



**CREATOR**<sup>®</sup>  
Professional Audio-Visual Manufacturer

Think Control 1.0  
Help Document  
V1.0

CREATOR CHINA

# Preface

The help file for Think Control 1.0 mainly introduces the operation on the interface, language rule and code organization of Think Control 1.0 software.

The Manual serves as user's operation instruction, rather than for maintenance service purpose. Since the date of release, any function or relevant parameter alteration will be in supplement instruction. Please refer to the manufacturer or dealers for inquiry.

CREATOR Electronics own the copyright of the Manual. Without permission, any unit or person shall not take part or total of the Manual for business purpose.

The copyright of the Manual is protected by Copyright Law of People's Republic of China and other Intellectual Property Law. Without written permission, any copy or distribution is prohibited.

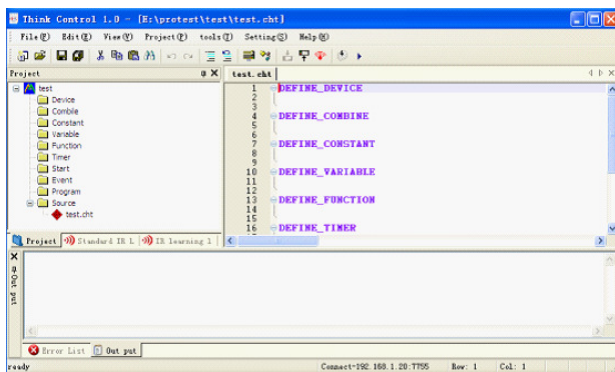
# Index

Chapter 1 Overview.....	1
1.1 About Think Control1.0 .....	1
1.2 Operation on the interface .....	1
1.2.1 How to create a new project? .....	1
1.2.2 How to open an existing project?.....	1
1.2.3 How modify project property? .....	2
1.2.4 How to modify items?.....	2
1.2.5 How to use edit function?.....	3
1.2.6 How to add device and event? .....	4
1.2.7 How to use infrared database?.....	6
1.2.8 How to compile project?.....	6
1.2.9 How to define and compile a template? .....	7
1.2.10 How to compile a project of cascading connection? .....	8
1.2.11 Running and debugging .....	9
1.2.12 Regular expression .....	10
Chapter 2 CREATOR Language rule .....	12
2.1 Basic grammar rules .....	12
2.1.1 Type and value.....	12
2.1.2 System internal object.....	13
2.1.3 Array.....	14
2.1.4 Operator .....	14
2.1.5 Assignment statement .....	16
2.1.6 Flow control statement.....	16
2.1.7 WAIT statement.....	20
2.1.8 Function.....	21
2.2 Code Organization .....	23
2.2.1 Composition of code organization .....	23
2.2.2 Control device function .....	28
2.2.3 Other functions.....	32
2.2.4 Sample program.....	39

# Chapter 1 Overview

## 1.1 About Think Control1.0

Think Control 1.0 is programming software developed for CREATOR programmable central mainframe. The so-called programmable refers to programming on the mainframe with the use of Think Control 1.0, so as to realize a variety of control logics. Think Control 1.0 software mainly comprises of editing module, compiling module, infrared code management module and download module etc.




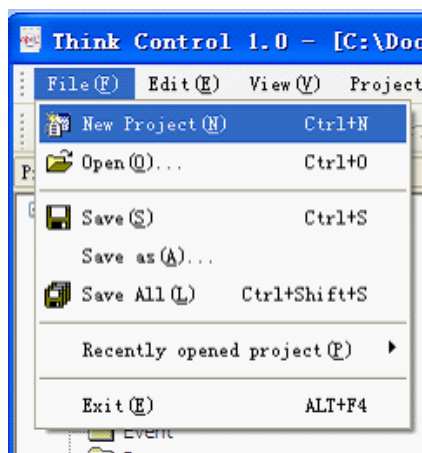
Main window of the program

## 1.2 Operation on the interface

### 1.2.1 How to create a new project?

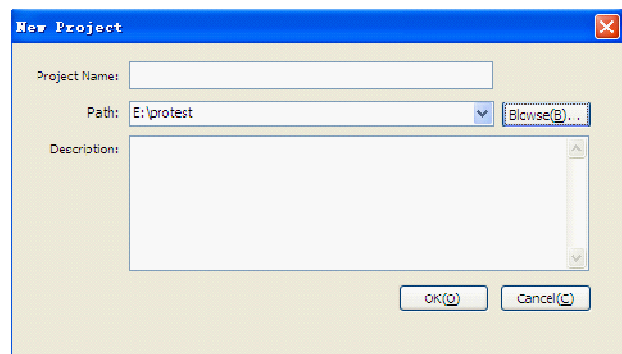
Step 1:

Select “file”-“new” on menu or tool bar button  ;



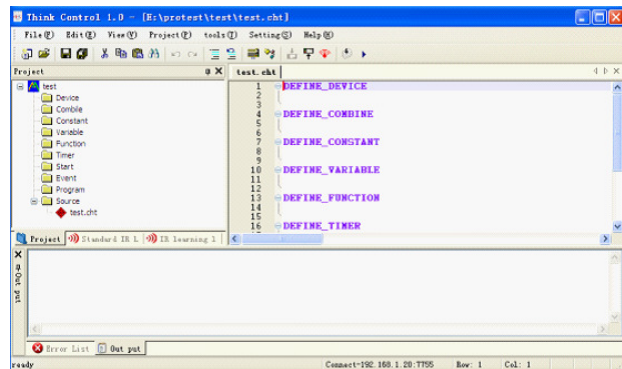
Step 2:

In the pop-up dialog box of “new project,”select “new project,”input new project name like “test”; and allocate a directory for keeping all files related with the project. Here it is “E:\protect.”You may fill in the description of the project into the box below. Also, you may modify the message through “project” ”project property”



Step 3:

Click on “confirm” button to establish the new project;




### 1.2.2 How to open an existing project?

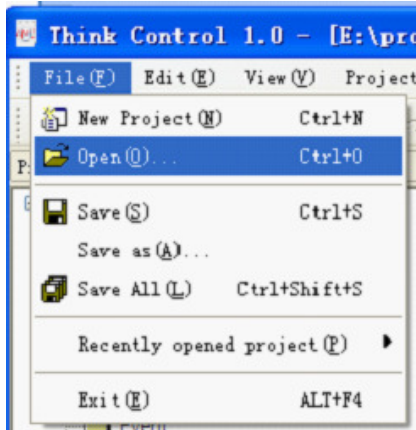
There are 2 methods:

1、 If the file has been associated with, you may double-click on the file to open the project.

