIGVULDIGEN
KWD_DIVIDE_T	TEILEN	DIVIDERE	VERDELEN
KWD_START	STARTEN	PARTENZA	STARTEN
KWD_STOP	STOPPEN	ARRESTO	STOPPEN
KWD_SUB_BEGIN	SUB_BEGIN	SUB_INIZIO	SUB_BEGIN
KWD_SUB_END	SUB_ENDE	SUB_FINE	SUB_EINDE
KWD_END	ENDE	FINE	EINDE
KWD_HYSTERESIS	HYSTERESE	ISTERESI	HYSTERESIS
KWD_START_PLC	START_PLC	PARTENZA_PLC	BEGIN_PLC
KWD_DAY	TAG	GIORNO	DAG
KWD_WEEK	WOCHE	SETTIMANA	WEEK
KWD_MONTH	MONAT	MESE	MAAND
KWD_PERIOD	PERIODE	PERIODO	PERIODE
KWD_VALUE	WERT	VALORE	WAARDE
KWD_INITIALIZE	INITIALISIEREN	INIZIALIZZARE	INITIALIZE
KWD_PV	ISTWERT	VP	PV
KWD_SP	SOLLWERT	SP	SETPUNT
KWD_PB	PB	B_PROP	PB
KWD_INTEG	INTEG	INTEG	INTEG
KWD_EEPROM	EEPROM	EEPROM	EEPROM
KWD_PLC_COUNTER	PLC_ZAEHLER_SET	PLC_CONTATORE_SET	PLC_TELLER_SET
KWD_ON_CHANGE	BEI_AENDERUNG	SUL_CAMBIAMENTO	OP_VERANDERING
KWD_LAST_PERIOD	LETZTE_PERIODE	ULTIMO_PERIODO	LAATSTE_PERIODE
KWD_PARTIAL_KW	PARTIELL_KW	PARZIALE_KW	PARTIEEL_KW
KWD_PARTIAL_ACC	PARTIELL_AKKUMULIERT	PARZIALE_ACUMULATO	PARTIEEL_GEACCUMULEERDE
KWD_SAMPLE_COUNT	PROBENZAEHLUNG	CONTEGGIO_CAMPIONI	MONSTER_TELLER
KWD_ALT_OUTPUT	ALT_AUSGANG	ALT_USCITA	ALT_UITGANG
KWD_ALT_STAGES	ALT_STUFEN	ALT_STADIO	ALT_ETAPPE
KWD_ALT_LEADER	ALT_FUHRER	ALT_DIRETTORE	ALT_AANVOERDER
KWD_ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC
KWD_ALT_TMR_NEXT_STAGE	ALT_NACHSTE_STUFE	ALT_PROSSIMO_STADIO	ALT_VOLGENDE_ETAPPE
KWD_ALT_TMR_ALARM	ALT_TMR_ALARM	ALT_TMR_ALLARME	ALT_TMR_ALARM
KWD_ALT_EXT_ENABLE	ALT_EXT_FREIGABE	ALT_ABILITAZIONE_EST	ALT_EXT_SCHAKELEN
KWD_ALT_FEEDBACK	ALT_RUCKKOPPLUNG	ALT_RISPOSTA	ALT_TERGUGKOPPELING
KWD_ALT_PARALLEL	ALT_PARALLEL	ALT_PARALLELO	ALT_PARALLEL
KWD_PARTIAL_COUNT	TEILZAEHLUNG	CONTEGGIO_PARZIALE	GEDEELTELIJKE_TELLING
KWD_COOLING	KUEHLUNG	REFFREDDAMENTO	KOELING
KWD_HEATING	HEIZUNG	RISCALDAMENTO	VERWARMING
KWD_TMR_MINIMUM_ON	TMR_MIN_EIN	TMR_MIN_ACCENDERE	TMR_MIN_IN

Keywords table 6 of 6

Keyword	DE	IT	NL
KWD_TMR_MINIMUM_OFF	TMR_MIN_AUS	TMR_MIN_SPEGNERE	TMR_MIN_UIT
KWD_TIMER_INTERSTAGE	TMR_ZWISCHENSTUFE	TMR_INTERSTADIO	TMR_INTERSTAGE
KWD_STAGE_RUN	STUFE_LAUF	STADIO_FUNZIONARE	ETAPPE_LOPEN
KWD_INTERVAL_MINUTES	INTERVALL_MINUTEN	INTERVALLO_MINUTI	INTERVAL_MINUTEN
KWD_READ_COIL	READ_COIL	READ_COIL	READ_COIL
KWD_READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS
KWD_READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER
KWD_READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER
KWD_ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE
KWD_BINARY_VALUE	BINARY_VALUE	BINARY_VALUE	BINARY_VALUE
KWD_NX_SLAVE	NX_SLAVE	NX_SCHIAVO	NX_SLAAF
KWD_WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C
KWD_WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F
KWD_WLS_REL_HUM	WLS_REL_FEUCHTIGKEIT	WLS_UMIDITA_REL	WLS_REL_VOCHTIGHEID
KWD_WLS_MODE	WLS_MODUS	WLS_MODALITA	WLS_MODE
KWD_WLS_FAN_SPEED	WLS_LUFTER_GESCHWINDIGKEIT	WLS_VELOCITA_VENT	WLS_VENT_SNELHEID
KWD_WLS_KEYBOARD	WLS_TESTATUR	WLS_TESTIERA	WLS_TOETSENBORD
KWD_WLS_SP_TEMP	WLS_SP_TEMP	WLS_SP_TEMP	WLS_SP_TEMP
KWD_WLS_SP_HUM	WLS_SP_FEUCHTIGKEIT	WLS_SP_UMIDITA	WLS_SP_VOCHTIGHEID
KWD_WLS_SP_T1	WLS_SP_T1	WLS_SP_T1	WLS_SP_T1
KWD_WLS_SP_PB	WLS_SP_PB	WLS_SP_PB	WLS_SP_PB
KWD_WLS_SP_UNOCC	WLS_SP_UNBESETZ	WLS_SP_DISOCCUPATO	WLS_SP_ONBEZET
KWD_WLS_BATTERY_VOLTAGE	WLS_BATTERIESPANNUNG	WLS_VOLTAGIO_BATTERIA	WLS_BATTERIJ_VOLTAGE
KWD_WLS_AUX_INP	WLS_AUX_EIN	WLS_AUX_ING	WLS_AUX_ING
KWD_WLS_LINK_TMR	WLS_LINK_TMR	WLS_LINK_TMR	WLS_LINK_TMR
KWD_WLS_SEC_LINK_LOST	WLS_SEK_KEIN_LINK	WLS_SEC_SENSA_COLLEGAMENTO	WLS_SEC_ZONDER_KOPPELING
KWD_CONTROLLER_NAME	REGLERNAME	NOME_DISPOSITIVO	CONTROLLER_NAAM
KWD_WLS_GROUP	WLS_GRUPPE	WLS_GRUPPO	WLS_GROEP
KWD_WLS_ADDRESS	WLS_ADDRESSE	WLS_INDIRIZZO	WLS_ADRES
KWD_DEVICE_ADDRESS	REGLERADRESSE	INDIRIZZO_DISPOSITIVO	CONTROLLER_ADRES
KWD_PARITY_NONE	PARITAT_KEINE	PARITA_NESSUNA	PARITEIT_NONE
KWD_PARITY_ODD	PARITAT_UNGERADE	PARITA_DISPARI	PARITEIT_ONEVEN
KWD_PARITY_EVEN	PARITAT_SOGAR	PARITA_PARI	PARITEIT_EVEN
KWD_STOP_BIT_0	STOP_BIT_0	STOP_BIT_0	STOP_BIT_0
KWD_STOP_BIT_1	STOP_BIT_1	STOP_BIT_1	STOP_BIT_1
KWD_PROTOCOL_ASCII_TERMINAL	PROTOKOLL_ASCII_TERMINAL	PROTOCOLLO_ASCII_TERMINAL	PROTOCOL_ASCII_TERMINAL
KWD_PROTOCOL_OPTO_22_SLAVE	PROTOKOLL_OPTO_22_SLAVE	PROTOCOLLO_OPTO_22_SCHIAVO	PROTOCOL_OPTO_22_SLAAF
KWD_PROTOCOL_N2_OPEN_SLAVE	PROTOKOLL_N2_OPEN_SLAVE	PROTOCOLLO_N2_OPEN_SCHIAVO	PROTOCOL_N2_OPEN_SLAAF
KWD_PROTOCOL_MODBUS_SLAVE	PROTOKOLL_MODBUS_SLAVE	PROTOCOLLO_MODBUS_SCHIAVO	PROTOCOL_MODBUS_SLAAF
KWD_PROTOCOL_BANCET_MSTP	PROTOKOLL_BANCET_MSTP	PROTOCOLLO_BANCET_MSTP	PROTOCOL_BANCET_MSTP
KWD_PROTOCOL_O22_MASTER	PROTOKOLL_OPTO22_MEISTER	PROTOCOLLO_OPTO22_MASTER	PROTOCOL_OPTO22_MEESTER
KWD_PROTOCOL_MODBUS_MASTER	PROTOKOLL_MODBUS_MEISTER	PROTOCOLLO_MODBUS_MASTER	PROTOCOL_MODBUS_MEESTER
KWD_BAUD_RATE	BAUD	BAUD	BAUD
KWD_AI_CONFIG	AE_KONFIGURATION	IA_CONFIGURAZIONE	IA_CONFIGURATIE
KWD_AI_CALIBRATION	AE_KALIBRIERUNG	IA_CALIBRAZIONE	IA_KALIBRIERING
KWD_SCRIPT_LAST	SCRIPT_END	SCRIPT_END	SCRIPT_END

Detailed keyword usage

The following section gives detailed description in the use of each of the keywords, as well as syntax rules and example code. Only the English language of the script is given in detail. The keywords can be substituted by the user selected language if the first keyword of the file is the language selection keyword.

If the script files are in a language other than English, **the first line** of either **script_def.txt** if it exists or **script_1.txt** must contain the **language directive keyword** or an error will be generated because the keyword´s language will be undefined. These language directives keywords are as listed in the following table:

	GB	FR	ES	DE	IT	NL
Language selector	**ENGLISH**	**FRENCH**	**SPANISH**	**GERMAN**	**ITALIAN**	**DUTCH**
Alternate		**FRANÇAIS**	**ESPAÑOL**	**DEUTSCH**	**ITALIANO**	**NEDERLANDS**

These keywords besides being in the very first line of the script or definition file, must be alone in the line, no other additional keywords or comments should be on it.
The following is an example to set the script file language to French:

```
FRENCH
// Must be first word of first line of first file (DEF or SCRIPT)
```

Also the French keyword version could be used, it can be in capitals or non-capital letters.

```
Français
// Must be the first word in the first line of the first file (DEF or SCRIPT)
```

Note that the English version of the database names and keywords will always be available, so when a user writing for example the script in French, he or she can freely mix both languages and use some English and some French keywords as well as database objects names indiscriminately.

On the preceding pages, the keyword list for the different supported languages is shown. Note that accents and other nonstandard characters have been replaced with standard non-accentuated characters to make keyboard input and language translation easy to work with.

All words are case insensitive so the user can write in capital or non-capital or mixed characters. For some keywords and database objects there exist shorthand versions to make script programming easier.

Keyword: **DEFINE**

Shortcut: **DEF**

GB	FR	ES	DE	IT	NL
DEFINE	DEFINIR	DEFINE	DEFINIEREN	DEFINIRE	BEPALEN
DEF	DEF	DEF	DEF	DEF	DEF

The 'DEFINE' or its shortcut 'DEF' keyword defines equates to be able to call data base objects by user defined names:

Syntax: **DEFINE** [DB OBJECT] [optional '=' operator keyword to make reading easier]
[NEW LABEL THAT EQUATES TO DB OBJECT BY NAME] 32 bit characters or less.
Underscores are allowed in the label, it is case insensitive

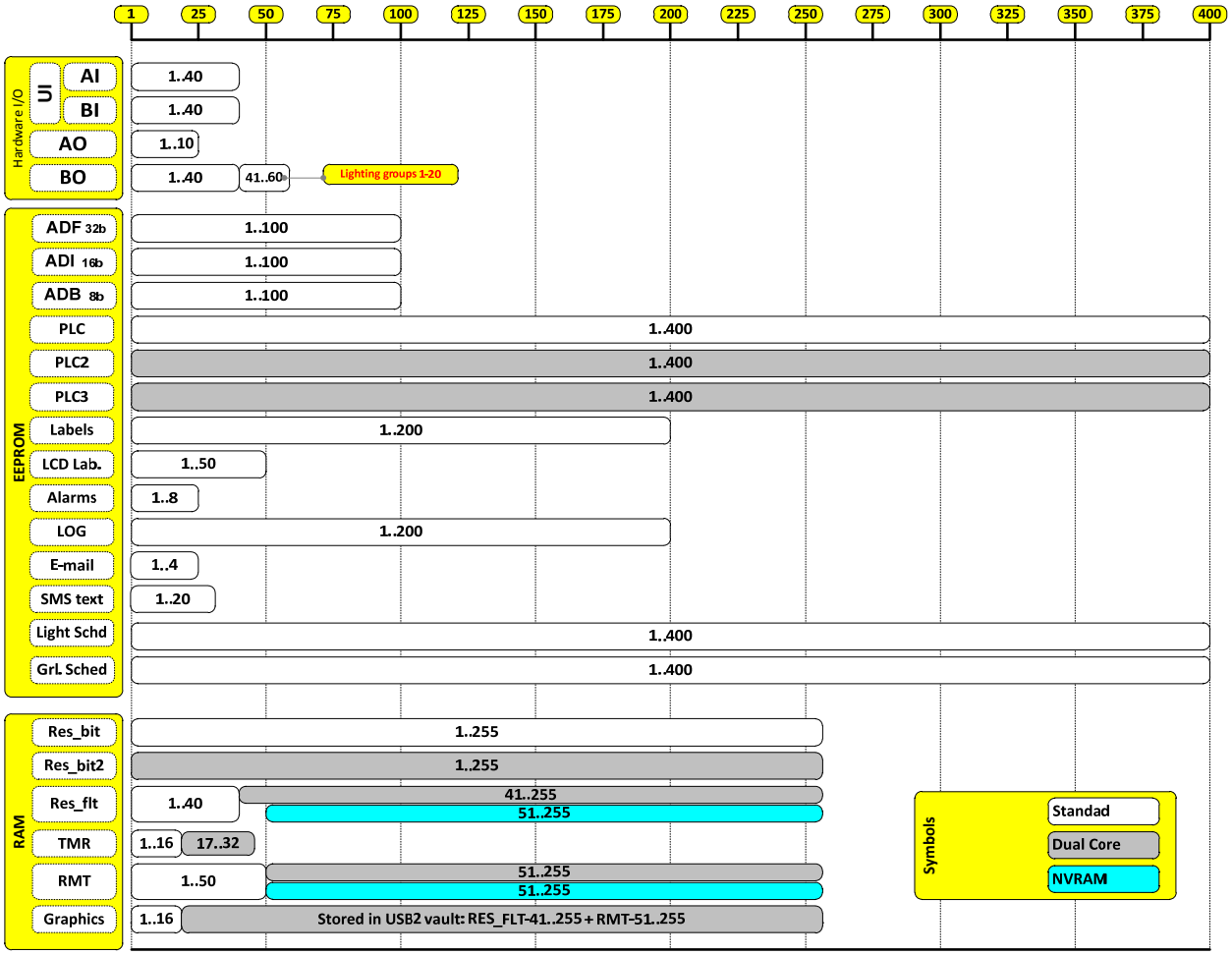
```
DEFINE      BI_1   ENTRY_DOOR   // Standard definition
DEFINE      BI_1 = ENTRY_DOOR   // Optional '=' added for clarity
DEF         BI_1   ENTRY_DOOR   // Short notation
DEF         BI_1 = ENTRY_DOOR   // Short notation with optional '=' added for clarity
```

OpenBAS NX database OBJECTS will be the operands or results in the rest of the equations and can be:

Description	Defined as	Notes
Analog inputs	AI_1 to 40	(shared with binary inputs thru universal inputs)
Binary inputs	BI_1 to 40	(shared with analog inputs thru universal inputs)
Analog outputs	AO_1 to 10	
Binary outputs	BO_1 to 60	(41 to 60 are lighting groups)
EEPROM setpoint 32 bits	ADF_1 to 100	
EEPROM setpoint 16 bits	ADI_1 to 100	
EEPROM setpoint 8 bits	ADB_1 to 100	
Lighting groups	LG_1 to 20	(remapped to binary outputs 41 to 60)
Result bits registers	RES_BIT_1 to 255	Used to store results of binary (digital) operations
Result float registers	RES_FLT_1 to 40	Used to store results of any other math operations (255 with dual core)
System timers	TMR_1 to 16	(32 with dual core)
Remote points	RMT_1 to 50	Wired or wireless operations (255 with dual core)
Integer 8 bit constants	KBYT_0 to 254	Used as constants for values between 0 and 254
Remote points result reg	RMT_RES_41 to 255	Used as additional remote points or to store results of math operations (with dual core)

A table with a visual representation of the full OpenBAS NX database can be found on the next page.

Table with OpenBAS NX database objects:



As seen on the preceding examples, comments can be added anywhere on any line using the double forward slash characters '//' similar to Visual studio languages such as C, C++, C# etc.
Anything beyond the comment characters will be treated as comments and not processed by the compiler.

Adding comments to your program can help to better understand the logic or sequence of operation, there is no limit on how many lines with instructions or comments a script file can have.

The line length is however restricted to 250 characters, if longer lines are needed or if breaking them to improve readability the backward slash character '\' can be used. This way the preprocessor will join lines ending with the '\' character before parsing them and extracting the individual tokens. On the next page multiline syntax is shown.

Sample code in a single line of script

```
    IF AI_1 > 17.5 AND AO_1 = 25  OR TEMPERATURE <= ADF_1 + 3 THEN BO_1 = ON ALSO BO_2 = OFF    .
```

This is the same code but now using multiple lines using the backward slash character to break the lines to improve readability.

```
IF    AI_1 > 17.5 AND          \
      AO_1 = 25   OR           \
      TEMPERATURE <= ADF_1 + 3 \
  THEN BO_1 = ON               \
  ALSO BO_2 = OFF
```

Also be aware that as in any written language, spaces or tabs are needed to separate the tokens that make the words, keywords, operators, operands, database objects etc. and make the program readable.

Even while the pre-processor has a token breaker feature so a line such as:

`AO_1=ADF_1+3` will be converted to: `AO_1 = ADF_1 + 3`

It is a good practice to write script programs the same way a standard sentence is constructed by placing spaces between the words that make up the whole sentence.

Also indentation using tabs or spaces is a good programming practice, to make programs more readable to humans, aligning data vertically as nesting goes into the program logic, helps to visually better understand a long program.

Keyword: INITIALIZE

GB	FR	ES	DE	IT	NL
INITIALIZE	INITIALISER	INICIALIZA	INITIALISIEREN	INIZIALIZZARE	INITIALIZE

Once an EEPROM variable name has been created to alias to a database object, and even without being named it can be initialized to a given value, this way when the script is compiled, it´s value will be generated and initialized one time when the generated program is downloaded to the controller.

The syntax to initialize an EEPROM database object is a follows:

Syntax: EEPROM_REGISTER = Value INITIALIZE

Example:

```
ADF_1 = 22.5 INITIALIZE  // Any valid 32 bit float value
ADI_1 = 1000 INITIALIZE  // Any value from 0..65535
ADI_B = 7    INITIALIZE  // Any value from 0..255
```

If labels are previously created using the DEFINE keyword, these names can be used instead, for example:

Example:

```
DEFINE ADF_1 TEMPERATURE_SET_POINT
TEMPERATURE_SET_POINT = 22.5 INITIALIZE   // Initialize by name
```

Important note: If the INITIALIZE keyword is not used and only a simple assignment is used, the variable will be written all the time. For example in the following expression; `ADF_1 = 22.5` will assign the value of 22.5 to the ADF_1 all the time the program is running, so if the user manually adjusts the set point, it will be automatically be re-written with the 22.5 value while the program is being executed.
But if the INITIALIZE keyword is used as in: `ADF_1 = 22.5 INITIALIZE` then the value will only be assigned when the program is loaded, and then any changes done by the user will not be affected.

There are times under program control that EEPROM values need to be modified. For example in the following example the set point is only adjusted if there is an event, and this is completely acceptable:

```
// Will set the value od ADF under a certain condition only
Example:    IF BI_1 = CLOSED THEN ADF_1 = 25.5
```

Please note an EEPROM register has a write life time of 1,000,000 cycles and repeatedly changing it under program control will damage the memory cell. In the example above because no damage happens because the OS takes care to only write EEPROM values when the assigned value changes.

Keyword: LT_GROUP

GB	FR	ES	DE	IT	NL
LT_GROUP	GROUPE_D_ECLAIRAGE	GRUPO_ILUM	BELEUCHTUNGSGRUPPE	GRUPO_ILUM	LICHTGROEP

The 'LT_GROUP' keyword allows to group up to eight binary outputs into a logical group. Even while the name implies it is used mainly for lighting, it is just an output grouping function, so it can be to turn on simultaneously any kind of load such as motors, valves, lights, etc. The lighting group can be created using the following syntax

Syntax: [RESULT REGISTER] = **LT_GROUP** [OPERAND_1] .. [optional up to 8 operands]

```
Example:    RES_BIT_1 = LT_GROUP BO_1 BO_2 BO_3 BO_4
            // Up to 8 binary outputs can be grouped with into
            // a single control bit, up to 20 groups can be created
```

If labels are previously created using the DEFINE keyword for the binary outputs, these names can be used instead when creating the lighting group, for example:

```
// Use user defined labels instead of database ID's to make line more
// readable
Example:    DEFINE RES_BIT_1  = SOURCE_OF_CONTROL
            DEFINE BO_1       = NORTH_CORRIDOR
            DEFINE BO_2       = SOUTH_CORRIDOR
            DEFINE BO_3       = MAIN_ENTRANCE
            [LIGHTING_GR_1]
            SOURCE_OF_CONTROL = LT_GROUP  NORTH_CORRIDOR \
                                          SOUTH_CORRIDOR \
                                          MAIN_ENTRANCE
            END
```

In the above example we can see how the names for the variables were defined in advance, so the lighting group can be created with these names instead. Also to note is that indenting is used and the line where the lighting group is created was broken down to a multiline by using the back slash character.

One thing to note is that a PLC label called [LIGHTING_GR_1] was added before the lighting group lines. This kind of labels is needed when jumps and calls are used in the program to jump to a label or call a subroutine by name. A PLC label is created with the limitation that only the first nine characters will be stored as a PLC label instruction. Even though the label can be up to 32 bytes in size, the same as any custom defined name or tag ID label.

Also note the use of the **END** keyword, any code after the END keyword will not be added to the program. If the END label is not used, the compiler will automatically create one virtual END instruction at the very end of the last used program instruction.

Only subroutines that are explained later in this user guide are allowed to be placed above the last END instruction where the program ends, because they are called from within the main program loop and so if a subroutine is found after an END keyword, it will be included in the program.

Keyword: END

Keyword: JUMP

Keyword: CALL

Keyword: SUB_BEGIN

Keyword: SUB_END

Keyword: [X] (Custom defined ID Labels, 1 to 32 characters in size)

GB	FR	ES	DE	IT	NL
END	FIN	FIN	ENDE	FINE	EINDE
JUMP	SAUT	BRINCA	SPRING	SALTARE	SPRING
CALL	APPEL	LLAMA	ANRUF	CHIAMATA	OPROEP
SUB_BEGIN	SUB_DEMARRER	SUB_INICIO	SUB_BEGIN	SUB_INIZIO	SUB_BEGIN
SUB_END	SUB_FIN	SUB_FIN	SUB_ENDE	SUB_FINE	SUB_EINDE

As shown in the previous example on the last page, the **END** keyword is used to mark the end of the program. Any code after the END keyword will not be added to the program. If the END label is not used, the compiler will automatically create one virtual END instruction at the very end of the last used program instruction.

By using the **JUMP** and **CALL** instructions the END instruction can be skipped under program control to do logic based on the state of the database (I/O's data, schedules etc.).

This is very powerful to allow the program to branch based on the status of the different variables that make up the controller's database.

The 'JUMP' keyword allows for branching of program flow based in process status, it takes the following format:

Syntax: **JUMP** [LABEL] (**Custom defined ID Labels, 1-32 characters in size)**

```
Example:    JUMP SKIP_1              // Unconditional jump
            .. Some instructions jumped unconditionally
            [SKIP_1]                 // Placeholder label for jump instruction
            .. Program jumps to the SKIP_1 label
               and continues processing from this point
```

Another use of the JUMP instruction in conjunction with the **IF** keyword, is to branch the program when a condition is detected.

```
Example:    IF BI_1 IS OPEN THEN JUMP SKIP_2    // Conditional jump
            .. Some instructions jumped if BI_1 is open
            [SKIP_2]                // Placeholder label for jump instruction
            .. Program jumps to the SKIP_2 label depending on BI_1 status
               and continues processing from this point
```

In the example program above, the JUMP skips some instructions if the result of the comparison is TRUE, otherwise the instructions marked in red would get executed. More information on the IF keyword will be given later in this chapter.

The 'CALL' keyword allows for branching of program flow based in process status. It differentiates from the jump, that after the "called" subroutine ends, it returns control to the program on the line following the CALL instruction, it takes the following format:

```
Syntax:     CALL [LABEL] (Custom defined ID Labels, 1-32 characters in size)

Example:    CALL SUBROUTINE_PROGRAM       // Unconditional subroutine call
            .. After the subroutine ends, // from main program
               the program will continue executing
               the lines following the CALL keyword
            END   // Program will end here

            SUB_BEGIN [SUBROUTINE_PROGRAM] // Labels for documentation or
            .. Subroutine program          // calls or jumps can be easily
                                           // created using square brackets
            SUB_END                        // Program returns control from
                                           // a subroutine wehen the END_SUB
                                           // instruction is found
```

Usually subroutines are located after the END of the main program, so it even makes sense to add them in separate script files (other than script_1.txt) to keep the main program clean and readable.

Remember that at compile time the pre-processor merges all the script files along with the definition file to be able to have full visibility of the program.

One thing to be aware of is that when the SUB_BEGIN keyword is found, the compiler automatically adds an END instruction before the subroutine starts for if the user forgets to do so and avoid unexpected results when the program executes.

Similar to the conditional JUMP, the CALL instruction can be used in conjunction with the **IF** keyword, to branch the program when a condition is detected.

```
Example:    IF BI_1 IS OPEN THEN CALL SUBROUTINE_PRG_2
                                      // Conditional subroutine call
            .. after the subroutine ends, // from main program
               the program will continue executing
               the lines following the CALL keyword
            END   // Program will end here

            SUB_BEGIN [SUBROUTINE_ PRG_2]  // Labels for documentation or
            .. Subroutine program          // calls or jumps can be easily
                                           // created using square brackets
            SUB_END                        // Program returns control from
                                           // a subroutine wehen the END_SUB
                                           // instruction is found
```

In the example program above, the CALL calls the subroutine if the result of the comparison is TRUE, otherwise the subroutine will not be called at all. More information on the IF keyword will be given later in this chapter.

NOTE: It is important to have in mind that subroutines are not re-entrant, this means a subroutine can´t call another subroutine, so basically there exist only two levels of program running at any time:

- The main program loop.
- Subroutine levels.

So every time an END instruction is found is interpreted depending on the context, if the program is in the main loop the program will end, and a new iteration will begin.

If instead the END is found while inside a subroutine, the subroutine will end and yield control to the instruction following the subroutine CALL in the main loop.

Also keep in mind that in processors with dual core that can have three PLC's running simultaneously, each PLC runs isolated and separated from the other PLC's, so they can´t call subroutines located in another PLC´s.

The only "glue" that ties the PLC's together is the OpenBAS database collection of objects, where information can be passed between them such as values, semaphores, etc

Keyword: IF

Keyword: ELSE

Keyword: THEN

Keyword: ALSO

GB	FR	ES	DE	IT	NL
IF	SI	SI	OB	NEL_CASO_QUE	ALS
ELSE	AUTRE	CASO_CONTRARIO	SONST	ALTRIMENTI	ANDERS
THEN	PUIS	ENTONCES	DANN	ALORA	DAN
ALSO	AUSSI	TAMBIEN	AUCH	ANCHE	OOK

The '**IF**' keyword uses comparison operators to compare two operands, and sets a result if the expression is TRUE after the THEN keyword, the optional ELSE keyword gets executed instead if the comparison result is FALSE.

The comparison can be simple or nested comparisons can be joined by logical AND or OR keywords, each comparison is evaluated independently and ANDED or ORED with the next as they appear on the line.

Each of the left and right operands can be simple or complex, complex operands can use the:

- MIN/MAX/AVG keyword
- or have simple math such as: 'MAX a b c > MIN d e f ' , ' a + b > c + d ' , ' a > b '

After the THEN or ELSE a single assignment can be performed such as a = b or multiple assignments can take place using the ALSO keyword.

There can be two types of IF expressions, UNARY or BINARY:

Syntax: IF [OPERAND_1] [**COMPARISON operators: <, <=, >, >=, ==, =, !=**] [OPERAND_2] \
[AND/OR for nested expressions] [... more comparison expressions] \
THEN [RESULT] [= or IS] [VALUE] ALSO [... more assignments] // Unary IF expression

Syntax: IF [OPERAND_1] [**COMPARISON operators: <, <=, >, >=, ==, =, !=]** [OPERAND_2] \
[AND/OR for nested expressions] [... more comparison expressions] \
THEN [RESULT] [= or IS] [VALUE] ALSO [... more assignments] \
ELSE [RESULT] [= or IS] [VALUE] ALSO [... more assignments] // Binary IF expression

The difference between the UNARY and the BINARY expressions is that:

- The unary expression, the assignments after the THEN keyword will be executed only if the result of the single or nested comparison is TRUE. If the result is FALSE no assignment will be executed.

- Whereas in the binary expression, the single or multiple assignments after the THEN keyword will be executed only if the result of the comparison(s) is (are) TRUE. And the assignments following the ELSE will be executed only if the result of the comparison(s) is (are) FALSE.

The following examples show some simple and advanced use if this powerful keyword.

```
Example:     IF AI_1 > 22.5 THEN BO_1 = ON
```

Using the first syntax the binary output 1 is set to ON (= 1) if the value of the analog input 1 is greater than 22.5. Note that the output will not revert if it is less or equal than 22.5, as there is no matching ELSE keyword to do this.

```
Example:     IF AI_1 > 22.5 THEN BO_1 = ON ELSE BO_1 = OFF
```

Using the second example, the binary output 1 is set to ON (=1) if the value of the analog input 1 is greater than 22.5, otherwise the output will be reset to OFF (=0) by means of the ELSE keyword.

```
Example:     [IF_COMPLEX_OPERANDS]   // Simple Math can be applied to operands
             IF AI_1 * 2 > AI_2 + 3                                  \
                THEN AO_1 = 10 ALSO AO_2 = 20 ALSO BO_1 = ON         \
                ELSE AO_1 = 30 ALSO AO_2 = 40 ALSO BO_1 = OFF

// Also MIN, MAX and AVG keywords can be used for left or right operands
Example:     IF MAX AI_1 AI_2 AI_3 AI_4 > MIN AI_4 AI_5 AI_6         \
                THEN AO_1 = 10 ALSO AO_2 = 20                        \
                ELSE AO_1 = 30 ALSO AO_2 = 40

Example:     [IF_AND_OR_NESTED_EXPRESIONS]
             IF BI_1 IS OPEN AND BI_2 = CLOSED OR BI_3 = CLOSED OR \
                BI_4 = CLOSED AND BI_4 IS NOT OPEN                   \
                THEN BO_1 = ON  ALSO BO_1 = ON                       \
                ELSE BO_1 = OFF ALSO BO_1 = OFF

Example:     IF BI_1 = OPEN AND BI_2 = CLOSED                        \
                THEN BO_1 = ON                              \
                ELSE BO_1 = OFF
```

```
Example:    [SIMPLE _IF]
Example:    IF BI_1 = BI_21  THEN BO_1 = ON

Example:    [TIMER_LOAD_RUN_COMMAND]
Example:    IF BI_1 = OPEN  AND BI_2 = CLOSED          \
                THEN LOAD TMR_1                        \
                ELSE RUN  TMR_1  ALSO MOTOR = OFF

// using IS and IS NOT instead of the
// '=' or '==' or '!=' (comparison operators)
Example:    IF BI_1 IS OPEN AND BI_2 IS NOT OPEN       \
                THEN BO_1 = ON                         \
                ELSE BO_1 = OFF

Example:       IF BI_1 IS = OPEN AND AO_3 IS >= 50     \
                THEN BO_1 = ON                         \
                ELSE BO_1 = OFF
```

Note that AND / OR keywords can be used to create nested expressions, they are evaluated as they are found and there is no precedence. If the standard follow thru precedence is not enough, intermediate variables can be first created with the desired Boolean or math operations and the used in the IF expression such as in the following example:

```
Example:    RES_FLT_1 = MIN AI_1 AI_2 AI_3      // Get the maximum value
            RES_FLT_1 *= 3                      // Multiply by 3
            RES_FLT_2 = AVG AI_4 AI_5 AI_6      // Get the average value
            RES_FLT_2 += RMT_7                  // Add value of remote 7
            // Now do the IF expression and evaluate
            IF RES_FLT_1 != RES_FLT_2 AND RES_FLT_1 + 5 IS >= RES_FLT_1 \
                THEN BO_1 = ON                                          \
                ELSE BO_1 = OFF
```

On the next pages are tables with operators, and keywords that can be used with the IF instruction:

Table with comparison operators:

GB	FR	ES	DE	IT	NL
<	<	<	<	<	<
<=	<=	<=	<=	<=	<=
>	>	>	>	>	>
>=	>=	>=	>=	>=	>=
=	=	=	=	=	=
!=	!=	!=	!=	!=	!=
==	==	==	==	==	==
IS	ETRE	ES	IST	SEI	IS
NOT	NON	NO	NICHT	NO	NIET

Note that the IS keyword only helps to make reading easier and is interpreted as an '=' ASSINMENT or '==' EQUAL operator.