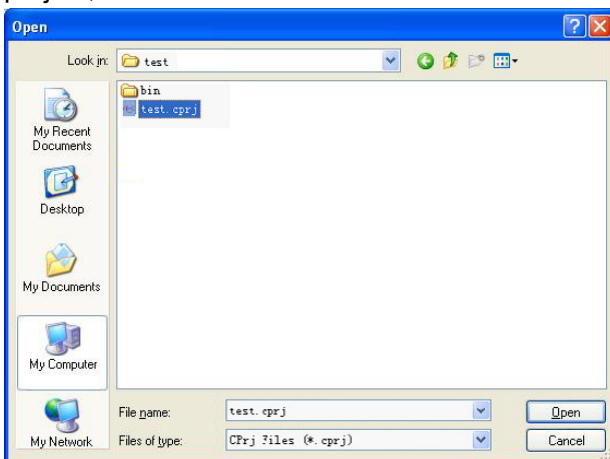
2、 In the 2<sup>nd</sup> method, please take the following steps:

Step1:

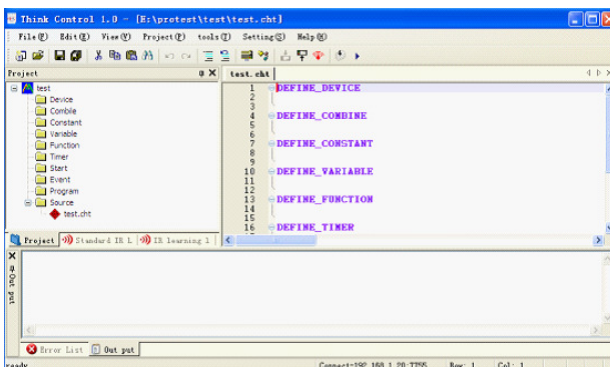
Use “file” on menu “open” or tool bar button ;



Step2: Select the project file to open from the pop-up dialog box, press “open” to open the project;

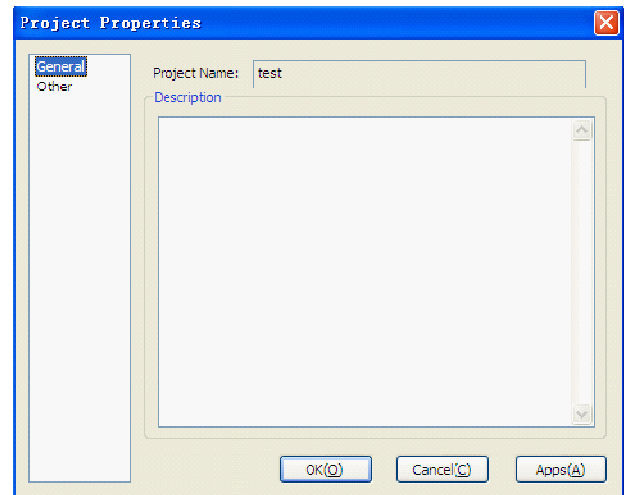


After opening the project, the main interface is as below

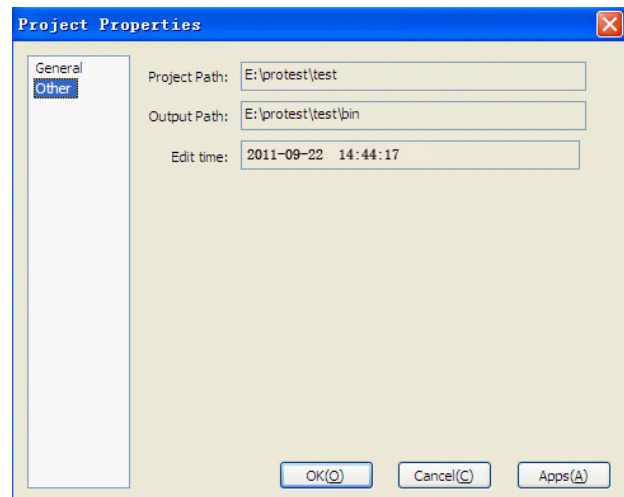


### 1.2.3 How modify project property?

If you'd like to change the project property, please open a project firstly. Select “file” on menu “project property” to open “project property” dialog box to have configuration. “file name” can be seen in “routine” items, but cannot be modified, while “description” is available for modification.



“Project path ,” “output path” and “modification time” can be seen in “others” item

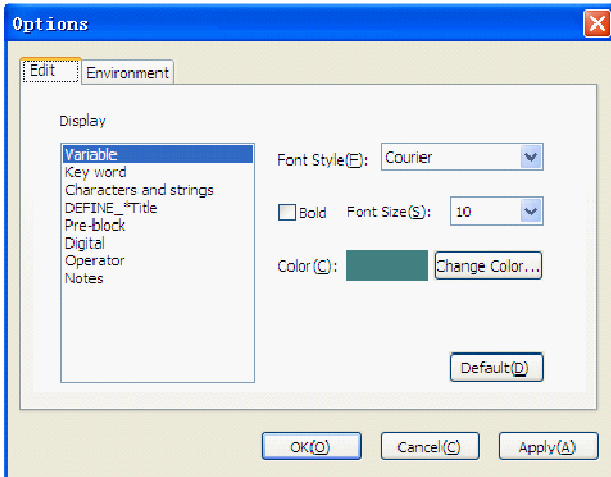


### 1.2.4 How to modify items?

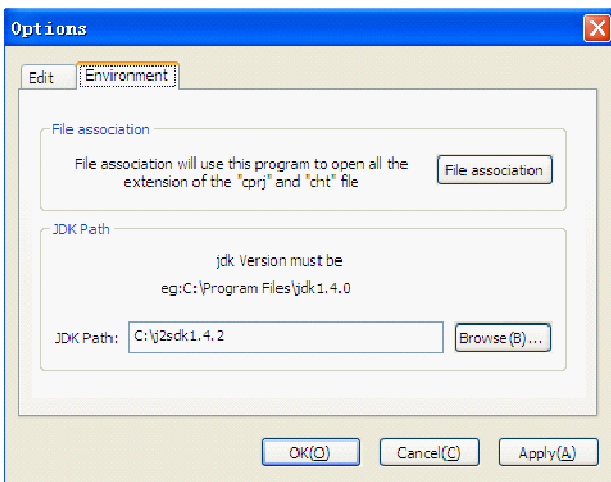
Modify program configuration message  
Select “Setting” on menu→ “Options”