Take caution however when using it together with the NOT keyword as it inverts the logic of the operators, for example in the following examples, both comparison expressions give the same the same results:

```
Example:    IF BI_1 = OPEN            THEN BO_1 = ON
            IF BI_1 IS OPEN           THEN BO_1 = ON
            IF BI_1 IS = OPEN         THEN BO_1 = ON
            IF BI_1 != CLOSED         THEN BO_1 = ON
            IF BI_1 IS NOT CLOSED     THEN BO_1 = ON
            IF BI_1 IS NOT = CLOSED   THEN BO_1 = ON
```

Also the following keywords are available and predefined for the user to make the scripts easier to read and understand:

Table with predefined labels and constants:

GB	FR	ES	DE	IT	NL
ON	ALLUME	ENCENDIDO	EINSCHALT	ACCENDE	AANDOEN
OFF	ETEINDRE	APAGADO	AUSSCHALT	SPENGE	UITDOEN
OPEN	OUVERT	ABIERTO	OFFEN	APERTO	OPEN
CLOSED	FERME	CERRADO	GESCHLOSSEN	CHIUSO	DICHT
TRUE	VRAI	VERDADERO	WAHR	VERO	WAAR
FALSE	FAUX	FALSO	VALS	FALSO	VALS
RUN	COURIR	CORRE	LAUF	CORRERE	LOPEN
LOAD	CHARGER	CARGA	BELAST	CARICARE	LADEN

Table with math operators:

GB	FR	ES	DE	IT	NL
+	+	+	+	+	+
-	-	-	-	-	-
*	*	*	*	*	*
/	/	/	/	/	/
MIN	MIN	MIN	MIN	MIN	MIN
MAX	MAX	MAX	MAX	MASSIMO	MAX
AVG	MOYENNE	PROMEDIO	DURCHSCHNITT	MEDIA	GEMIDDELDE
+=	+=	+=	+=	+=	+=
-=	-=	-=	-=	-=	-=
/=	/=	/=	/=	/=	/=
*=	*=	*=	*=	*=	*=

Note that the current implementation of the compiler only has simple math on the right side of the '=' assignment operation, as there is no precedence or parenthesis enforced precedence. That is a task that might be later implemented.

Table with binary controlling operators:

GB	FR	ES	DE	IT	NL
RUN	COURIR	CORRE	LAUF	CORRERE	LOPEN
LOAD	CHARGER	CARGA	BELAST	CARICARE	LADEN
SET	SET	FIJAR_A_1	SETZEN	SET	SET
RESET	RESET	FIJAR_A_0	ZURUECKSETZEN	RESET	RESET
START	DEMARRER	ARRANQUE	STARTEN	PARTENZA	STARTEN
STOP	ARRETEZ	PARO	STOPPEN	ARRESTO	STOPPEN

Beside the LOAD/RUN pair used for timers, the SET/RESET and START/STOP pair can be used in the ELSE/THEN sections of the IF keyword to turn binary registers ON/OFF. See the following examples below:

```
Example:    IF BI_1 = OPEN     THEN LOAD  TMR_1  ELSE RUN   TMR_1
            IF BI_1 IS OPEN    THEN SET   BO_1   ELSE RESET BO_1
            IF BI_1 IS OPEN    THEN START BO_1   ELSE STOP  BO_1
```

Note there is no '=' assignment operator between the mentioned keywords and the assigned register, as in this case the destination operand (the one receiving the value) is the right operand, whereas in the normal assignment the destination operand is the left operand.

There is a different use for the SET/RESET and START/STOP keywords that will be explained later in this chapter.

This finishes the IF keyword section. The next keywords described will be some math and Boolean instructions that might become handy when designing programs with large amount of variables and extensive operations are needed.

Keywords: symbolic math operators + - * /

GB	FR	ES	DE	IT	NL
+	+	+	+	+	+
-	-	-	-	-	-
*	*	*	*	*	*
/	/	/	/	/	/

These MATH OPERATOR keywords allow doing simple math using the script compiler such as:

```
Syntax:      [RESULT REGISTER] = [OPERAND_1] MATH_OPERATOR [OPERAND_2]
                        Optional up to 20 operands
                        (10 for multiply and divide)

Example:     RES_FLT_1    = BI_1       + -15.3
             RES_FLT_2    = AI_1       - AI_2
             AO_1         = RES_FLT_1  * RMT_1
             RMT_1        = 22.5       / RMT_1
```

Complex math or use of parentheses is not supported so complex math sequences should be broken into simple two operand instructions.

However using compound instructions, math is not limited to the simple math operations shown above. The next pages show some usage of advanced math using compound operators.

Also other math operators are given in the two next tables:

Keywords: text math operators ADD, SUBSTRACT, MULTIPLY, DIVIDE

GB	FR	ES	DE	IT	NL
ADD	ADDITION	SUMA	ADDITION	AGGIUNGERE	TOEVOEGEN
SUBSTRACT	SOUSTRACTION	RESTA	SUBSTRAKTION	SOTTRAZIONE	AFTREKKEN
MULTIPLY	MULTIPLICATION	MULTIPLICACION	MULTIPLIZIEREN	MOLTIPLICARE	VERMENIGVULDIGEN
DIVIDE	DIVIDE	DIVISION	TEILEN	DIVIDERE	VERDELEN

Keywords: extra math operators MIN, MAX, AVG

GB	FR	ES	DE	IT	NL
MIN	MIN	MIN	MIN	MIN	MIN
MAX	MAX	MAX	MAX	MASSIMO	MAX
AVG	MOYENNE	PROMEDIO	DURCHSCHNITT	MEDIA	GEMIDDELDE

```
// Symbolic math operators and their text counterparts can take up to
// 20 variables or constants to be calculated for addition and subtraction
Example:    RES_FLT_1 = + AI_1 AI_2 AI_3 AI_4 AI_5 22.5 17.5 AO_10

            RES_FLT_2 = - AI_1 AI_2 AI_3 AI_4 AI_5 22.5 17.5 AO_10

            RES_FLT_3 = ADD AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7 AI_8

            RES_FLT_4 = SUBSTRACT AI_1 AI_2 AI_3 AI_4 AI_5 22.5

// Multiply and divide keywords can take only up to 10 keywords variables and
// constants to be be calculated
Example:    RES_FLT_1 = * AI_1 AI_2 AI_3 AI_4 AI_5 AI_6

            RES_FLT_2 = / AI_1 AI_2 AI_3 AI_4 AI_5 AI_6

            RES_FLT_3 = MULTIPLY AI_1 AI_2 AI_3 AI_4 AI_5 AI_6

            RES_FLT_4 = DIVIDE AI_1 AI_2 AI_3 AI_4 AI_5 AI_6

// When using simple assignment operator '=' For minimum, maximum or average
// keywords of up to 20 variables and constants can be calculated on a single
// line
Example:    RES_FLT_1 = MIN AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7

            RES_FLT_2 = MAX AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7 AI_8

            RES_FLT_3 = AVG AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7 AI_8
```

Keywords: compound math operators += -= *= /=

GB	FR	ES	DE	IT	NL
+=	+=	+=	+=	+=	+=
-=	-=	-=	-=	-=	-=
/=	/=	/=	/=	/=	/=
*=	*=	*=	*=	*=	*=

The compound assignment tokens '+=', '-=', '*=', '/=' perform math instructions by; adding, subtracting, multiplying or dividing the left operand with the right operand, and then storing back the result of the operation in the left operand.

When using the compound operands the MIN, MAX and AVG keywords can be used together to obtain the minimum, maximum or average of the operands, and then doing the compound math as can be seen on the examples on the next page.

Simple compound operands can take one or two operands, and optionally a MIN, MAX or AVG keyword with up to each operand.

```
Syntax:     [RESULT REGISTER] += -= *= /= [OPERAND_1]


Example:    RES_FLT_1   += 1500.3   // Samples of compound math
            AO_1        -= AI_2     // with one single operand.
            RES_FLT_2   *= RMT_1
            RMT_1       /= -1.1


            // Now some samples of compound math with two operands
            // and an additional math operator to do some basic math
            // before doing the compound assignment.
            RES_FLT_1   += 1500.3 * ADI_1
            RES_FLT_2   += RES_FLT_2 - AO_1
            RES_FLT_3   -= 1500.3 / ADI_1
            RES_FLT_4   *= AO_1 + 25
```

Up to four operators can be used after the MIN, MAX, AVG keywords to be processed before their result is applied to compound instructions.

```
// Compound assignments using the MIN, MAX, AVG can take 1 to 4 parameters
Syntax:     [RESULT REGISTER] += -= *= /= MIN [OP_1] [OP_2] [OP_3] [OP_4]

Syntax:     [RESULT REGISTER] += -= *= /= MAX [OP_1] [OP_2] [OP_3] [OP_4]

Syntax:     [RESULT REGISTER] += -= *= /= AVG [OP_1] [OP_2] [OP_3] [OP_4]

Example:    RES_FLT_1 += MIN AI_1 AI_2

            RES_FLT_2 -= MAX AI_1 AI_2 AI_3 AI_4

            RES_FLT_3 *= AVG AI_1 AI_2 AI_3

            RES_FLT_4 /= AVG AI_1 AI_2 AI_3 AI_4
```

Simple assignment operand can take up to twenty operands when combined with the symbolic or text math operator keywords as well as with the MIN, MAX and AVG keywords.

In this case, after the '=' simple assignment operator a space must be left and then the desired math operator and then up to 20 operands (10 in the case of multiplication and division) each separated with a space or a tab.

```
// When using simple assignment operator '=' for minimum, maximum or average
// keywords of up to 20 variables and constants can be calculated on a single
// line
Example:          RES_FLT_1 = MIN AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7

                  RES_FLT_2 = MAX AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7 AI_8

                  RES_FLT_3 = AVG AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7 AI_8


// Also symbolic math operators and their text counterparts can take up to
// 20 variables or constants to be calculated for addition and subtraction
Example:          RES_FLT_1 = + AI_1 AI_2 AI_3 AI_4 AI_5 22.5 17.5 AO_10

                  RES_FLT_2 = - AI_1 AI_2 AI_3 AI_4 AI_5 22.5 17.5 AO_10

                  RES_FLT_3 = ADD AI_1 AI_2 AI_3 AI_4 AI_5 AI_6 AI_7 AI_8

                  RES_FLT_4 = SUBSTRACT AI_1 AI_2 AI_3 AI_4 AI_5 22.5


// Multiply and divide keywords can take only up to 10 keywords variables and
// constants to be be calculated
Example:          RES_FLT_1 = * AI_1 AI_2 AI_3 AI_4 AI_5 AI_6

                  RES_FLT_2 = / AI_1 AI_2 AI_3 AI_4 AI_5 AI_6

                  RES_FLT_3 = MULTIPLY AI_1 AI_2 AI_3 AI_4 AI_5 AI_6

                  RES_FLT_4 = DIVIDE AI_1 AI_2 AI_3 AI_4 AI_5 AI_6
```

Keyword: AND
Keyword: NAND
Keyword: OR
Keyword: NOR
Keyword: XOR
Keyword: NXOR
Keyword: INVERT
Keyword: AND_OR
Keyword: ANO_OR_INVERTED

GB	FR	ES	DE	IT	NL
AND	ET	Y	UND	E	EN
NAND	ET_INVERSE	Y_NEGADO	UND_UMDREHT	E_ROVESCIATO	EN_OMZETTEND
OR	OU	O	ODER	O	OF
NOR	OU_INVERSE	O_NEGADO	ODER_UMDREHT	O_ROVESCIATO	OF_OMZETTEND
XOR	XOR	O_EXCLUSIVO	ODER_EXCLUSIF	O_EXCLUSIVO	OZ_EXCLUSIEF
NXOR	XOR_INVERSE	O_EXCLUSIVO_NEGADO	ODER_EXCLUSIF_UMDREHT	O_EXCLUSIVO_ROVESCIATO	OZ_EXCLUSIEF_OMZETTEND
INVERT	INVERSER	INVERTIR	UMDREHT	ROVESCIATO	OMZETTEND
AND_OR	ET_OU	Y_O	UND_ODER	E_O	EN_OF
AND_NOR	ET_OU_INVERSE	Y_O_NEGADO	UND_ODER_UMDREHT	E_O_ROVESCIATO	EN_OF_OMZETTEND

No PLC or controller would be complete without a good and powerful set of Boolean logic, so this section covers the Boolean instruction capabilities of the script compiler.

Remember that even when a different language can be selected for the compiler, the English keywords which are the native script language will always be available. This is noted as many programmers working with Boolean logic still prefer to use the Standard English keyword as this is what most schools in many countries use.

Boolean equations can be created using the following syntax:

```
Syntax:     [RESULT REGISTER] = AND  [OPERAND_1] [OPERAND_2] ..
                        Optional up to 20 operands can follow

Syntax:     [RESULT REGISTER] = NAND [OPERAND_1] [OPERAND_2] ..
                        Optional up to 20 operands can follow

Syntax:     [RESULT REGISTER] = OR   [OPERAND_1] [OPERAND_2] ..
                        Optional up to 20 operands can follow

Syntax:     [RESULT REGISTER] = NOR  [OPERAND_1] [OPERAND_2] ..
                        Optional up to 20 operands can follow

Syntax:     [RESULT REGISTER] = XOR  [OPERAND_1] [OPERAND_2]

Syntax:     [RESULT REGISTER] = NXOR [OPERAND_1] [OPERAND_2]

Syntax:     [RESULT REGISTER] = INVERT [OPERAND_1]
```

These last two compound Boolean instructions as well as the SET/RESET and START/STOP with EMERGENCY STOP and HYSTERESIS instructions were added to the PLC on version **2.68.** Therefore to use the script compiler it will be required to upgrade the firmware of the controller before using the compiler and the **eZ HVAC App wizard** if the version loaded in it is less than 2.68

```
Syntax:     [RESULT REGISTER] = AND_OR     [OPERAND_1] [OPERAND_2] \
                                           [OPERAND_3] [OPERAND_4]

Syntax:     [RESULT REGISTER] = AND_NOR    [OPERAND_1] [OPERAND_2] \
                                           [OPERAND_3] [OPERAND_4]

Examples:
   // Up to 20 operands allowed for these Boolean instructions
   BO_1 = AND      BI_1 BI_2 BI_3 BI_4 BI_5 BI_6 BI_7 BI_8 BI_9 BI_10
   BO_1 = AND      BI_1 BI_2 BI_3 BI_4 BI_5 BI_6 BI_7 BI_8
   BO_1 = AND      BI_1 BI_2 BI_3 BI_4 BI_5 BI_6 BI_7
   BO_1 = AND      BI_1 BI_2 BI_3 BI_4
   BO_1 = AND      BI_1 BI_2 BI_3
   BO_2 = NAND     BI_1 BI_2 BI_3 BI_4 BI_5 BI_6 BI_7 BI_8 BI_9 BI_10
   BO_3 = OR       BI_1 BI_2 BI_3 BI_4 BI_5 BI_6 BI_7 BI_8 BI_9 BI_10
   BO_4 = NOR      BI_1 BI_2 BI_3 BI_4 BI_5 BI_6 BI_7 BI_8 BI_9 BI_10

   // Invert keyword takes one operand only
   RES_BIT_1 =     INVERT      BI_1
   BO_1 =          INVERT      BI_1
   RES_BIT_1 =     INVERT      0
   AO_1_1 =        INVERT      1.1

   // XOR / NXOR keywords take two operands only
   RES_BIT_1 = XOR      BI_1 BI_2
   BO_1 =      XOR      BI_1 BI_2
   RES_BIT_1 = NXOR     BI_1 BI_2
   BO_1 =      NXOR     BI_1 BI_2

   // Compound AND-OR keywords take always four operands,
   // Unused terms can be set to: 1
   RES_BIT_1 = AND_OR   BI_1 BI_2 BI_3 BI_4
   BO_1 =      AND_OR   BI_1 BI_2 BI_3 BI_4
   RES_BIT_1 = AND_NOR  BI_1 BI_2 BI_3 BI_4
   BO_1 =      AND_NOR  BI_1 BI_2   1   BI_4 // Unused AND term is set to "1"
```

Remember that always user defined labels with the DEFINE keyword can be used instead of the database object names to make the syntax of the sentences more clear.