Dialog box pops up. Select “edit” item, you may configure all displaying formats of code;



In “environment” item, you may associate “cprj” with “cht,” then you can directly open the file by double-clicking on a certain project file. Besides, you may know the path of JDK. And JDK must be version 1.4. Default JDK path is as the JAVA\_HOME environmental variable value



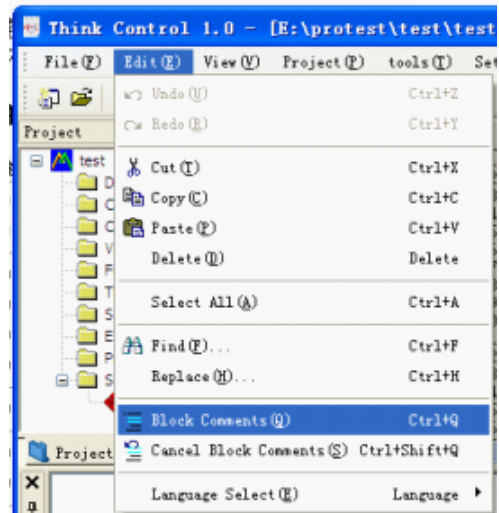
### 1.2.5 How to use edit function?

Edit function includes: text input, cancel, redo, block comment, cancel block comment, cut, paste,

select all, search and replace. Edit function cannot be used before file is opened. Most functions are similar in use with those of common text editor.

1. “block comment,” “cancel block comment” function can be found respectively in edit menu,

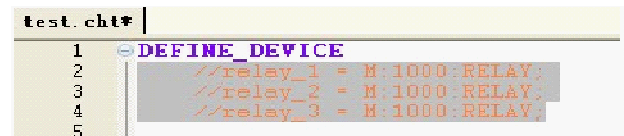
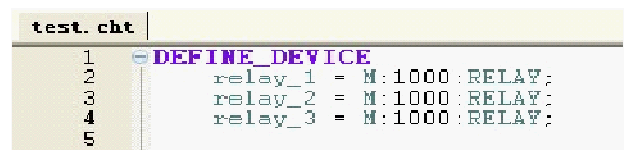
or you may use tool bar button



#### 1.1 Code block annotation

Firstly, select the code to note, and click on “edit” on menu à “block annotation” or directly click

on tool bar button



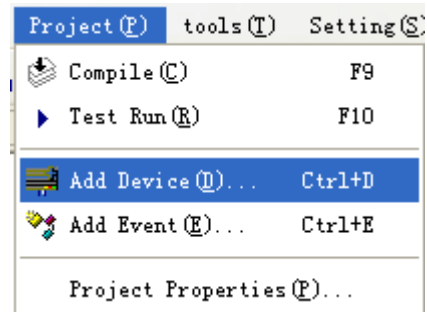
#### 1.2 Cancel code block annotation

Firstly, select the code for cancelling annotation, and then click on “edit” on menu à “cancel block annotation,” or directly click on tool

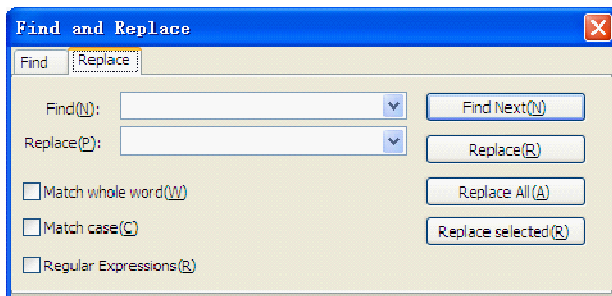
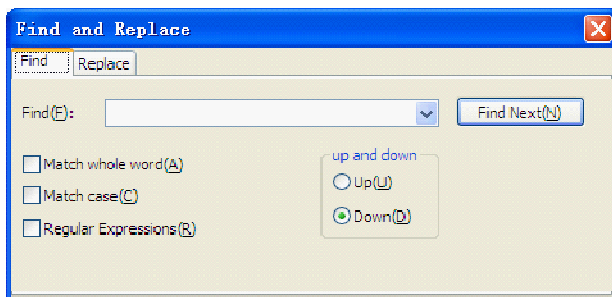
bar

```
test.cht#
1 DEFINE_DEVICE
2 //relay_1 = M:1000:RELAY;
3 //relay_2 = M:1000:RELAY;
4 //relay_3 = M:1000:RELAY;
5
```

```
test.cht
1 DEFINE_DEVICE
2 relay_1 = M:1000:RELAY;
3 relay_2 = M:1000:RELAY;
4 relay_3 = M:1000:RELAY;
5
```



## 2. Find and replace




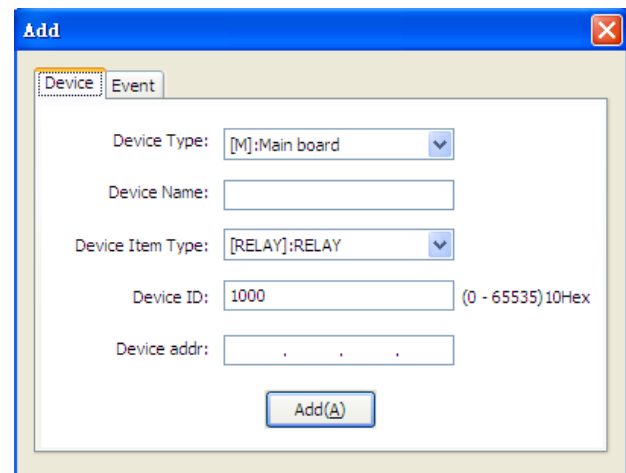
There are many ways of finding, just like the function of common editor. If select regular expression to find, the matched content will be displayed. Please refer to [11. Regular expression.](#)

### 1.2.6 How to add device and event?

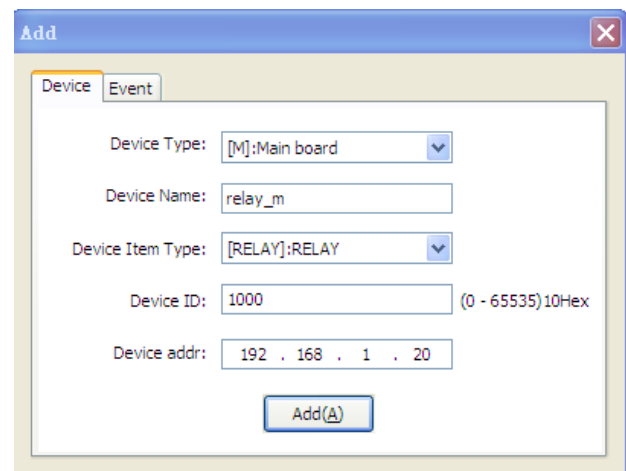
User may directly compile code to add device, interlock and event. At the meantime, user may add it by using “add device ,” “add event” on the

menu, or tool  bareasily.