Keyword: SET
Keyword: RESET
Keyword: INSIDE
Keyword: OUTSIDE
Keyword: HYSTERESIS
Keyword: START
Keyword: STOP
Keyword: EMERG_STOP

GB	FR	ES	DE	IT	NL
SET	SET	FIJAR_A_1	SETZEN	SET	SET
RESET	RESET	FIJAR_A_0	ZURUECKSETZEN	RESET	RESET
START	DEMARRER	ARRANQUE	STARTEN	PARTENZA	STARTEN
STOP	ARRETEZ	PARO	STOPPEN	ARRESTO	STOPPEN
EMRG_STOP	ARRET_D_URGENCE	PARO_EMERG	NOT_HALT	ARRESTO_EMERG	NOODSTOP
INSIDE	INTERIEUR	DENTRO_DE	INNEN	DENTRO	BINNEN
OUTSIDE	EXTERIEUR	FUERA_DE	AUSSEN	FUORI	BUITEN
HYSTERESIS	HYSTERESE	HISTERESIS	HYSTERESE	ISTERESI	HYSTERESIS

The 'SET/RESET' command can be used to set or reset any binary register or output data base object . The 'START/STOP/EMRG_STOP' keywords can be used to set a start / stop circuit with emergency stop using the following syntax:

```
Syntax:     [RESULT REGISTER] =     SET         [OPERAND_1]       \
                                    RESET       [OPERAND_2]

Syntax:     [RESULT REGISTER] =     START       [OPERAND_1]       \
                                    STOP        [OPERAND_2]       \
                                    EMRG_STOP   [OPERAND_3]

// Start and Stop circuit emulator with emergency stop
Example:    BO_2 =      START BI_1   STOP   BI_2   EMRG_STOP    BI_3

// Simple hysteresis expression for HVAC can be performed
// with SET-RESET equations
Example:    BO_1 =      SET   BI_1   RESET BI_2
```

The 'INSIDE/OUTSIDE/HYSTERESIS' keywords can be to test if a process variable is inside or outside a given range. The INSIDE keyword is similar to a dual comparison such as if LOW < PV < HIGH so if the value of PV is 'inside' the low and high limits the result will be TRUE. The OUTSIDE keyword is similar to a dual comparison such as if LOW > PV > HIGH, so if the value of PV is 'outside' the low and high limits the result will be TRUE. The INVERT keyword at the end will invert the result of the comparison.

The 'INSIDE/OUTSIDE/HYSTERESIS' keywords can be to test if a process variable is inside or outside a given range using the following syntax:

The HYSTERESIS keyword is similar but checks only the crossing of the minimum and maximum thresholds, so for example will set the output to TRUE when the PV crosses the maximum level and will only return to FALSE after the PV goes down and crosses the minimum set point, this is very useful for HVAC applications to provide a dead band and avoid cycling

The difference between using standard nested IF keywords against these instructions, is code saving, as IF instructions might take as much as six to ten instructions to do the comparisons and the respective jumps to skip the output codes. These instructions do the job using a single instruction.

```
Syntax:      [RESULT REGISTER] = INSIDE    [DATA BASE OBJECT FOR MIN]    \
                                           [DATA BASE OBJECT FOR MAX]    \
                                           [DATA BASE OBJECT FOR PV]     \
                                           [Optional INVERT keyword]

Syntax:     [RESULT REGISTER] = OUTSIDE    [DATA BASE OBJECT FOR MIN]    \
                                           [DATA BASE OBJECT FOR MAX]    \
                                           [DATA BASE OBJECT FOR PV]     \
                                           [Optional INVERT keyword]

Syntax:     [RESULT REGISTER] = HYSTERESIS [DATA BASE OBJECT FOR MIN]    \
                                           [DATA BASE OBJECT FOR MAX]    \
                                           [DATA BASE OBJECT FOR PV]

// Output BO_1 will be TRUE if AI_1 > ADF_1 and AI_1 < ADF_2 (normal logic)
Example:    BO_1 = INSIDE ADF_1 ADF_2 AI_1

// Output BO_1 will be FALSE if AI_1 < ADF_1 or AI_1 > ADF_2 (inverted logic)
Example:    BO_2 = INSIDE ADF_1 ADF_2 AI_1 INVERT

// Output BO_1 will be TRUE if AI_1 < ADF_1 or AI_1 > ADF_2 (normal logic)
Example:    BO_1 = OUTSIDE ADF_1 ADF_2 AI_1

// Output BO_1 will be FALSE if AI_1 > ADF_1 and AI_1 < ADF_2
// (inverted logic)
Example:    BO_1 = OUTSIDE ADF_1 ADF_2 AI_1 INVERT

// Output BO_1 will be TRUE the moment AI_1 > ADF_2 and will remain so
// until AI_1 < ADF_1 at which point will be FALSE
Example:    BO_1 = HYSTERESIS ADF_1 ADF_2 AI_1
```

The HYSTERESYS does not have an inverted output logic incorporated in the instruction, if needed it should be inverted after the instruction is evaluated HYSTERESIS with an INVERT keyword .

Keyword: SCHEDULE

GB	FR	ES	DE	IT	NL
SCHEDULE	CALENDRIER	HORARIO	ZEITPLAN	ORARIO	DIENSTREGELING
ON	ALLUME	ENCENDIDO	EINSCHALT	ACCENDE	AANDOEN
OFF	ETEINDRE	APAGADO	AUSSCHALT	SPENGE	UITDOEN
PERIOD	PERIODE	PERIODO	PERIODE	PERIODO	PERIODE
DAY	JOUR	DIA	TAG	GIORNO	DAG
WEEK	SEMAINE	SEMANA	WOCHE	SETTIMANA	WEEK
MONTH	MOIS	MES	MONAT	MESE	MAAND

The OpenBAS NX controllers feature a real time clock and can handle up to 400 different schedules. With the SCHEDULE keyword weekly or specific date schedules can be created.

Different type of schedules can be created:

- Turn ON
- Turn OFF
- Turn On and Off at specific times
- Maintain On inside a specific period
- Set a value based on a schedule.

Weekly schedules

Weekly schedules can be created using a week day flag to make them active on any day from Monday to Sunday and including holidays using the following syntax.

```
Syntax:     [RESULT REGISTER] = SCHEDULE [WEEK]                          \
                  {Week day flags 7 days + holiday in the form [MTWTFSSH]\
                   not used days should be '-'}                           \
                  [ON or OFF]                                             \
                  [time 00:00 to 23:59]

Syntax:     [RESULT REGISTER] = SCHEDULE [WEEK]                          \
                  {Week day flags 7 days + holiday in the form [MTWTFSSH]\
                   not used days should be '-'}                           \
                  [ON] [time 00:00 to 23:59]                              \
                  [OFF] [time 00:00 to 23:59]                             \
                  [PERIOD is optional to force during period to on]

Syntax:     [RESULT REGISTER] = SCHEDULE [WEEK]                          \
                  {Week day flags 7 days + holiday in the form [MTWTFSSH]\
                   not used days should be '-'}                           \
                  [VALUE] [Scheduled commanded value or set point]        \
                  [time 00:00 to 23:59]
```

Specific date schedules

Specific date schedules use a day / month information to operate on that specific day of the year.

```
Syntax:     [RESULT REGISTER] = SCHEDULE [DAY] [Day of month]     \
                  [MONTH]     [month of year]                     \
                  [ON or OFF] [time 00:00 to 23:59]

Syntax:     [RESULT REGISTER] = SCHEDULE [DAY] [Day of month]     \
                  [MONTH]     [month of year]                     \
                  [ON]        [time 00:00 to 23:59]               \
                  [OFF]       [time 00:00 to 23:59]               \
                  [PERIOD is optional to force during period to on]

// Outputs or bit registers can be commanded by schedule
Example:    BO_1 = SCHEDULE WEEK {MTWTF---} ON  {8:00}
            BO_1 = SCHEDULE WEEK {MTWTF---} OFF {17:45}

// A schedule with ON and OFF times can be also set
Example:    BO_1 = SCHEDULE WEEK {MTWTF---} ON {8:00} OFF {18:30}

// Or a period of time during which the output will be commanded to ON
Example:    BO_1 = SCHEDULE WEEK {MTWTFS--} ON {8:00} OFF {18:30} PERIOD

// Also set points and values can be changed at a given time to pre-set value
Example:    ADF_1 = SCHEDULE WEEK {MTWTF--H} VALUE   30 {15:00}

// Specific dates can be created instead of weekly schedules
Example:    BO_1 = SCHEDULE DAY 30 MONTH   1   ON {15:00}
            BO_1 = SCHEDULE DAY 30 MONTH   1   OFF {22:15}

// Specific dates can be created instead of weekly schedules
Example:    BO_1 = SCHEDULE DAY 30 MONTH   1  ON {15:00} OFF {18:30}

// Specific dates can be created instead of weekly schedules
Example:    BO_1 = SCHEDULE DAY 30 MONTH 1 ON {15:00} OFF {18:30} PERIOD

// Specific dates can be created instead of weekly schedules
Example:    ADF_1 = SCHEDULE DAY 30 MONTH 1 VALUE 22.5 {15:00}
```

Keyword: TIMER

Keyword: OSCILATOR

Keyword: FREQUENCY

GB	FR	ES	DE	IT	NL
TIMER	MINUTEUR	TEMPORIZADOR	TIMER	TIMER	TIMER
OSCILATOR	OSCILLATEUR	OSCILADOR	OSZILLATOR	OSCILLATORE	OSCILLATOR
FREQUENCY	FREQUENCE	FRECUENCIA	FREQUENZ	FREQUENZA	FREQUENTIE
SECONDS	SECONDES	SEGUNDOS	SEKUNDEN	SECONDI	SECONDS
SEC_1_10	SEC_1_10	SEG_1_10	SEK_1_10	SEC_1_10	SEC_1_10

The 'TIMER' keyword is used to configure a timer´s operation mode, each controller has sixteen system timers that can be configured as:

- Second down counter
- 1/10 of second down counter
- Oscillator with pulse output
- Frequency converter with pulse output

The timers can be used alone or in conjunction with special purpose instructions for HVAC applications to synchronize events, the following examples will explain how to set them up.

```
// The result register is the timer status down counter timer
Syntax:     [RESULT REGISTER] = TIMER [timer number 1 to 16]
                  [ADI_1..100] [reload value] [SECONDS or SEC_1_10]

// Outputs the oscillator period to the binary result register
Syntax:     [RESULT REGISTER] = TIMER [timer number 1 to 16]
                  [OSCILATOR] [OPERAND_1 to convert to a period]

// Outputs the frequency to the the binary result register
Syntax:     [RESULT REGISTER] = TIMER [timer number 1 to 16]
                  [FREQUENCY] [OPERAND_1 to convert to frequency]

// Prepare timer reload value and set default value into ADI reload register
// SECONDS or SEC_1_10 flag can be used
Example:    TMR_1 = TIMER 1 ADI_1 5 SECONDS

// A binary output or result bit can be made to toggle at a given frequency
Example:    BO_1 = TIMER 1 OSCILATOR 5

// Or instead oscillate at a fixed or variable rate
Example:    BO_1 = TIMER 1 OSCILATOR AI_1

// A frequency (1/OSCILATOR) can also be used.
Example:    RES_BIT_1 = TIMER 1 FREQUENCY AI_1
```

Keyword: TOTALIZE

GB	FR	ES	DE	IT	NL
TOTALIZE	TOTALISATEUR	TOTALIZADOR	ZAEHLER	TOTALIZZATORE	TOTALIZATOR
PV	VP	VP	ISTWERT	VP	PV
ON_CHANGE	SUR_LE_CHANGEMENT	AL_CAMBIAR	BEI_AENDERUNG	SUL_CAMBIAMENTO	OP_VERANDERING
LAST_PERIOD	DERNIERE_PERIODE	ULTIMO_PERIODO	LETZTE_PERIODE	ULTIMO_PERIODO	LAATSTE_PERIODE
PARTIAL_KW	PARTIELLE_KW	PARCIAL_KW	PARTIELL_KW	PARZIALE_KW	PARTIEEL_KW
PARTIAL_ACC	PARTIELLE_ACCUM	PARCIAL_ACUM	PARTIELL_AKKUMULIERT	PARZIALE_ACUMULATO	PARTIEEL_GEACCUMULEERDE
SAMPLE_COUNT	NUMERO_ECHANTILLON	NUMERO_MUESTRAS	PROBENZAEHLUNG	CONTEGGIO_CAMPIONI	MONSTER_TELLER

Totalizers can also be called counters, as they keep count of events or quantities; they are used for many different purposes such as:

- The number of times a motor started, or stopped or both
- The number of times a door or window was opened, closed or both.
- The number of people entering or exiting a building, or both.
- The amount of liters used when using a pulsed water flow meter.
- The amount of gas or energy used when using pulsed gas or energy meters.
- Adding a period can convert the previous readings into for example liters per minute, cubic feet per hour, kilowatts hour, etc.
- Also analog signals can be used for example to convert current or watts to energy readings and store KVAh or KWh.

Setting up a totalizer is straightforward, and takes the following forms:

```
Syntax: [EEPROM REGISTER updated every 5 minutes] = TOTALIZE
            PV              = [OPERAND_1 process variable]
            ON_CHANGE       = [change to 0, change to 1, 2 = any change]
            PARTIAL_ACC     = [OPERAND_2 5 minute partial accumulated value]

Syntax: [RESULT REGISTER] = TOTALIZE
            PV              = [OPERAND_1 process variable]
            ON_CHANGE       = [change to 0, change to 1, 2 = any change]
            LAST_PERIOD     = [OPERAND_2 to store last period]
            MIN             = [totalisation period 1,5,10,20,30,60 minutes]

Syntax: [EEPROM REGISTER updated KWH every 10 minutes]   = TOTALIZE
            PV              = [OPERAND_1 process variable]
            PARTIAL_KW      = [OPERAND_2 partial sum of 10 min accum. KWh]
            SAMPLE_COUNT    = [OPERAND_3 nr. of samples per 10 min. period]
```

The only limit on the amount of totalizers is the number of registers available to store the totalized counters.

On the first example, an EEPROM register will be updated every 5 minutes with the partial accumulated count that is stored in RAM, this is done because EEPROM's have a limited number of write cycles (1,000,000) and stressing it beyond that point will damage the memory cell.

```
// Totalisation (accumulation) of pulses when input changes to:
// to 0, to 1 or toggles (2) and with EEPROM backup can be created
Example: ADF_1 =  TOTALIZE     PV = BI_1                    \
                               ON_CHANGE = 1                \
                               PARTIAL_ACC = RES_FLT_1
```

On this next example a periodic count is kept in a pair of registers, rge result register keeps track of the total count of the current period and the last period register stores the last period.

So for example if we were measuring water and the period is set to 60 minutes, the result register will have the count of liters of the current hour and will be incrementing the counter every time a pulse that represents a certain amount of water is received. Meanwhile the last period will keep the record of liters used during the previous hour.

When the time period expires, the current period counter is reset to zero after having copied its current value to the last period register to start a new count.

```
// Totalisation per period of time between 1,5,10,20,30,60 minutes
Example: RES_FLT_1 = TOTALIZE PV = BI_1                     \
                              ON_CHANGE = 1                 \
                              LAST_PERIOD = RES_FLT_2       \
                              MIN    = 60
```

The last example is totalizator for energy; it´s input is an analog value that can be a current sensor that is multiplied by voltage to get VA (Volt-Amps) and optionally also multiplied by the power factor to get Watts. This value is sampled and a partial KW count is updated several times per second and the number of samples for the minute period is also stored.

At the end of each 10 minute period, the totalized energy which is stored in an EEPROM register is incremented by the energy stored in the 10 minute partial register and stored back. The 10 minute energy register as well as the sample counter are then reset to zero to start a new count

```
// Totalisation of Kilo Watts hours can be created using totalizers
Example: ADF_1 = TOTALIZE      PV = RES_FLT_1               \
                               PARTIAL_KW   = RES_FLT_1 \
                               SAMPLE_COUNT = RES_FLT_2
```

Keyword: HOUR_COUNTER

GB	FR	ES	DE	IT	NL
HOUR_COUNTER	COMPTEUR_D_HEURES	HOROMETRO	STUNDEN_ZAEHLER	CONTAORE	URENTELLER
PV	VP	VP	ISTWERT	VP	PV
PARTIAL_COUNT	COMPTE_PARTIEL	CUENTA_PARCIAL	TEILZAEHLUNG	CONTEGGIO_PARZIALE	GEDEELTELIJKE_TELLING
EEPROM	EEPROM	EEPROM	EEPROM	EEPROM	EEPROM

Hour counters keep track of time of operation of any given controlled device, such as a pump, an air handler unit, time a door or window is open or closed, or the time a light is on or a temperature is in alarm etc.

Later this time count can be used to trigger alarms or sequences based on hours of operation such as:

- Changing to alternate equipment to balance the hours of operation in machinery arrays.
- Issuing maintenance alarms.
- Warning that a filter needs to be replaced.
- Etc.

Setting up a counter is relatively easy, when the process variable is TRUE, the count is incremented, and otherwise it sits still. As with the totalizers, because the hours of operation is stored in EEPROM and it has a write life time of 1,000,000 cycles, a partial count is kept in a RAM based partial count, and the contents of the EEPROM are updated every 5 minutes with the most recent hour count.

The hour counter takes the following format:

```
Syntax:      [RESULT REGISTER] = HOUR_COUNTER
                 PV            = [OPERAND_1 process variable]
                 PARTIAL_COUNT = [OPERAND_2 partial 5 minutes counter]
                 EEPROM        = [OPERAND_3 eeprom updated every 5 minutes]

// An hour counter to keep track of any input / output being ON
// can be easily implemented
Example:    RES_FLT_1 = HOUR_COUNTER  PV = BO_1 PARTIAL_COUNT = RES_FLT_1 \
                        EEPROM = ADF_1
```

The only limit on the amount of counters is the number of registers available to store the hour count.

Keyword: PROP_CTRL

GB	FR	ES	DE	IT	NL
PROP_CTRL	CONTROLE_PROPORTIONNEL	CONTROL_PROP	PROP_BEDIENUNG	CONTROLLO_PROP	PROP_BEDIENING
PV	VP	VP	ISTWERT	VP	PV
SP	CONSIGNE	P_AJ	SOLLWERT	SP	SETPUNT
PB	B_PROP	B_PROP	PB	B_PROP	PB
MIN	MIN	MIN	MIN	MIN	MIN
MAX	MAX	MAX	MAX	MASSIMO	MAX
INTEG	INTEG	INTEG	INTEG	INTEG	INTEG
EEPROM	EEPROM	EEPROM	EEPROM	EEPROM	EEPROM

Proportional control instructions are used to modulate machinery that can vary a process such as:

- Variable frequency drives (VFD's).
- Dimmable lighting ballasts.
- Proportional water valves for heating and cooling coils.
- Damper actuators with proportional positioning, for HVAC applications.
- Etc.

The output of a proportional control will swing from 0-100% and it can be output to analog outputs for direct control of the above mentioned devices, or sent via communication channels to command VFD´s, dampers and actuators for example via Optomux, N2-Open, Modbus or BACnet.

The proportional control with minimum, maximum and integration has the following format:

```
Syntax:      [RESULT REGISTER] = PROP_CTRL
                        PV     = [Process variable]
                        EEPROM = [Index to EEPROM storage, ADF or ADI or ADB]
                        SP     = [OPERAND_1]    set point]
                        PB     = [OPERAND_2]    proportional band
                        MIN    = [OPERAND_3]    minimum output range
                        MAX    = [OPERAND_4]    maximum output range
                        INTEG  = [OPERAND_5]    integration time (ramp)

// Proportional controls to position dampers, valves or speed of VFD´s can be
// created on a single line, here the syntax with the '=' operator between
// keywords and values is shown
Example:  AO_1 =  PROP_CTRL          \
                  PV     = AI_1      \
                  EEPROM = ADF_1     \
                  SP     = 22.5      \
                  PB     = 1.5       \
                  MIN    = 10        \
                  MAX    = 90        \
                  INTEG  = 10
```

```
// Also the line can be written without the '=' operator
Example:    AO_1 = PROP_CTRL              \
                   PV         AI_1        \
                   EEPROM     ADF_1       \
                   SP         22.5        \
                   PB         1.5         \
                   MIN        10          \
                   MAX        90          \
                   INTEG      10

// Even a very simple line can also be created using only the operands
// Each parameter´s position is assumed to be in the before mentioned order
Example:    AO_1 = PROP_CTRL AI_1 ADF_1 22.5 1.5 10 90 10
```

The only limit on the proportional control instructions is the number of registers available to store the variables and set points.

Note that the result register for this instruction can only be analog outputs or REF_FLT registers. For example the output of the proportional control will be sent to a remote point to do control over a field bus protocol, after the proportional control instruction, a write to the remote point has to be issued separately. This is shown on the following example.

```
// First the proportional control is executed
Example:    RES_FLT_1 = PROP_CTRL AI_1 ADF_1 22.5 1.5 10 90 10
            RMT_1 = RES_FLT_1 // Now the value can be sent to the field bus
                              // remote point, every time a remote point is
                              // the left side of an assignment operation,
                              // it will be written instead of read in the
                              // bus using the selected protocol.
```

More information on setting up remote points using the RMT keyword as well as configuring field buses with the COMM keyword to operate as either slaves or masters is given their corresponding sections on this user's guide.

Keyword: HVAC_STAGE

GB	FR	ES	DE	IT	NL
HVAC_STAGE	ETAPE_HVAC	ETAPA_HVAC	HVAC_STUFE	STADIO_HVAC	HVAC_ETAPPE
COOLING	REFROIDISSEMENT	ENFRIAMIENTO	KUEHLUNG	REFFREDDAMENTO	KOELING
HEATING	CHAUFFAGE	CALEFACCION	HEIZUNG	RISCALDAMENTO	VERWARMING
SP	CONSIGNE	P_AJ	SOLLWERT	SP	SETPUNT
PB	B_PROP	B_PROP	PB	B_PROP	PB
PV	VP	VP	ISTWERT	VP	PV
STAGE_RUN	VALIDACION_ETAPE	PERMISIVO_ETAPA	STUFE_LAUF	STADIO_FUNZIONARE	ETAPPE_LOPEN
TMR_MINIMUM_ON	TMR_MIN_ALLUMER	TMR_MINIMO_ENC	TMR_MIN_EIN	TMR_MIN_ACCENDERE	TMR_MIN_IN
TMR_MINIMUM_OFF	TMR_MIN_ARRET	TMR_MINIMO_APAG	TMR_MIN_AUS	TMR_MIN_SPEGNERE	TMR_MIN_UIT
TMR_INTERSTAGE	TMR_INTERETAPES	TMR_INTERETAPAS	TMR_ZWISCHENSTUFE	TMR_INTERSTADIO	TMR_INTERSTAGE

Powerful yet simple HVAC control sequences can be built using this application specific HVAC_STAGE instruction. In the programmer manual for the OpenBAS NX controllers some additional information and examples are given to fully implement air handler units control sequences. The **eZ App wizard** goes a step further by fully automating the build process for simple to complicated air handling units.

The HVAC stage takes the following format:

```
Syntax:     [RESULT REGISTER] = HVAC_STAGE
                  [COOLING or HEATING flag]
                  EEPROM      [Index to EEPROM storage, ADF or ADI or ADB]
                  SP                        [OPERAND_1]
                  PB                        [OPERAND_2]
                  PV                        [OPERAND_3]
                  STAGE_RUN                 [OPERAND_4]
                  TMR_MINIMUM_ON_SEC        [OPERAND_5] **optional
                  TMR_MINIMUM_OFF_SEC       [OPERAND_6] **optional
                  TMR_INTERSTAGE            [OPERAND_7] **optional

// Timers are included for: minimum On time, minimum Off time
// as well as inter-stage, the timers have to be independently created
// using the TIMER keywords
Example:    BO_1 = HVAC_STAGE       COOLING     \
                  EEPROM            = ADF_1     \
                  PV                = AI_1      \
                  SP                = 22.5      \
                  PB                = 2         \
                  STAGE_RUN         = BI_1      \
                  TMR_MINIMUM_ON    = TMR_1     \
                  TMR_MINIMUM_OFF   = TMR_1     \
                  TMR_INTERSTAGE    = TMR_3
```

```
// Simple stage control
Example:     BO_2 = HVAC_STAGE        HEATING      \
                    PV                = AI_1       \
                    SP                = 22.5       \
                    PB                = 2          \
                    STAGE_RUN         = BI_1       \
                    EEPROM            = ADF_7
```

Air handling units are very diverse mechanically and in operation sequence. This stage instruction can be linked together to make any control no matter how complex it is.

Basically you provide the stage with the following information so it can perform its job:

- A binary result register that will be turned On/Off as needed
- Information weather the stage is intended for Heating or Cooling. This can be changed at run time
- A process variable that is the feedback to control the result register output based on the set point and proportional band, that in this case is used as a differential to provide hysteresis as to when the stage has to be On or Off.
- A stage run signal, that usually comes from a Boolean AND instruction and has all the chained information to let the stage run such as: emergency stop, low pressure cut off, high pressure cut off, water flow present, manual, scheduled or remote start command.
- There are two timers that can optionally set a minimum ON and minimum OFF time, this is very important when driving compressors, motors and other load that are sensitive to frequent start and stop cycles.
- An inter-stage timer that links together multiple stages, so not all of them start simultaneously.

The **eZ App wizard** generates tested solutions that can be further personalized to suit any kind of simple to complex solutions.

The HVAC stage instruction is not limited only to air handler applications, as the operands are generic and can control any variable such as: pressure, current, energy, water or gas flow, etc.

Keyword: ALTERNATE

GB	FR	ES	DE	IT	NL
ALTERNATE	ALTERNER	ALTERNADO	ALTERNIEREN	ALTERNATO	AFWISSELEN
PV	VP	VP	ISTWERT	VP	PV
SP	CONSIGNE	P_AJ	SOLLWERT	SP	SETPUNT
PB	B_PROP	B_PROP	PB	B_PROP	PB
EEPROM	EEPROM	EEPROM	EEPROM	EEPROM	EEPROM
ALT_OUTPUT	ALT_SORTIE	ALT_SALIDA	ALT_AUSGANG	ALT_USCITA	ALT_UITGANG
ALT_STAGES	ALT_ETAPES	ALT_ETAPAS	ALT_STUFEN	ALT_STADIO	ALT_ETAPPE
ALT_LEADER	ALT_LEADER	ALT_LIDER	ALT_FUHRER	ALT_DIRETTORE	ALT_AANVOERDER
ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC	ALT_DEC_INC
ALT_TMR_NEXT_STAGE	ALT_TM_SUIV_ETAPE	ALT_TM_SIG_ETAPA	ALT_NACHSTE_STUFE	ALT_PROSSIMO_STADIO	ALT_VOLGENDE_ETAPPE
ALT_TMR_ALARM	ALT_TM_ALARME	ALT_TM_ALARMA	ALT_TMR_ALARM	ALT_TMR_ALLARME	ALT_TMR_ALARM
ALT_EXT_ENABLE	ALT_ACTIV_EXT	ALT_HABIL_EXT	ALT_EXT_FREIGABE	ALT_ABILITAZIONE_EST	ALT_EXT_SCHAKELEN
ALT_FEEDBACK	ALT_RETOUR	ALT_RETROALIM	ALT_RUCKKOPPLUNG	ALT_RISPOSTA	ALT_TERGUGKOPPELING
ALT_PARALLEL	ALT_PARALLELE	ALT_SIMULTANEO	ALT_PARALLEL	ALT_PARALLELO	ALT_PARALLEL

The ALTERNATE instruction is a simple but yet powerful instruction that can works by alternating machinery such as: pumps, ventilators, chillers, boilers, AHU´s etc.

It is used to balance operation of the before mentioned equipment, but not also alternates but can also be programmed to parallel operation, so if the first stage can´t cope with the load, such as in pump applications, a second, third and up to eight stages can be paralleled if necessary.

It has advanced features such as:

- Stage feedback to look for not operating stages.
- External enable to switch out stages that might be in maintenance or are out of order.
- Timers to sequence the operation of the next stage or detect faulty stages.
- Incremental or decremented staging.
- Can be set to work with analog process variables such as analog pressure transducers or binary type such as pressure switches for complex or simple operation.

Pump alternate and paralleling can be created using the following syntax:

```
Syntax:     BO_1 = ALTERNATE
                   PV                 [OPERAND_1]
                   ALT_STAGES         [OPERAND_2] 2 to 8
                   EEPROM             [Index to EEPROM, ADF or ADI or ADB]
                   ALT_DEC_INC        [OPERAND_3] 0/1          **optional
                   ALT_LEADER         [OPERAND_4] 1 to 8       **optional
                   ALT_TMR_NEXT_STAGE[OPERAND_5] 1 to 16       **optional
                   ALT_TMR_ALARM      [OPERAND_6] 1 to 16      **optional
                   SP                 [OPERAND_7]              **optional
                   PB                 [OPERAND_8]              **optional
                   ALT_FEEDBACK       [OPERAND_9]              **optional
                   ALT_EXT_ENABLE     [OPERAND_10]             **optional
```

In the first example we can see how to set up a simple four stage pump array alternator. In this case the process variable is a pressure switch connected to binary input 1 that would start the leader pump in sequence every time the pressure drops so it closes, and stops them when the pressure rises and the pressure switch opens.
If the pressure switch operates inverse a simple invert instruction can be used to invert the signal that is fed into binary input 1.

```
// Simplest construction with binary PV
Example:    BO_1 = ALTERNATE              \
                   PV          = BI_1     \
                   ALT_STAGES  = 4        \
                   EEPROM      = ADF_1
```

In this other example an analog pressure transducer connected to analog input 1 is used to start the current leader pump when the pressure falls below 100 PSI and conversely to stop it and change the pump sequence when it reaches 120 PSI.

```
// Simplest construction with analog PV, set point and proportional band have
// to be provided as well
Example:    RES_BIT_1 = ALTERNATE
                   PV          = AI_1     \
                   ALT_STAGES  = 4        \
                   EEPROM      = ADF_1    \
                   SP          = 100      \
                   PB          = 15
```

All other parameters of the ALTERNATE keyword are optional and can be added to the line in any order. They work to personalize the operation of the ALTERNATE module. For more information refer to the OpenBAS NX programmer's manual to get examples of wiring and online programming of this instruction.

```
ALT_PARALLEL        = 0 = alternate, 1 = alt + parallel
ALT_DEC_INC         = 0 = decrement, 1 = increment sequence
ALT_LEADER          = 2 to 8
ALT_FEEDBACK        = BI_1    Starting feedback input
ALT_EXT_ENABLE      = BI_2    Starting external enable inp.
ALT_TMR_NEXT_STAGE  = TMR_10  Timers have to be set with
ALT_TMR_ALARM       = TMR_2   the TIMER keyword in
                              individual lines with their
                              corresponding time values
```

NOTE: Every time an ALTERNATE module is set up or its operating features other than the set point or proportional band are modified, it is recommended to reset the OpenBAS controller to reset the internal status sequence of operation of this instruction.

Keyword: TREND

GB	FR	ES	DE	IT	NL
TREND	TENDANCE	TENDENCIA	TENDENZ	TENDENZA	TREND

The OpenBAS NX controllers have the capability to log data and then later with the aid of the software this logged data can be retrieved to create graphics. The TREND keyword adds in setting up to sixteen available trends. Trend data takes the following format:

```
Syntax:      TREND [OPERAND 1]
             INTERVAL_MINUTES = [1, 5, 10, 15, 20, 30, 60 minutes]

// Trending made easy up to 16 trends at: 1,5,10,15,20,30,60 minute interval
Example:     TREND AI_1 INTERVAL_MINUTES = 15

// Trending for binaries, only change of state (COS) is detected and stored
Example:     TREND BI_1
```

Standard OpenBAS NX controllers can store up to 124 samples for analog values at the programmed interval, so for example if 60 minutes is selected, the last 124 hours or five days will be stored.

The following table gives storage capacity at different intervals:

Using standard storage

Save interval	Hours of storage	Days of storage
1	2.1	0.1
5	10.3	0.4
10	20.7	0.9
15	31.0	1.3
20	41.3	1.7
30	62.0	2.6
60	124.0	5.2

Using expanded memory

Save interval	Hours of storage	Days of storage
1	16.7	0.7
5	83.3	3.5
10	166.7	6.9
15	250.0	10.4
20	333.3	13.9
30	500.0	20.8
60	1,000.0	41.7

Note: 1 minute interval saving reverts to 5 minutes after 124 samples to protect EEPROM memory

For binary values, the last 248 state changes are stored without regard to any sampling rate, so if for example a variable changes states twice a day, the buffer will contain data of 124 days or four months.

If 128K or 256K expanded memory or a dual core is installed, the trend data logging is incremented to 1,000 samples.

If the dual core is installed in an installed USB memory the data of trends created with the TREND keyword will be stored on a one minute period with unlimited capability. The space of the USB memory should be sufficient to provide for years of storage.