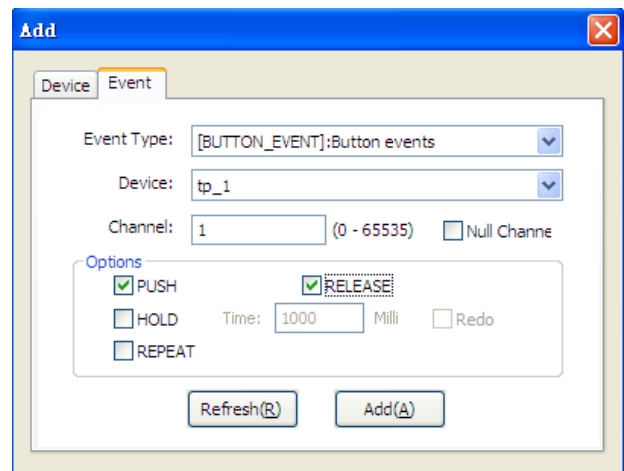
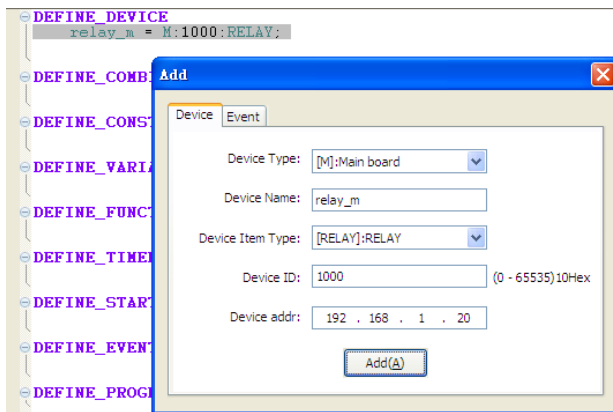
1. “Project”à “Add device” or tool bar button  to open device dialog box:




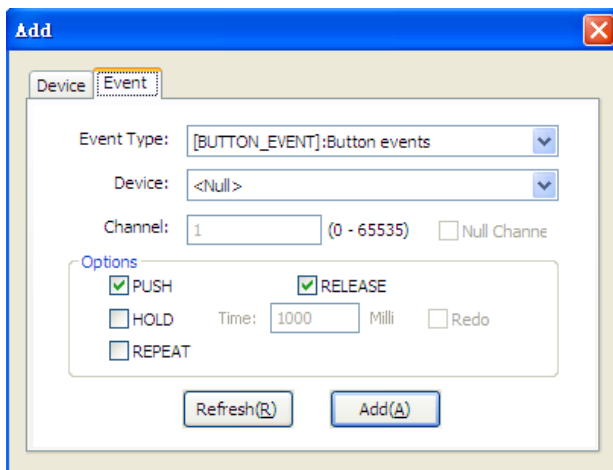
Input the desired device name like “relay\_M”; select device type like “[M]: Mainframe board type; input device ID like “1000 ,”select device element type like [RELAY]: relay type; input device IP address like 192.168.1.20”.



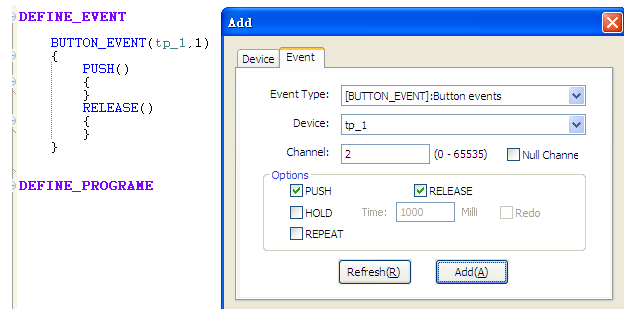
Finally, click on “add” button to finish the process.



2. “Project”à “add event” or tool bar button  to open adding event dialog box:



Finally, click on “add” button to finish adding button event.



### 3.2 Add level event, select [LEVEL\_EVENT].

In “device” pull-down list, select a device or you may select “<void>”.

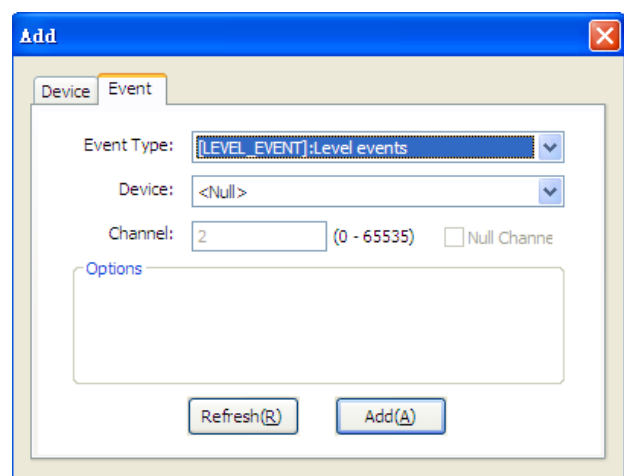
In “channel number ,” input the channel number of target button, if it’s not needed, please tick “without channel number”.

### 3.1 Add button event

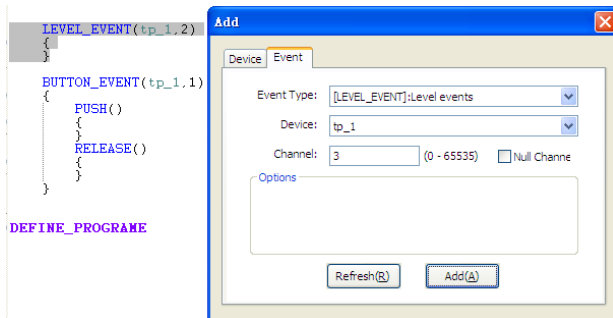
In “Event type” select [BUTTON\_EVENT]  
In “Device” pull-down list, select a device, like “tp\_1 ,” or you may select “<void>”.

In “Channel number ,” input the channel number of your target button. If the number is not needed, please tick “Without channel number”. The channel number of touch screen device is its join number value.

In “Options” select the desired event function. With regard to HOLD function, you may configure the interval and repeat times.



Finally, click “add” button to finish adding level event.



### 3.3 Add data event

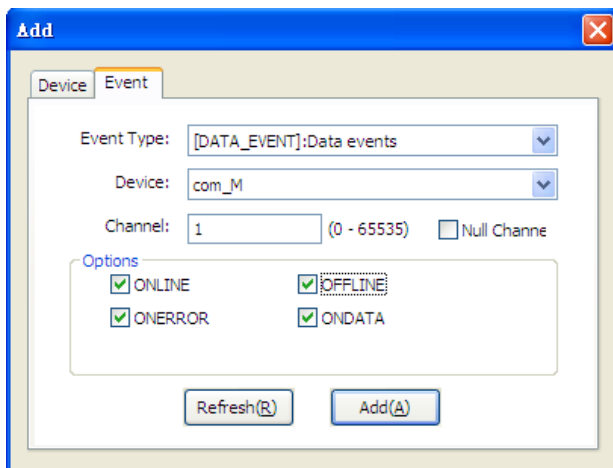
In “event type ,”select [DATA\_EVENT].

In “Device” pull-down list, select a device or select “<void>”.

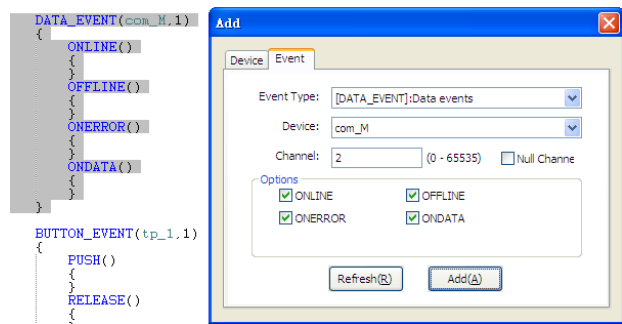
In “channel number ,”you may select channel number. For devices with serial port, the channel allocates serial port number, like the com\_M defined on main board.

Here, you select channel number 1, which means the 1<sup>st</sup> serial port on main board. The rest are likewise.

In “options ,”select your desired event function.



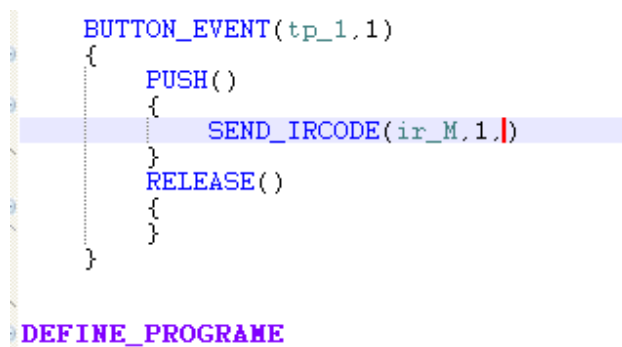
Finally, click on “add” button to finish adding button event.



### 1.2.7 How to use infrared database?

You may directly select corresponding infrared code from infrared database, and insert it into project file, without writing it on your own

1. Firstly, move the cursor onto the position for inserting infrared code;




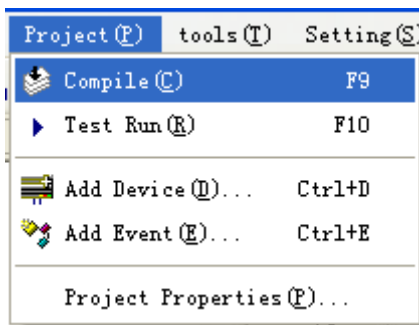
2. In infrared database, select your desired infrared code, and double-click it to insert



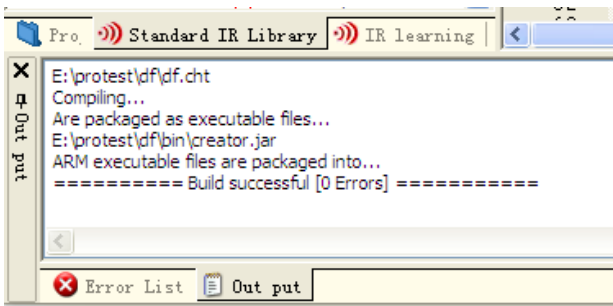
### 1.2.8 How to compile project?

Use “project” and “compile” menu or tool bar

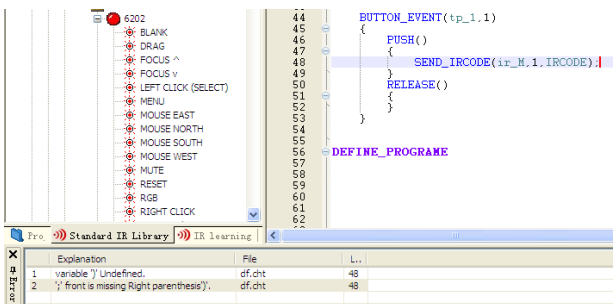
button  to execute compiling operation;



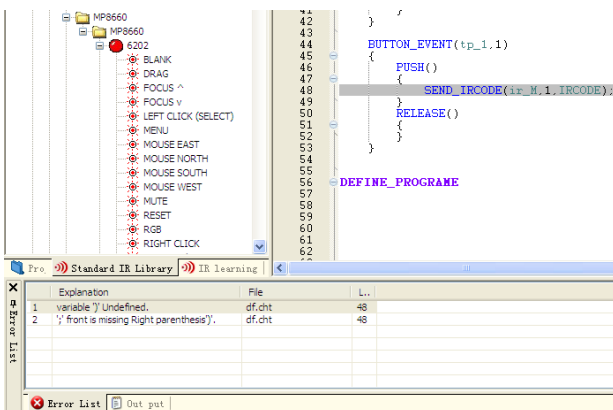
When compiling, you may acquire the detailed information from “output” window;



If with error when compiling, after the process, it will auto-jump to “error” window to show the errors encountered;



Double-click the 1<sup>st</sup> error message to jump to the error code;



### 1.2.9 How to define and compile a template?

Template definition format

Template (template parameters list)

```
{
  BUTTON_EVENT list
  LEVEL_EVENT list
  DATA_EVENT list
}
```

parameter types

T:TP T:TP:C M:COM M:COM:C M:RELAY

M:RELAY:C M:IO M:IO:C M:IR M:IR:C

N:TP N:TP:C N:LITE N:LITE:C N:RELAY

N:RELAY:C N:VOL N:VOL:C N:WM

N:WM:C

L:LITE L:LITE:C L:RELAY L:RELAY:C

L:VOL L:VOL:C L:WM L:WM:C L:DMX512

L:DMX512:C L:ACAR L:ACAR:C

Int string boolean byte char

Template parameters instruction:

- T:TP** Touch screen device type
- T:TP:C** Touch screen device channel (Join Number0 type)
- M:COM** Serial port on main board
- M:COM:C** Serial port type on main board
- M: RELAY** Relay type on main board
- M: RELAY:C** Relay channel type on main board
- M:IO** IO port type on main board
- M:IO:C** IO port channel type on main board
- M:IR** Infrared port type on main board
- M:IR:C** Infrared port channel type on main board

...