Keyword: COMM

GB	FR	ES	DE	IT	NL
COMM	COMM	COMM	COMM	COMM	COMM
DEVICE_ADDRESS	ADDRESSE_COM	DIRECCION_COM	REGLERADRESSE	INDIRIZZO_DISPOSITIVO	CONTROLLER_ADRES
PARITY_NONE	PARITE_SANS	PARIDAD_SIN	PARITAT_KEINE	PARITA_NESSUNA	PARITEIT_NONE
PARITY_ODD	PARITE_IMPAIRE	PARIDAD_NON	PARITAT_UNGERADE	PARITA_DISPARI	PARITEIT_ONEVEN
PARITY_EVEN	PARITE_PAIRE	PARIDAD_PAR	PARITAT_SOGAR	PARITA_PARI	PARITEIT_EVEN
STOP_BIT_0	BIT_D_ARRET_0	BIT_PARO_0	STOP_BIT_0	STOP_BIT_0	STOP_BIT_0
STOP_BIT_1	BIT_D_ARRET_1	BIT_PARO_1	STOP_BIT_1	STOP_BIT_1	STOP_BIT_1
BAUD_RATE	BAUD	BAUDIOS	BAUD	BAUD	BAUD
PROTOCOL_ASCII_TERMINAL	PROTOCOLE_ASCII_TERMINAL	PROTOCOLO_ASCII_TERMINAL	PROTOKOLL_ASCII_TERMINAL	PROTOCOLLO_ASCII_TERMINAL	PROTOCOL_ASCII_TERMINAL
PROTOCOL_OPTO_22_SLAVE	PROTOCOLE_OPTO_22_ESCLAVE	PROTOCOLO_OPTO_22_ESCLAVO	PROTOKOLL_OPTO_22_SLAVE	PROTOCOLLO_OPTO_22_SCHIAVO	PROTOCOL_OPTO_22_SLAAF
PROTOCOL_N2_OPEN_SLAVE	PROTOCOLE_N2_OPEN_ESCLAVE	PROTOCOLO_N2_OPEN_ESCLAVO	PROTOKOLL_N2_OPEN_SLAVE	PROTOCOLLO_N2_OPEN_SCHIAVO	PROTOCOL_N2_OPEN_SLAAF
PROTOCOL_MODBUS_SLAVE	PROTOCOLE_MODBUS_ESCLAVE	PROTOCOLO_MODBUS_ESCLAVO	PROTOKOLL_MODBUS_SLAVE	PROTOCOLLO_MODBUS_SCHIAVO	PROTOCOL_MODBUS_SLAAF
PROTOCOL_BANCET_MSTP	PROTOCOLE_BANCET_MSTP	PROTOCOLO_BANCET_MSTP	PROTOKOLL_BANCET_MSTP	PROTOCOLLO_BANCET_MSTP	PROTOCOL_BANCET_MSTP
PROTOCOL_OPTO22_MASTER	PROTOCOLE_OPTO22_MAITRE	PROTOCOLO_OPTO22_MAESTRO	PROTOKOLL_OPTO22_MEISTER	PROTOCOLLO_OPTO22_MASTER	PROTOCOL_OPTO22_MEESTER
PROTOCOL_MODBUS_MASTER	PROTOCOLE_MODBUS_MAITRE	PROTOCOLO_MODBUS_MAESTRO	PROTOKOLL_MODBUS_MEISTER	PROTOCOLLO_MODBUS_MASTER	PROTOCOL_MODBUS_MEESTER

Standard OpenBAS NX controllers have two communication ports; dual core adds the third communication port. The COMM keyword is used to set up a communication port.

```
Default values are highlighted in green:
Syntax:     COMM [communication port 1, 2, 3]
                PROTOCOL            PROTOCOL_ASCII_TERMINAL, PROTOCOL_OPTO_22_SLAVE,
                                    PROTOCOL_N2_OPEN_SLAVE,  PROTOCOL_MODBUS_SLAVE,
                                    PROTOCOL_BANCET_MSTP,    PROTOCOL_OPTO22_MASTER,
                                    PROTOCOL_MODBUS_MASTER,
                DEVICE_ADDRESS      1 to 253
                BAUD_RATE           2400, 4800, 9600, 19200, 38400, 57600, 76800
                PARITY              PARITY_NONE, PARITY_ODD, PARITY_EVEN
                STOP_BIT            STOP_BIT_0, STOP_BIT_1
```

Note: Each one of the communication ports can be setup only one per script, so if a COMM keyword for a given communication port is used an error will be generated.

```
Example:    COMM 1 PROTOCOL_BANCET_MSTP     DEVICE_ADDRESS = 5    \
               BAUD_RATE = 38400

Example:    COMM 1 PROTOCOL_MODBUS_SLAVE    DEVICE_ADDRESS = 1    \
               BAUD_RATE = 9600  PARITY_ODD STOP_BIT_1

Example:    COMM 1 PROTOCOL_MODBUS_MASTER BAUD_RATE = 19200 PARITY_ODD

Example:    COMM 1 PROTOCOL_ASCII_TERMINAL  BAUD_RATE = 9600

Example:    COMM 1 PROTOCOL_OPTO_22_SLAVE   DEVICE_ADDRESS = 1

Example:    COMM 1 PROTOCOL_N2_OPEN_SLAVE   DEVICE_ADDRESS = 10
```

Each of the communication ports can be as master or slave of the supported protocols of each port.

```
Example:    COMM 2 PROTOCOL_ASCII_TERMINAL   BAUD_RATE = 9600

Example:    COMM 2 PROTOCOL_OPTO_22_SLAVE    DEVICE_ADDRESS = 5

Example:    COMM 2 PROTOCOL_N2_OPEN_SLAVE    DEVICE_ADDRESS = 5

Example:    COMM 2 PROTOCOL_OPTO22_MASTER    BAUD_RATE = 9600
```

On controllers with the dual installed the third communication port is available.

```
Example:    COMM 3 PROTOCOL_OPTO_22_SLAVE    DEVICE_ADDRESS = 5

Example:    COMM 3 PROTOCOL_N2_OPEN_SLAVE    DEVICE_ADDRESS = 5

Example:    COMM 3 PROTOCOL_MODBUS_SLAVE     DEVICE_ADDRESS = 5    \
                 BAUD_RATE = 9200   PARITY_NONE
```

Note: If the baud rate is changed, the new baud rate will not take effect until the controller is reset.

The reason for this is because the configuration of the communication port can be also done via the communication port itself. If baud rate change were performed during the communication port configuration there would be no way to finish and verify the configuration gracefully.

So the changes to the speed of the communication port will only take effect after a reset. Either hardware or software invoked.

Keyword: WIRELESS_LINK

GB	FR	ES	DE	IT	NL
WIRELESS_LINK	LIAISON_SANS_FIL	ENLACE_INALAMBRICO	DRAHTLOSE_VERBINDUNG	COLLEGAMENTO_SENZA_FILI	DRAADLOZE_VERBINDING
WLS_GROUP	WLS_GRUPE	WLS_GRUPO	WLS_GRUPPE	WLS_GRUPPO	WLS_GROEP
WLS_ADDRESS	WLS_ADDRESSE	WLS_DIRECCION	WLS_ADDRESSE	WLS_INDIRIZZO	WLS_ADRES

Standard OpenBAS NX controllers can receive information sent from up to ten wireless thermostats if the wireless I2C interface is installed.

The WIRELESS_LINK keyword aids in programming each one of the ten wireless links that can be active at any given time.

To Configure I2C wireless interface takes the format:

```
Syntax:     WIRELESS_LINK     [1 to 10]
            WLS_GROUP         [1 to 10]
            WLS_ADDRESS       [1 to 199]

// Up to 10 wireless links can be created with groups set between 1-10
// and addresses 1-199
Example:    WIRELESS_LINK = 1 WLS_GROUP = 1 WLS_ADDRESS = 1

// Up to 10 wireless links can be created with groups set between 1-10
// and addresses 1-199
Example:    WIRELESS_LINK 2 WLS_GROUP 5 WLS_ADDRESS 90
```

After the wireless links have been created, each of the parameters that the wireless transmitter sends can be added as remote points using the REMOTE keyword that will be explained shortly.

Keyword: REMOTE

GB	FR	ES	DE	IT	NL
REMOTE	ELOIGNE	REMOTO	FERNPUNKT	PUNTO_A_DISTANZA	AFGELEGEN
READ_COIL	READ_COIL	READ_COIL	READ_COIL	READ_COIL	READ_COIL
READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS	READ_INPUT_STATUS
READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER	READ_INPUT_REGISTER
READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER	READ_HOLDING_REGISTER
ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE	ANALOG_VALUE
BINARY_VALUE	BINARY_VALUE	BINARY_VALUE	BINARY_VALUE	BINARY_VALUE	BINARY_VALUE
NX_SLAVE	NX_ESCLAVE	NX_ESCLAVO	NX_SLAVE	NX_SCHIAVO	NX_SLAAF
DEVICE_ADDRESS	ADDRESSE_COM	DIRECCION_COM	REGLERADRESSE	INDIRIZZO_DISPOSITIVO	CONTROLLER_ADRES

Once a communication port has been set up as a master, the REMOTE points can be added so the master starts polling the information. Normally the master reads the set up remote points and keeps the remote table updated with the most recent value of the slave device.

However when a remote point is written via a communication port or the PLC issues a write to a remote point, the logic in the master port will instead write the remote point to the slave with the contents stores in the remote point value register.

Add remote points to the mapping table with the following format for wired remote points:

```
Syntax:     REMOTE
            COMX                Communication port COM1, COM2 or COM3
            [OPERAND_1]         Database object or protocol dependant type
            DEVICE_ADDRESS      Device address based on protocol limits
            [NAME]              Optionally a name can be given after the remote
                                point and it will be automatically added as a
                                label

Example:    REMOTE  COM1  READ_INPUT_REGISTER     1   DEVICE_ADDRESS 1 OAT
            REMOTE  COM1  READ_HOLDING_REGISTER   100 DEVICE_ADDRESS 1 Z_TEMP
            REMOTE  COM1  READ_COIL               25  DEVICE_ADDRESS 1 ALARM

            REMOTE  COM2  NX_SLAVE     DEVICE_ADDRESS    100    SLAVE_1
            REMOTE  COM2  NX_SLAVE     DEVICE_ADDRESS    101    SLAVE_2
            REMOTE  COM2  AI_1         DEVICE_ADDRESS    1      PRESSURE
            REMOTE  COM2  AI_2         DEVICE_ADDRESS    1
            REMOTE  COM2  BO_1         DEVICE_ADDRESS    5
            REMOTE  COM2  BO_10        DEVICE_ADDRESS    5

            REMOTE  COM3  NX_SLAVE     DEVICE_ADDRESS    103    ANOTHER_SLAVE
            REMOTE  COM3  NX_SLAVE     DEVICE_ADDRESS    104
            REMOTE  COM3  AI_1         DEVICE_ADDRESS    1      TEMPERATURE
            REMOTE  COM3  BI_1         DEVICE_ADDRESS    1      START_SIGNAL
```

Besides wired remote points that will communicate over the controllers field buses, OpenBAS controllers can also map wireless points that come into the system via the I2C wireless adapter. On the previous page prior to the REMOTE keyword we learned how to create WIRELESS_LINKS.

Keywords: Parameters to map wireless parameters into remote points

GB	FR	ES	DE	IT	NL
WIRELESS_LINK	LIAISON_SANS_FIL	ENLACE_INALAMBRICO	DRAHTLOSE_VERBINDUNG	COLLEGAMENTO_SENZA_FILI	DRAADLOZE_VERBINDING
WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C	WLS_TEMP_C
WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F	WLS_TEMP_F
WLS_REL_HUM	WLS_HUM_REL	WLS_HUM_REL	WLS_REL_FEUCHTIGKEIT	WLS_UMIDITA_REL	WLS_REL_VOCHTIGHEID
WLS_MODE	WLS_MODE	WLS_MODO	WLS_MODUS	WLS_MODALITA	WLS_MODE
WLS_FAN_SPEED	WLS_VEL_VENTILATEUR	WLS_VEL_VENTILADOR	'LS_LUFTER_GESCHWINDIGKE	WLS_VELOCITA_VENT	WLS_VENT_SNELHEID
WLS_KEYBOARD	WLS_CLAVIER	WLS_TECLADO	WLS_TESTATUR	WLS_TESTIERA	WLS_TOETSENBORD
WLS_SP_TEMP	WLS_PC_TEMP	WLS_PA_TEMP	WLS_SP_TEMP	WLS_SP_TEMP	WLS_SP_TEMP
WLS_SP_HUM	WLS_PC_HUM	WLS_PA_HUM	WLS_SP_FEUCHTIGKEIT	WLS_SP_UMIDITA	WLS_SP_VOCHTIGHEID
WLS_SP_T1	WLS_PC_T1	WLS_PA_T1	WLS_SP_T1	WLS_SP_T1	WLS_SP_T1
WLS_SP_PB	WLS_PC_BP	WLS_PA_BP	WLS_SP_PB	WLS_SP_PB	WLS_SP_PB
WLS_SP_UNOCC	WLS_PC_INOCCUPE	WLS_PA_DESOC	WLS_SP_UNBESETZ	WLS_SP_DISOCCUPATO	WLS_SP_ONBEZET
WLS_BATTERY_VOLTAGE	WLS_VOLTAGE_BATTERIE	WLS_VOLTAJE_BATERIA	WLS_BATTERIESPANNUNG	WLS_VOLTAGIO_BATTERIA	WLS_BATTERIJ_VOLTAGE
WLS_AUX_INP	WLS_ENT_AUX	WLS_ENT_AUX	WLS_AUX_EIN	WLS_AUX_ING	WLS_AUX_ING
WLS_LINK_TMR	WLS_TM_LIEN	WLS_TM_ENLACE	WLS_LINK_TMR	WLS_LINK_TMR	WLS_LINK_TMR
WLS_SEC_LINK_LOST	WLS_SEC_SANS_LIEN	WLS_SEC_SIN_ENLACE	WLS_SEK_KEIN_LINK	LS_SEC_SENSA_COLLEGAMEN	WLS_SEC_ZONDER_KOPPELING

Once a wireless link is created it is easy to add the parameters that the wireless transmitters have. They can be added to the REMOTE points using the following format so they will get mapped into the wired REMOTE points tabled and can be used elsewhere in the system:

```
Syntax:     REMOTE
            COM2              Only COM2 point table supports wireless points
            [OPERAND_1]       Database object or protocol dependant type
            WIRELESS_LINK     Make a reference to an existing wireless links

Example:    REMOTE  COM2  WLS_TEMP_C      WIRELESS_LINK    1
            REMOTE  COM2  WLS_TEMP_F      WIRELESS_LINK    1
            REMOTE  COM2  WLS_HUM_REL     WIRELESS_LINK    1

            REMOTE  COM2  WLS_TEMP_C      WIRELESS_LINK    2
            REMOTE  COM2  WLS_TEMP_F      WIRELESS_LINK    2
            REMOTE  COM2  WLS_HUM_REL     WIRELESS_LINK    2
```

There are the standard 50 remote points available with standard memory:

- **RMT_1...51**

420 additional remote points are available for dual core or if 32KKNV expansion memory is installed:

- **RMT_51…255**
- **RES_FLT_41...255** ← these registers can be indistinctively used as either REMOTE points or standard RES_FLT result registers.

For wired protocols the following lists show the kind of objects that can be mapped into REMOTE points:

Standard regions for N2_OPEN and OPTO22

```
REGION_ERROR        // Invalid (OR NULL) region
AI                  // (1..40) Analog Inputs
BI                  // (1..10) Binary Inputs
AO                  // (1..10) Analog Outputs
BO                  // (1..40) Binary Outputs
ADF                 // (1..100 xee) (101..140 ram)    Internal RES_FLT
ADI                 // (1..100 xee) (101..116 ram)    Timers
ADB                 // (1..100 xee)
```

Additional regions for OPTO22 to support slave devices

```
NX_SLAVE
```

Additional regions for Wireless remote points

```
WLS_TEMP_C
WLS_TEMP_F
WLS_REL_HUM
WLS_MODE
WLS_FAN_SPEED
WLS_KEYBOARD
WLS_SP_TEMP
WLS_SP_HUM
WLS_SP_T1
WLS_SP_PB
WLS_SP_UNOCC
WLS_BATTERY_VOLTAGE
WLS_AUX_INP
WLS_LINK_TMR
WLS_SEC_LINK_LOST
```

Modbus types supported

```
READ_COIL
READ_INPUT_STATUS
READ_INPUT_REGISTER
READ_HOLDING_REGISTER
```

BACnet types supported

```
ANALOG_VALUE
BINARY_VALUE
```

Keyword: AI_CONFIGURATION

Keyword: AI_CALIBRATION

GB	FR	ES	DE	IT	NL
AI_CONFIGURATION	EA_CONFIGURATION	EA_CONFIGURACION	AE_KONFIGURATION	IA_CONFIGURAZIONE	IA_CONFIGURATIE
AI_CALIBRATION	EA_ETALONNAGE	EA_CALIBRACION	AE_KALIBRIERUNG	IA_CALIBRAZIONE	IA_KALIBRIERING

Universal inputs can be used as either digital or analog inputs, in order to get correct readings from the analog inputs. They must be configured correctly with the AI_CONFIGURATION to set their operating mode and with AI_CALIBRATION to set their calibration value.

Also keep in mind that each universal input has an associated **hardware DIP SWITCH** that has to be set to **ON** if used with resistive type temperature sensors, and **OFF** to all other analog types.