As to others, N is for CRNET board; L is for CAN board; WM is for on-wall panel; VOL is for volume controller; ACAR is for analog card.

Note: Those with C are actually the aliases of int type, equivalent to typedef \*\*:C int of C.

#### BUTTON\_EVENT list

As those defined in common projects,

`BUTTON_EVENT(tp1,j1)` // tp1 and j1 refer to the parameters in parameters list.

```
{
    PUSH(){}
    RELEASE(){}

    REPEAT(){}
    HOLD(){}
}
```

#### LEVEL-EVENT list

As those defined in common project, like

`LEVEL_EVENT(tp1,j1)`

```
{
}
```

#### DATA\_EVENT list

As those defined in common project, like

`DATA_EVENT (com,c2)`

```
{
    ONLINE(){}
    OFFLINE(){}
    ONERROR(){}
    ONDATA(){}
}
```

Example:

1. To create a new project file, delete all `DEFINE_*` labels;

2. Add `DEFINE_TEMPLATE` label;

3. Compile template code, as following example:

```
DEFINE_TEMPLATE
Template abc(
```

```
T:TP tp1 ,T:TP:C J1 ,T:TP:C J2 ,
M:COM com ,M:COM:C C1 ,int kkk
```

```
)
{
    BUTTON_EVENT(tp1,J1)
    {
        PUSH()
        {
            TRACE("sfsdfsfsafsaf");
            SEND_COM(com,1,"123456789");
            //ON_RELAY(relay, mr1);
//SEND_IRCODE(mir,ir1,IRCODE<"StanderIRDb
:3M:CODEC:VCS3000:POLYCOM1:6289:2
(abc)">);

        }
    }
}
```

Example of calling template:

1. To create a project;
2. Copy the xxx.jar which compiled in template project to the directory of new project (In the above project, abc.jar file will be created in bin directory.)
3. Under `DEFINE_CALL_TEMPLATE` label, compile and call abc.jar code (note to the correspondence between the calling parameter and its definition);

**Eg:**

```
DEFINE_DEVICE
qq = T:10:TP;
COM = M:1000:COM;
```

```
DEFINE_CALL_TEMPLATE
abc(qq,3,1,COM,1,5);
```

### 1.2.10 How to compile a project of cascading connection?

With function of cascading connection, you may use a mainframe (A) to control another mainframe (B), by sending touch screen data via network.

1. The function for sending touch screen data:

```
SEND_M2M_JNPUSH("192.168.1.2",12);//
press when sending touch screen
joinmuber12 to mainframe 192.168.1.2
```

```
SEND_M2M_JNRELEASE("192.168.1.2",12)
;// release when sending touch screen to
mainframe 192.168.1.2
```

```
SEND_M2M_LEVEL("192.168.1.2",12,13); //
the joinmuber 12 level data 13 when sending
touch screen to mainframe 192.168.1.2
```

Note: When sending touch screen data, the mainframe (B) to be controlled needs to re-define device module. Definition ID is 0; main type is M; IP I 0.0.0.0 mainframe touch screen device (example: mtp=M:0:TP:0.0.0.0;)

Also, in DEFINE\_COMBINE, it is needed to set the device equal to other touch screen device before under control.

2. Sending data between mainframes

The function of sending data between two mainframes is

```
SEND_M2M_DATA("192.168.1.2","123456"),
which means sending characters 123456 to
mainframe 192.168.1.2.
```

3. Receiving data between mainframes

It is received through M2MDATA\_EVENT

Example:


```
M2MDATA_EVENT() // receive the data sent to
the mainframe from all the other mainframes.
```

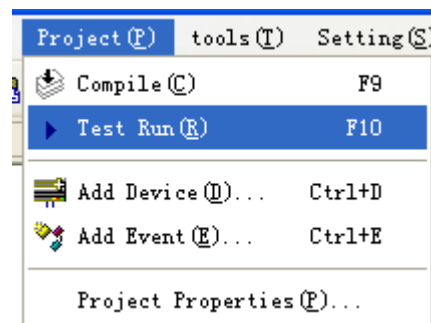
```
{
    ONDATA()
    {
        TRACE("all
ip:"+DATA.STR_M2MIPADDR +" data:"
+DATA.DataString);
    }
}
```

```
M2MDATA_EVENT("192.168.1.15") //
receive the data from mainframe 192.168.1.15
only.
```

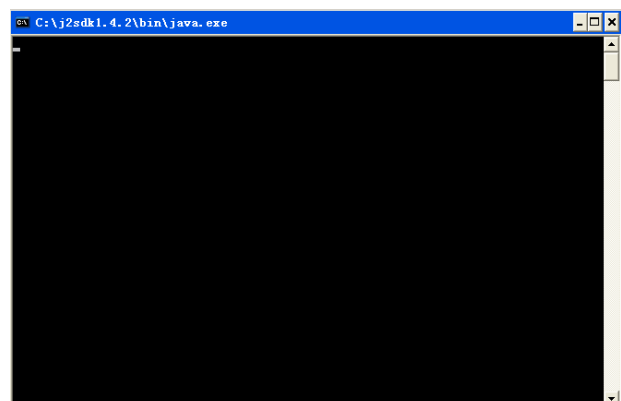
```
{
    ONDATA()
    {
        TRACE("dai ip
ip:"+DATA.STR_M2MIPADDR +" data:"
+DATA.DataString);
        int p = DATA.B1;
        string ip =
DATA.STR_M2MIPADDR;
    }
}
```

### 1.2.11 Running and debugging

If there is no error in compilation, select “option” à “local trial running” or tool bar button  to run the compiled program on local machine.



After running the program, a window with black background will pop up as follows:



After running the program, device simulator

can be used to simulate the events of pressing button and moving draw bar and so on.

By simulating these device events, program can be tested. After passing the test, the program can be available for download for running on real environment

### 1.2.12 Regular expression

Regular expression is actually a pattern string, using a special string to describe a specific mode, and to match another target string to see if it is with such mode. For example, pattern string “ab+” represents for the strings start with “a” and

“b”. Therefore, target strings of “ab”, “abb”, “abbbbb” are all meeting such mode

#### 1. Rule of regular expression

##### 1.1 Ordinary characters

Alphabet, numbers, Chinese characters, underline and those punctuations with special meaning, but without appearing on the following section are all “common characters”. When matching a target pattern string, the ordinary characters in pattern string will match the same character. As the above example, “ab+” in pattern string has ordinary characters of “a” and “b”. And “a” appears in the first position, which means the target character string has to be starting with “a”. In such way, it’s likely to be successfully matched.

##### 1.2 Simple escape characters

To show some characters inconvenient for writing such as tab key and enter etc., it adopts the way of putting “\” before some characters with special meaning.

Pattern string	For matching
\n	Line break
\r	Enter
\t	Tab key

\\	character\'itself
\^	character'^itself
\\$	character'&itself
\\.	character'.itself

And there are some punctuation marks for special use appearing in the next chapters. After putting “\” ahead, it represents the symbol itself. For example: + and \* are both with special meaning. If you’d like to match the “+” and “\*” in the target string, pattern string should be correspondingly changed as “\+” and “\\*”.

#### 1.3 Pattern string for matching a certain character set ( a certain type of characters)

In regular expression, some expressions can match any character in a certain characters set. For example: patter string “\d” can match any number (any one from 0 to 9). Despite it can match any character, but just for one, not for multiple.

Pattern String	For matching
\d	Any number ranging from 0 to 9
\w	Any character, number or underline, that’s any one from A~Z,a~z,0~9,_,.
\s	Any one from space, tab key and form feed character.
.	Decimal point can match any character but line break(\n).

#### 1.4 Self-define the pattern string for matching characters set.

Pattern string is a series of characters in [], which means matching any character in the square brackets. Using “^” in the square brackets means “except” the content, and matching any character but such character. Likewise, although it can match anyone of them, but for one not for multiple.

Pattern string	For matching
[ab5@]	Match "a" or "b" or "5" or "@"
[^abc]	Match any character except "a", "b", "c"
[f-k]	Match any character within "f"~"k"
[^A-F0-3]	Match any character except "A"~"F", "0"~"3"

### 1.5 Special symbols modified for times of matching

If add special symbols modified for times of matching to the patter string, it is no need to repeatedly write patter string to have matching

repeatedly.

Method: Put "times modification symbol" behind "the patter string to be modified". For example: patter string "[bcd] [bcd]" can be re-written as "[bcd]{2}".

Pattern string	Function
{n}	Repeat pattern string for n times. For example: "\w{2}" equals to "\w\w"; "a{5}" equals to "aaaaa"
{m,n}	Repeat pattern string at least for m times, and n times at maximum. For example: "ba{1,3}" can match "ba" or "baa" or "baaa"
{m,}	Repeat pattern string at least for m times. For example: "\w{d{2,}" can match "a12", "_456", "M12344"...
?	Match pattern string for 0 or 1 time, equaling to {0,1}. For example: "a[cd]?" can match "a", "ac", "ad"
+	Pattern string at least appears once, equaling to {1,}. For example: "a+b" can match

	"ab", "aab", "aaab"...
*	Pattern string doesn't appear or appears for any times, equaling to {0,}. For example: "\^*b" can match "b", "^^^b"...

### 1.6 Other symbols with special meanings.

Some symbols in pattern string represent special abstract meaning:

Symbol	Function
^	Match the target character string from the beginning, without matching any specific character.

\$	Match the end of target character string, without matching any specific character.
\b	Match a word border, where is the position between the word and space, without matching any specific character.

### 2. Some symbols can influence the relationship within the sub-pattern strings.

Symbol	Function
	"or" relation between the left and right pattern strings, use left or right pattern string to have the matching.
()	(1). When modifying the matching times, the pattern string in brackets can be modified as a whole. (2). To acquire matching result, the contents matching the pattern string in brackets can be separately acquired.

## Chapter 2 CREATOR Language rule

### 2.1 Basic grammar rules

#### 2.1.1 Type and value

##### 1. Basic data type

Type names	Key word	Occupied space(byte)	Value range	Default
Byte type	byte	1	-128—127	0
Integer type	int	4	-2147483648—2147483647	0
Double precision floating point	double	8	-1.79769313E+308—1.79769313E+308	0.0
Character type	char	2	0—65535	0
Boolean type	boolean		True or false	false
Character string	string			""
Void type	void			

##### 2. Identifiers

Any variable, constant, function, array and device are with a name-- that's called identifier. Identifier can be freely appointed by programmer, but it has to be in accordance with a certain grammatical rules. The system has following stipulations for the definition of identifier.

(1) Character string composed by letter, number and underline, and starts with letter or underline.

(2)Reserved characters cannot be identifiers, including key word and system object name etc.

(3) Identifiers are case sensitive.

(4) The name of WAIT sentence must be in accordance with the requirement, i.e. the name of WAIT sentence is also an identifier.

An example of a legal identifier:

```
dog // start with letter and composed of letters
dog //start with underline and composed of
letters
car2 // start with letter and composed of letters
and numbers
_2_wheel_ // start with underline and
composed of letters, underline and numbers
```

An example of illegal identifier:

```
2dog // start with number is not allowed
do*g // the character--"*" doesn't belong to
number, letter or underline.
do g // the space--" " doesn't belong to number,
letter or underline.
while // system keyword cannot be an identifier.
BUTTON // System object cannot be an
identifier.
```

**Note:** As identifier is case sensitive, “dog” and “Dog” are regarded as two different identifiers.

### 2.1.2 System internal object

Instruction: Internal object is especially made for button event, drawing bar event and data event, indicating when triggering the event, internal object show details of the device that triggers event, especially when there is no channel in triggering event, internal object can be used for checking channel number.

eg:

```
BUTTON_EVENT(tp_1)
{
    PUSH()
    {
        switch(BUTTON.Channel)
        {
            case 1://press the 1st button
                //operation executed
                break;
            case 2://press the 2nd button
                // operation executed
```

```
break;
    }
}
}
```

#### 1. BUTTON

Property name	Type	Instruction
DeviceType	char	Device type, such as: 'M', 'S' etc.
DeviceID	Int	Device ID
Module	int	Device element module, such as: 'TP', 'RELAY'
Channel	int	Element identifier
JoinNumber	int	Channel number of triggering button