Use the following format to set up analog inputs:

```
Syntax:     AI_CONFIGURATION AI_1 = 1 AI_CALIBRATION = 1.000
```

```
The following list gives the constant for each analog input type:
   • TYPE_12bit         0    // ADC 12-bits : 0 - 4095
   • TYPE_1kNI_C        1    // 1000 ohms Nickel °C   DIP SWITCH = ON
   • TYPE_1kSI_C        2    // 1000 ohms Silicon °C  DIP SWITCH = ON
   • TYPE_1kA99_C       3    // 1000 ohms A99 °C      DIP SWITCH = ON
   • TYPE_0_10_vdc      4    // 0-10.0 vdc    or    (0-20 ma.  @250 ohms)
   • TYPE_4_20_ma       5    // 4-20ma @ 250 ohms
   • TYPE_1kNI_F        6    // 1000 ohms Nickel °F   DIP SWITCH = ON
   • TYPE_1kSI_F        7    // 1000 ohms Silicon °F  DIP SWITCH = ON
   • TYPE_1kA99_F       8    // 1000 ohms A99 °F      DIP SWITCH = ON
   • TYPE_RATIO         9    // Ratiometric 0-100% = 10-90 FSR = 0.5-4.5 VDC
   • TYPE_TERMOCOUPLE_J 10   // Type J thermocouple, 51.7 uV/°C x191 amp
   • TYPE_TERMOCOUPLE_K 11   // Type K thermocouple, 40.6 uV/°C x191 amp
   • TYPE_ST_S3K 10K    12   // NTC Kele  0K Ohms @ 25°C Type III
   • TYPE_TC_1000_1     13   // TC current transformer NX5-SF 1000:1 ratio;
```

```
Example:    AI_CONFIGURATION AI_1 = 1 AI_CALIBRATION = 0.000
            AI_CONFIGURATION AI_4 = 1 AI_CALIBRATION = 10.000
```

For additional information regarding wiring and calculation for calibration value for the analog inputs, read the OpenBAS NX manual.

The **eZ App wizard** automatically generates preconfigured analog inputs configuration and calibration based on the chosen application.

Keyword: PLC_COUNTER

GB	FR	ES	DE	IT	NL
PLC_COUNTER	AJUSTER_INSTRUCTION_PLC	AJUSTE_INSTRUCCION_PLC	PLC_ZAEHLER_SET	PLC_CONTATORE_SET	PLC_TELLER_SET

The PLC_COUNTER keyword is used to provide for additional blank space between instructions, the compiler usually auto increments its counter when generating PLC instructions from the compiled expressions.

However if some code is needed at a specific address which could be a case for subroutines, this keyword comes handy to read just the PLC counter.

The only limitation is the plc counter given should be higher than the current PLC counter or an error would be generated due to an overlapping in memory addresses,

```
Syntax:     PLC_COUNTER [NEW PLC COUNTER ADDRESS] The range is 1-400

// Adjusts PLC counter to a new number, must be higher than current PLC
// instruction counter
Example:    PLC_COUNTER 100

// The next example shows how subroutines TWO and TWO2 are located at
// absolute PLC addresses 200 and 300 respectively

      [PROGRAM_START]
      CALL TWO
      CALL TWO2
      RES_BIT_1 = LT_GROUP BO_1 BO_2 BO_3 BO_4 BO_5 BO_6 BO_7 BO_8
      RES_BIT_2 = LT_GROUP BO_9 BO_10 BO_11 BO_12
      END   // Main program ends here

      PLC_COUNTER 200   // Change PLC counter to place TWO @ address 200
      SUB_BEGIN [TWO]   // Will add END before SUB to prevent runaway code
            BO_41 = LT_GROUP BO_20_BO_21 BO_22
            BO_42 = LT_GROUP BO_23 BO_24 BO_25
      SUB_END

      PLC_COUNTER 300   // Change PLC counter to place TWO2 @ address 300
      SUB_BEGIN [TWO2]  // Will add END before SUB to prevent runaway code
            AO_1 = 50
      SUB_END
```

Keyword: E_MAIL

GB	FR	ES	DE	IT	NL
E_MAIL	E_MAIL	E_MAIL	E_MAIL	E_MAIL	E_MAIL

Send e-mail based on a given condition, up to 4 mails can be programmed per OpenBAS-NWK-ETH3 or OpenBAS-NWK-XP Ethernet web server with the E_MAIL keyword.

```
Syntax:     E_MAIL [OPERAND_1 trigger]
            TO [MAIL ADDRESS RECEPIENT (50 chars)]
            CC [MAIL ADDRESS RECEPIENT (50 chars)]
            BCC [MAIL ADDRESS RECEPIENT (50 chars)]

Syntax:     E_MAIL E_MAIL_SUBJECT [SUBJECT STRING (100 chars)]

Syntax:     E_MAIL E_MAIL_BODY [SUBJECT STRING (190 chars)]


Example:    E_MAIL RES_BIT_1  TO rmedina@mircomgroup.com    \
                              CC rikmed@prodigy.net.mx      \
                              BCC nx5-net@rikmed.com

            E_MAIL E_MAIL_SUBJECT   This is a sample message sent by e-mail

            E_MAIL E_MAIL_BODY      Will be sent when the trigger register \
                                    is = 1, up to 160 characters can be added
```

In order to send an e-mail, besides setting the controller with an IP or it can be automatically provided, the network Ethernet switch it is connected to should have connectivity to the internet, and any firewall's limitations to access the mail server for sending mails using the SMTP on the needed port should be removed by IT administrators.

Keyword: SMS_TEXT

GB	FR	ES	DE	IT	NL
SMS_TEXT	TEXTE_SMS	SMS_TEXTO	SMS_TEXT	SMS_TESTO	SMS_TEKST

Send SMS text message based on a given condition, up to 20 SMS different text messages can be sent per each OpenBAS-NWK-SMS text message generator via SMS/GSM cellular network using the SMS_TEXT keyword.

The format to send an SMS message is as follows:

```
Syntax:     SMS_TEXT [OPERAND_1]
                 TO          [PHONE NUMBER]
                 SMS_REPEAT  [0..10]
                 SMS_RATE    [1..255 minutes]
                 SMS_TEXT    [100 char SMS message]

      Example:   SMS_TEXT RES_BIT_1 TO          +5215538213626    \
                                    SMS_REPEAT  3                 \
                                    SMS_RATE    5                 \
                                    SMS_TEXT    Alarm has been activated
```

Prior to sending SMS text messages, an OpenBAS-NWK-SMS text message generator should have a valid SIM card installed with enough balance to be able to send the messages. Also the GSM/SMS antenna should be installed to the module and coverage from the cellular company the SIM card is attached to, should be available.

Annex A, Compiler errors listing

The script compiler generates an error if any of the keywords or parameters are incorrect. Following is a comprehensive lists of the registered errors. For more information on each keyword syntax and rules refer to the detailed keyword description.

ERR_001_SCRIPT_FILES_1_TO_10:	SCRIPT NUMBER <1 or >10.
ERR_002_UKNOWN_TARGET_HARDWARE:	UKNOWN TARGET HARDWARE
ERR_003_MAX_TOKENS_PRE_PROCESS:	MAX_TOKENS ERROR (preProcessLine())
ERR_004_MAX_TOKENS_2_PRE_PROCESS:	MAX_TOKENS ERROR2 (printTokens())
ERR_005_LINE_LENGTH_EXCEEDED:	LINE LONGER THAN 250 CHARACTERS
ERR_006_TOKEN_DECODE:	TOKEN DECODE ERROR
ERR_007_READING_TOKEN_ARRAY:	ERROR READING TOKEN ARRAY
ERR_008_READING_PROCESSING_KEYWORD:	ERROR PROCESSING KEYWORD []
ERR_009_ERROR_PROCESSING_EXPRESSION:	ERROR PROCESSING EXPRESSION
ERR_010_INCORRECT_TOKEN_SEQUENCE:	INCORRECT TOKEN SEQUENCE
ERR_011_NUMBER_OF_TOKENS_EXCEEDED:	NUMBER OF TOKENS EXCEEDED
ERR_012_UNEXPECTED_END_OF_FILE:	UNEXPECTED END OF FILE FOUND IN CURRENT LINE {EOF}
ERR_013_POST_PROCESSOR_LINE_LENGTH_EXCEEDED:	POST PROCESSOR, LINE SIZE EXCEEDED 250 CHARACTERS
ERR_014_DEFINED_LABEL_IS_SAME_AS_KEYWORD:	LABEL [] IS SAME AS KEYWORD
ERR_015_DEFINED_LABEL_IS_SAME_AS_DB_OBJECT:	LABEL [] IS SAME AS DATABASE OBJECT
ERR_016_MAX_LABELS_EXCEEDED:	NUMBER OF LABELS EXCEEDED
ERR_017_UKNOWN_TOKEN:	UKNOWN TOKEN []
ERR_018_ERROR_DEFINING_NEW_LABEL:	SYNTAX TO DEFINE A NEW EQUATE LABEL IS INCORRECT
ERR_019_DATABASE_VARIABLE_INCORRECT:	DATABASE VARIABLE INCORRECT

ERR_020_CUSTOM_ID_LABEL_SAME_AS_KEYWORD:	CUSTOM ID LABEL IS SAME AS KEYWORD
ERR_021_MAX_DEFINITIONS_EXCEEDED:	NUMBER OF CUSTOM DEFINITIONS EXCEEDED
ERR_022_WRONG_SYNTAX_IF_KEYWORD:	WRONG SYNTAX IN 'IF' KEYWORD
ERR_023_WRONG_SYNTAX_JUMP_KEYWORD:	WRONG SYNTAX IN 'JUMP' KEYWORD
ERR_024_WRONG_SYNTAX_CALL_KEYWORD:	WRONG SYNTAX IN 'CALL' KEYWORD
ERR_025_INCORRECT_TOKEN_OR_OPERAND_1st_TOKEN:	INCORRECT KEYWORD OR OPERAND [] IN FIRST TOKEN IN LINE
ERR_026_LEFT_OPERAND_IS_NOT_WRITEABLE:	LEFT OPERAND IS NOT WRITEABLE (IT IS READ ONLY)
ERR_027_INCORRECT_ASSIGNMENT_OPERATOR:	INCORRECT ASSIGNMENT OPERATOR FOLLOWING DATA BASE OBJECT
ERR_028_MISSING_ASSIGNMENT_OPERATOR:	MISSING ASSIGNMENT OPERATOR FOLLOWING DATA BASE OBJECT
ERR_029_COMPLEX_MATH_NOT_SUPPORTED:	COMPLEX MATH NOT SUPPORTED
ERR_030_INCORRECT_MATH_OPERATOR:	INCORRECT MATH OPERATOR FOLLOWING OPERAND
ERR_031_MATH_EXPRESSION_EXCEEDS_OPERANDS:	MATH EXPRESSION EXCEEDS NUMBER OF OPERANDS
ERR_032_INVALID_OPERAND_FOUND_IN_EXPRESSION:	INVALID OPERAND FOUND IN EXPRESSION
ERR_033_NUMBER_OF_OPERANDS_EXCEEDED:	NUMBER OF OPERANDS EXCEEDED FOR MATH OPERATION
ERR_034_INVALID_OPERAND_IN_EXPRESSION:	INVALID OPERAND FOUND IN EXPRESSION
ERR_035_NUMBER_OPERATORS_EXCEEDED_BOOLEAN:	NUMBER OF OPERANDS EXCEEDED: BOOLEAN
ERR_036_INVERT_TAKES_ONE_OPERAND_ONLY:	INVERT INSTRUCTION TAKES ONLY ONE OPERAND
ERR_037_INVALID_OPERAND_IN_INVERT:	INVALID OPERAND FOUND IN INVERT INSTRUCTION
ERR_038_XOR_TAKES_ONLY_TWO_OPERANDS:	XOR/NXOR INSTRUCTION TAKES ONLY TWO OPERANDS
ERR_039_XOR_INVALID_OPERAND:	INVALID OPERAND FOUND IN XOR/NXOR INSTRUCTION

ERR_040_AND_OR_NEEDS_FOUR_OPERANDS:	AND_OR/AND_NOR INSTRUCTION TAKES ALWAYS FOUR OPERANDS
ERR_041_AND_OR__INVALID_OPERAND:	INVALID OPERAND FOUND IN AND_OR/AND_NOR INSTRUCTION
ERR_042_SET_RESET_NEEDS_TWO_OPERANDS:	SET/RESET INSTRUCTION TAKES ALWAYS TWO OPERANDS
ERR_043_START_STOP_EMRG_NEEDS_THREE_OPERANDS:	START/STOP[/EMRG_STOP] INSTRUCTION TAKES TWO OR THREE OPERANDS
ERR_044_START_STOP_EMRG_INVALID_OPERAND:	INVALID OPERAND FOUND IN SET/RESET OR START/STOP INSTRUCTION
ERR_045_SET_RESET_WRONG_SYNTAX:	SET/RESET WRONG SYNTAX
ERR_046_START_STOP_MISSING_EMERG_STOP:	START/STOP/EMRG_STOP MISSING EMERGENCY STOP 'EMRG_STOP' KEYWORD
ERR_047_IN_OUTSIDE_ERROR_IN_FLAG_PARAMETER:	INSIDE/OUTSIDE INSTRUCTIONS ERROR IN INVERT FLAG PARAMETER
ERR_048_IN_OUTSIDE_TAKES_3_4_OPERANDS:	INSIDE/OUTSIDE INSTRUCTIONS TAKE THREE OR FOUR OPERANDS
ERR_049_HYSTERESIS_TAKES_THREE_OPERANDS:	HYSTERESIS INSTRUCTION TAKES ALWAYS THREE OPERANDS
ERR_050_IN_OUT_HYST_WRONG_OPERAND:	INVALID OPERAND FOUND IN INSIDE/OUTSIDE/HYSTERESIS INSTRUCION
ERR_051_NUMBER_OF_SCHEDULES_EXCEEDED:	NUMBER OF SCHEDULES EXCEEDED
ERR_052_SCHEDULES_WEEK_FLAG_FORMAT_ERROR:	SCHEDULE, DAYS OF WEEK FLAG FORMAT ERROR {--------} {mtwtfssH}
ERR_053_SCHEDULES_WRONG_DATE:	SCHEDULE, WRONG DATE, DAYS MUST BE 1 TO 29,30,31 DEPENDING ONMONTH, AND MONTH 1 TO 12
ERR_054_SCHEDULES_WRONG_SYNTAX:	SCHEDULE, WRONG SYNTAX
ERR_055_UNHANDLED_KEYWORD:	UNHANDLED KEYWORD AFTER '=' OPERAND
ERR_056_RUNAWAY_EXPRESSION:	RUNAWAY EXPRESSION TRAP

ERR_057_WRONG_TIME_FORMAT_DELIMITER:	SCHEDULE, WRONG TIME FORMAT DELIMITER {hh:mm} IN 24 HOUR FORMAT {0:00} to {23:59}
ERR_058_WRONG_TIME_FORMAT:	SCHEDULE, WRONG TIME FORMAT {hh:mm} IN 24 HOUR FORMAT {0:00} to {23:59}
ERR_059_WRONG_TIME_RANGE:	SCHEDULE, WRONG TIME RANGE {hh:mm} IN 24 HOUR FORMAT {0:00} to {23:59}
ERR_060_INCORRECT_TIMER_NUMBER:	TIMER, INCORRECT TIMER NUMBER, MUST BE 1 TO 16
ERR_061_TOKEN_SHOULD_BE_FLOAT_CONSTANT:	TOKEN SHOULD BE A FLOAT CONSTANT (K_FLT)
ERR_062_STACK_FOR_CONSTANT_FLOATS_EXCEEDED:	STACK TO STORE CONSTANT FLOATS IS EXCEEDED
ERR_063_ADI_TO_STORE_TIMER_MUST_BE_1_TO_100:	ADI TO STORE TIMER MUST BE 1 TO 100
ERR_064_WRONG_SYNTAX_ASSIGNING_ADI_VALUE:	TIMER, WRONG SYNTAX ASSIGNING ADI VALUE, MUST BE 1 TO 9999
ERR_065_TIMER_WRONG_SYNTAX:	TIMER, WRONG SYNTAX OR WRONG NUMBER OF PARAMETERS
ERR_066_TIMER_NUMBER_MISMATCH:	TIMER NUMBER MISMATCH, RIGHT SIDE OF '=' DOESN´T MATCH TIMER IN RIGHT SIDE OF EXPRESSION
ERR_067_TIMER_OSC_FREQ_OUTPUT_WRONG_TYPE:	FREQUENCY CONVERTERS AND OSCILATORS CAN ONLY OUTPUT PULSES TO: RES_BIT, BINARY OUTPUTS OR TIMER_STATUS
ERR_068_TIMER_OSC_FREQ_OUTPUT_WRONG_VALUE:	TIMER, WRONG VALUE FOR OSCILATOR OR FREQUENCY CONVERTER, MUST BE 1 TO 999
ERR_069_INCORRECT_ASSGN_OPER_AFTER_THEN:	INCORRECT ASSIGNMENT OPERATOR AFTER THEN KEYWORD
ERR_070_INCORRECT_LEFT_OPER_CANT_WRITE:	INCORRECT LEFT OPERAND, HARDWARE INPUTS CAN'T BE WRITTEN
ERR_071_RUN_LOAD_TO_BE_USED_ONLY_WITH_TMRS:	RUN AND LOAD IS TO BE USED ONLY WITH TIMERS
ERR_072_ONLY_ONE_ELSE_PER_IF:	ONLY ONE ELSE KEYWORD IS ALLOWED PER IF STATEMENT

ERR_073_ONLY_ONE_THEN_PER_IF:	ONLY ONE THEN KEYWORD IS ALLOWED PER IF STATEMENT
ERR_074_PLC_CURRENT_INSTRUCTION_ERROR:	LINKER, PLC CURRENT INSTRUCTION OUTSIDE 1-400 RANGE
ERR_075_LINK_ERROR_LABEL_NOT_FOUND:	LINKER, LABEL NOT FOUND FOR JUMP OR CALL INSTRUCTION
ERR_076_LINK_JUMP_OUTSIDE_RANGE:	LINKER, JUMP EXCEEDS 200 INSTRUCTIONS OR NEGATIVE JUMP OR ABOVE INSTRUCTION 400
ERR_077_LINK_CALL_OUTSIDE_RANGE:	LINKER, CALL LIES OUTSIDE 1-400 INSTRUCTION RANGE
ERR_078_WRONG_EEPROM_INITIALIZATION_SYNTAX:	EEPROM, WRONG INITIALIZATION SYNTAX
ERR_079_WRONG_EEPROM_VALUE:	EEPROM, INITIALIZATION VALUE OUTSIDE RANGE FOR TYPE
ERR_080_PID_WRONG_SYNTAX:	PROPORTIONAL CONTROL, WRONG SYNTAX
ERR_081_PID_EEPROM_STORAGE_OUT_OF_RANGE:	PROPORTIONAL CONTROL, PARAMETERS EEPROM STORAGE OUT OF RANGE
ERR_082_WRONG_OUTPUT_REGION_SELECTED:	PROPORTIONAL CONTROL, OUPTUT MUST BE ANALOG OUTPUT OR RESULT FLOAT REGISTER
ERR_083_LIGHTING_GROUP_WRONG_SYNTAX:	LIGHTING GROUP WRONG SYNTAX
ERR_084_SUB_BEGIN_NEEDS_LABEL:	SUBROUTINE KEYWORD NEEDS AN ID LABEL SUCH AS: SUB_BEGIN [ID_LABEL]
ERR_085_PLC_COUNTER_SET_WRONG_SYNTAX:	PLC COUNTER SET WRONG SYNTAX
ERR_086_PLC_COUNTER_WRONG_VALUE:	PLC COUNTER SET WRONG VALUE, MUST BE HIGHER THAN CURRENT COUNTER
ERR_087_DEFINED_LABEL_IS_DUPLICATED:	DEFINED LABEL IS DUPLICATED, IT HAS ALREADY BEEN DEFINED BEFORE
ERR_088_FILE_REPOSITIONING_ERROR:	LINK FILE REPOSITIONING ERROR
ERR_089_ERROR_OPENING_DB_FILE:	OPENING DATA BASE FILE FOR WRITING RETURNED ERROR
ERR_090_PLC_DECODING_ERROR_MAP_FILE:	DECODING PLC INSTRUCTION FROM MAP FILE NOT POSSIBLE

ERR_091_ADX_DUPLICATED_INITIALIZATION:	CURRENT VARIABLE HAS DUPLICATE INITIALIZATION
ERR_092_PLC_DECODING_ERROR_RESULT_REGISTER:	WRONG RESULT REGISTER DECODING PLC INSTRUCTION FROM MAP FILE
ERR_093_PLC_DECODING_ERROR_OPERAND:	WRONG OPERAND WHEN DECODING PLC INSTRUCTION FROM MAP FILE
ERR_094_PLC_DECODING_INSTRUCTION_TYPE:	UKNOWN INSTRUCTION TYPE WHEN DECODING PLC INSTRUCTION FROM MAP FILE
ERR_095_LABEL_HAS_INCORRECT_DATABASE_OBJECT:	LABEL DEFINED HAS INCORRECT DATA BASE OBJECT TYPE IN MAP FILE
ERR_096_DUPLICATE_INDEXED_LABEL:	DUPLICATE INDEXED LABEL WAS FOUND IN MAP FILE
ERR_097_TOTALIZER_WRONG_SYNTAX:	TOTALIZER, WRONG SYNTAX
ERR_098_TOTALIZER_WRONG_PARAMETR_RANGES:	TOTALIZER, WRONG OPERAND RANGES
ERR_099_TOTALIZER_WRONG_INPUT_OPERAND:	TOTALIZER, WRONG INPUT OPERAND
ERR_100_ALTERNATOR_WRONG_SYNTAX:	ALTERNATOR, WRONG SYNTAX
ERR_101_ALTERNATOR_WRONG_PROCESS_VARIABLE:	ALTERNATOR, WRONG PROCESS VARIABLE
ERR_102_ALTERNATOR_WRONG_INDEX_TO_FIRST_OUTPUT:	ALTERNATOR, WRONG INDEX TO FIRST OUTPUT
ERR_103_ALTERNATOR_WRONG_NUMBER_OF_STAGES:	ALTERNATOR, WRONG NUMBER OF STAGES
ERR_104_ALTERNATOR_WRONG_EEPROM_PARAMETER_INDEX:	ALTERNATOR, WRONG EEPROM STORAGE PARAMETER
ERR_105_ALTERNATOR_WRONG_LEADER:	ALTERNATOR, WRONG LEADER OR LEADER > NUMBER OF STAGES
ERR_106_ALTERNATOR_WRONG_TIMER_SELECTED:	ALTERNATOR, WRONG TIMER SELECTED FOR NEXT STAGE OR ALARM
ERR_107_ALTERNATOR_WRONG_FEEDBACK_SELECTED:	ALTERNATOR, WRONG FEEDBACK SELECTED
ERR_108_ALTERNATOR_WRONG_EXTERNAL_ENABLE:	ALTERNATOR, WRONG EXTERNAL ENABLE SELECTED
ERR_109_ALTERNATOR_WRONG_SP_OR_PB:	ALTERNATOR, WRONG SET POINT OR PROPORTIONAL BAND VALUES

ERR_110_ALTERNATOR_TIMERS_ARE_SAME_NUMBER:	ALTERNATOR, IF TIMERS ARE ACTIVE THEY CAN´T BE THE SAME REGISTER
ERR_111_ALT_FEEDBACK_EXT_ENABLE_MISMATCH:	ALTERNATOR, MISMATCH WITH FEEDBACK AND EXTERNAL ENABLE INPUT OPERANDS
ERR_112_ALTERNATE_MISSING_SP_PB:	ALTERNATOR, MISSING SET POINT AND PROP. BAND WITH ANALOG PROCESS VALUE
ERR_113_HOUR_COUNTER_WRONG_SYNTAX:	HOUR COUNTER, WRONG SYNTAX
ERR_114_HOUR_COUNTER_WRONG_LEFT_OPERAND:	HOUR COUNTER, WRONG LEFT OPERAND, MUST BE RES_FLT
ERR_115_HOUR_COUNTER_RES_AND_PARTIAL_ARE_SAME:	HOUR COUNTER, RESULT REGISTER AND PARTIAL COUNTER CAN´T BE THE SAME REGISTER
ERR_116_HOUR_COUNTER_EEPROM_PARAMETER_INDEX:	HOUR COUNTER, WRONG EEPROM STORAGE PARAMETER
ERR_117_AHU_STAGE_WRONG_SYNTAX:	AHU STAGE, WRONG SYNTAX
ERR_118_AHU_STAGE_WRONG_PROCESS_VARIABLE:	AHU STAGE, WRONG PROCESS VARIABLE
ERR_119_AHU_STAGE_WRONG_SET_POINT:	AHU STAGE, WRONG SETPOINT
ERR_120_AHU_STAGE_WRONG_PROPORTIONAL_BAND:	AHU STAGE, WRONG PROPORTIONAL BAND (DIFFERENTIAL)
ERR_121_AHU_STAGE_WRONG_TIMER_MINIMUM_OFF:	AHU STAGE, WRONG MINIMUM OFF TIMER SELECTED
ERR_122_AHU_STAGE_WRONG_TIMER_MINIMUM_ON:	AHU STAGE, WRONG MINIMUM ON TIMER SELECTED
ERR_123_AHU_STAGE_WRONG_TIMER_INTERSTAGE:	AHU STAGE, WRONG MINIMUM INTERSTAGE TIMER SELECTED
ERR_124_AHU_STAGE_WRONG_OUTPUT:	AHU STAGE, WRONG OUTPUT OPERAND ON LEFT OF '='
ERR_125_AHU_STAGE_WRONG_EEPROM_PARAMETER_INDEX:	AHU STAGE, WRONG EEPROM STORAGE PARAMETER
ERR_126_AHU_STAGE_WRONG_PB:	AHU STAGE, WRONG SETPOINT OR PROPORTIONAL VALUES
ERR_127_AHU_STAGE_DUPLICATE_TIMERS:	AHU STAGE, WRONG DUPLICATE TIMERS SELECTED FOR ON, OFF OR INTERSTAGE TIMERS

ERR_128_AHU_STAGE_MISSING_REQUIRED_PARAMETERS: AHU STAGE, SOME OF THE REQUIRED PARAMETERS ARE MISSING

ERR_129_TREND_WRONG_DATABASE_OBJECT: TREND, WRONG DATABASE OBJECT

ERR_130_TREND_WRONG_TIME_SAMPLING_PERIOD: TREND, WRONG SAMPLING TIME PERIOD, MUST BE 1, 5, 10, 15, 20, 30 OR 60 MINUTES

ERR_131_TREND_MAXIMUM_NUMBER_OF_TRENDS_EXCEEDED: TREND, NUMBER OF GRAPHS (16) EXCEEDED

ERR_132_TREND_WRONG_SYNTAX: TREND, WRONG SYNTAX

ERR_133_TREND_HAS_INCORRECT_DATABASE_OBJECT: TREND, TRAND IN MAP FILE HAS INCORRECT DATABASE OBJECT

ERR_134_TREND_HAS_INCORRECT_INTERVAL: TREND, TREND IN MAP FILE HAS INCORRECT INTERVAL

ERR_135_SCHEDULES_MAP_FILE_WRONG_INDEX: SCHEDULES, MAP FILE WRONG INDEX

ERR_136_SCHEDULES_MAP_FILE_WRONG_SYNTAX: SCHEDULES, MAP FILE WRONG SYNTAX

ERR_137_SCHEDULES_MAP_FILE_WRONG_DESTINATION_OP: SCHEDULES, MAP FILE INCORRECT DESTINATION OPERAND

ERR_138_SCHEDULES_MAP_FILE_WRONG_DESTINATION_B: SCHEDULES, MAP FILE INCORRECT DESTINATION OPERAND FOR BINARY SCHEDULE

ERR_139_SCHEDULES_MAP_FILE_WRONG_DESTINATION_A: SCHEDULES, MAP FILE INCORRECT DESTINATION OPERAND FOR ANALOG SET VALUE SCHEDULE

ERR_140_REMOTE_POINT_WRONG_SYNTAX: REMOTE POINTS, WRONG SYNTAX

ERR_141_REMOTE_POINT_WRONG_ADDRESS_OR_NUMBER: REMOTE POINTS, WRONG ADRESS OR OBJECT NUMBER

ERR_142_REMOTE_POINT_WRONG_TYPE_OR_SYNTAX: REMOTE POINTS, WRONG TYPE OR WRONG SYNTAX

ERR_143_PROTOCOL_CONFLICT_IN_COMM_PORT: REMOTE POINTS, PROTOCOL CONFLICT IN COMM PORT

ERR_144_REMOTE_POINT_EXCEEDED_FOR_COM1: REMOTE POINTS, POINTS EXCEEDED FOR COM1

ERR_145_REMOTE_POINT_EXCEEDED_FOR_COM2: REMOTE POINTS, POINTS EXCEEDED FOR COM2

ERR_146_REMOTE_POINT_EXCEEDED_FOR_COM3: REMOTE POINTS, POINTS EXCEEDED FOR COM3

ERR_147_REMOTE_POINT_WRONG_NX_SLAVE_ADDRESS: REMOTE POINTS, WRONG SLAVE NX ADDRESS MUST BE 100..103

ERR_148_REMOTE_POINT_WRONG_WIRELESS_LINK_NR:	REMOTE POINTS, WRONG WIRELESS LINK NUMBER MUST BE 1..10
ERR_149_REMOTE_POINT_WRONG_POINT_ADDRESS:	REMOTE POINTS, WRONG ADDRESS, , FOR NX SLAVES 100..103 OTHER POINTS 1..99 or 104..253
ERR_150_REMOTE_POINT_DC_4_1st_RSRVD_NX_SLVS:	REMOTE POINTS, DUAL CORE REMOTES 51..54 RESERVED FOR NX SLAVE DEVICES
ERR_151_KEYWORD_NOT_SUPPORTED_ON_THIS_VERSION:	THIS KEYWORD IS NOT YET SUPPORTED ON THIS VERSION, WILL BE IMPLEMENTED IN A FUTURE REVISION
ERR_152_WIRELESS_LINK_WRONG_SYNTAX:	WIRELESS LINK, WRONG SYNTAX
ERR_153_WIRELESS_LINK_WRONG_INDEX:	WIRELESS LINK, WRONG LINK INDEX, MUST BE 1 to 10
ERR_154_WIRELESS_LINK_WRONG_GROUP:	WIRELESS LINK, WRONG GROUP NUMBER, MUST BE 1 to 10
ERR_155_WIRELESS_LINK_WRONG_ADDRESS:	WIRELESS LINK, WRONG ADDRESS, MUST BE 1 to 199
ERR_156_PROTOCOL_NOT_SUPPORTED_ON_SELECTED_PORT:	COMM PORT, PROTOCOL NOT SUPPORTED ON SELECTED PORT
ERR_157_ADDRESS_OUT_OF_RANGE_FOR_SEL_PROTOCOL:	COMM PORT, ADDRESS OUT OF RANGE FOR SELECTED PROTOCOL
ERR_158_WRONG_BAUD_RATE_SELECTED:	COMM PORT, WRONG BAUD RATE SPECIFIED
ERR_159_COMM_PORT_WRONG_SYNTAX:	COMM PORT, WRONG SYNTAX
ERR_160_COMM_PORT_SET_AS_SLAVE_NEEDS_ADDRESS:	COMM PORT, PORT SET AS SLAVE NEEDS AN ADDRESS DEFINED
ERR_161_N2_OPEN_WRONG_SETUP:	COMM PORT, N2-OPEN WRONG PARAMETERS SETUP
ERR_162_COMM_PORT_DEFINED_MORE_THAN_ONCE:	COMM PORT, DEFINED MORE THAN ONCE
ERR_163_MODBUS_SLAVE_WRONG_ADDRESS:	COMM PORT, MODBUS RTU WRONG SLAVE ADDRESS SELECTED
ERR_164_BACNET_WRONG_ADDRESS:	COMM PORT, BACNET MSTP WRONG SLAVE ADDRESS SELECTED
ERR_165_WRONG_COMM_PORT_SELECTED:	COMM PORT, WRONG NUMBER SELECTED, MUST BE 1, 2 OR 3

ERR_166_AI_CONFIG_WRONG_AI_SELECTED:	AI CONFIGURATION, WRONG ANALOG INPUT SELECTED
ERR_167_AI_CONFIG_WRONG_SYNTAX:	AI CONFIGURATION, WRONG SYNTAX
ERR_168_AI_CONFIG_WRONG_TYPE:	AI CONFIGURATION, WRONG TYPE SELECTED
ERR_169_AI_CONFIG_WRONG_CALIBRATION_VALUE:	AI CONFIGURATION, WRONG CALIBRATION VALUE
ERR_170_TREND_DIGITAL_DATABASE_POINT_WRONG_SYX:	TREND, DIGITAL DATABASE POINT WRONG SYNTAX

CANADA - Main Office
25 Interchange Way
Vaughan, ON L4K 5W3
Tel: (888) 660-4655
(905) 660-4655
Fax: (905) 660-4113

U.S.A
4575 Witmer Industrial Estates
Niagara Falls, NY 14305
Tel: (888) 660-4655
(905) 660-4655
Fax: (905) 660-4113

TECHNICAL SUPPORT
North America
Tel: (888) Mircom5
(888) 647-2665
International
Tel: (905) 647-2665

Printed in Canada
Subject to change without prior notice

www.mircomgroup.com