Note: Here Channel equals to JoinNumber

#### 2. LEVEL

Property name	Type	Instruction
DeviceType	char	Device type, such as: 'M', 'S' etc.
DeviceID	Int	Device ID
Module	int	Device element module, such as: 'TP', 'RELAY'
Channel	int	Element identifier
JoinNumber	int	Channel number of LEVEL button
Value	int	Value of LEVEL(analog)

eg: Use drawing bar to set up the volume in 1<sup>st</sup> channel

```
LEVEL_EVENT(tp_1,2)
{
    SET_VOLTOTOL(vol,1,LEVEL.Value);
}
```

#### 3. DATA

Property name	Type	Instruction
Device Type	Char	Device type, such

		as: 'M', 'S'等
DeviceID	int	Device ID
Module	int	Device element module, such as: 'TP', 'RELAY'
Channel	int	Element identifier
Data	byte[ ]	Data carried by DATA event

### 2.1.3 Array

Array is a combination of a type of data (e.g. integer data: int, or character data: char)

#### 1. Definition of array

Similar with variable, to define an array is necessary to define data type and the size of array, i.e. number of data stored in the array.

The format of defining array:

**Data\_type array\_name[size of array];**

**Data types** can only be one of the following types "int", "char", "double", "byte", "boolean", "string". **Array name** shares the naming rule as variable. For details, please refer to the naming rule of identifier in [Type and value](#).

**Array size** can only be integer number, such as 10 and 100 etc.

Note: The array here is dynamic, with size allocated by definition, which can be changed when processing. In other words, it is a pointer variable, pointing to different address in different process.

The following is to define an array with size of 10 integers:

```
DEFINE_VARIABLE
// define an integer array a with 10 elements
int a[10];
```

Visiting array

The format of visiting array:

**Array name [subscript]**

Subscript can be integer constant or integer variable, referring to the data sequence in the

array to be visited. Subscript ranges from 0 to size of array -1, i.e. the subscript of 1st element is 0; the subscript of 2nd element is 1...the rest are likewise. In the above array defined "int a[10];", the subscript ranges from 0 to 9.

Visit the above defined number:

```
DEFINE_START
```

```
a[0] = 9; // Assign 9 to the 1st element in the array
```

```
a[i] = 5; // Assign 5 to the i+1 element in the array (Assuming the integer variable "i" has been defined. Note: The value of "i" should be ensured within the legal range between 0 and 9.)
```

```
a[5+i] = a[0] + a[i]; // Assign the total value of the 1st element and the i+1th element to the i+6 element.
```

### 2.1.4 Operator

#### 1.1 Arithmetic operator

Arithmetic operator is especially for the operation of value type. The system-provided arithmetic operators are in table 1-1.

Table 1-1 Arithmetic operators

Symbol	Operation	Example	Function
	Plus sign	a + b	For the summation of a and b.
-	Minus sign	a - b	For the value of a subtracts b
*	Multiplication sign	a * b	For the value of a multiples b
/	Division sign	a / b	For the value of a divides b
%	Modulus	a % b	For the remainder of a divides b

Note: (1) Only integer(int) data can have modulus.

(2) When data from two integer types are taking division operation, the resulting integer part is taken, while the decimal part is eliminated.

#### 1.2. Relational operators

Relational operation is for comparing the value of two data. The relational operators are included in the table 1-2

Table 1-2 Relational operators

Symbol	Operation	Example	Function
==	Equal	a == b	Judge if a is equal to b
!=	Not equal	a != b	Judge if a is unequal to b
>	Larger	a > b	Judge if a is larger than b
<	Smaller	a < b	Judge if a is smaller than b
>=	Larger or equal	a >= b	Judge if a is larger or equal to b
<=	Smaller or equal	a <= b	Judge if a is smaller or equal to b

### 1.3、 Logical operators

Logical operation is especially for Boolean data. The result is still Boolean type. The logical operators are included in table 1-3.

Table 1-3 Logical operators

Symbol	Operation	Example	Operation rule
&	Bitwise logical "and"	a & b	True when both a and b are true
	Bitwise logical "or"	a   b	False when a or b is false.
!	Logical "non"	!x	False when x is true; True when x is false.
&&	Logical "and"	a && b	True only when both a and b are true.
	Logical "or"	a    b	False only when both a and b are false.
^^	Logical different	a ^^ b	False when both a and b are true, or both are false.

### 1.4 Bitwise operators

Bitwise operator is for the operation of binary bit number. The operation number and result are

both integer. The bitwise operators are in table 1-4.

Table 1-4 Bitwise operator

Symbol	Operation	Example	Function
~	Reverse position	~x	Reverse x as per bit
>>	Right move	x >> a	Move all bits to right by a positions
<<	Left more	x << a	Move all bits to left by a positions

### 1.5 Special symbol

Table1-5 Special symbol

Symbol	Operation	Function
{ }	Bracket operator	Combination of multiple order sets, mostly used for function
[ ]	Array operator	Use for array

//	Annotation operator	Annotation for a line
/**/	Annotation operator	Annotation for a paragraph
;	Semicolon	End of a statement

### 2、 Priority and combination of the operators

The priority of operators decides the sequence of different executions in the expression. The combination decides the sequence of executions for parallel operations. See table 2-1 for the priority of system operators

Table 2-1 Priority and Combination of operators

Priority	Description	Operation	Combination
1	Top priority	[ ] { }	Left/right
2	Unary operation	~ !	Right
3	Arithmetic multiplication and division	* ? %	Left
4	Arithmetic plus and minus	+ -	Left

5	Shifting operation	>> <<	left
6	Large or small operation	> < >= <=	Left
7	Equal relationship operation	== !=	Left
8	And	&	Left
9	Logical difference or operation	^^	Left
10	OR		Left
11	Logical and	&&	Left
12	Logical or		Right
13	Assignment	=	Right

You may assign value when defining variable, with the following example:

#### DEFINE\_VARIABLE

// Define variable and directly assign initial value to variable

```
string    a = "creator";
int       b = 100;
double    c = b + 5;
boolean d = (true && false);
char      e = 'g';
byte      f = 1;
```

**Note:** When assigning value, a type of variable can only be assigned a value of that type, rather than of other types.

### 2.1.5 Assignment statement

Assignment is a basic way to change variable value and table and field, with the following format: `variable=expression;`

Example:

#### DEFINE\_VARIABLE

```
// Define variable
int a;
double b;
boolean c;
string d;
char e;
byte f;
```

#### DEFINE\_START

```
// Assign value to variable

a = 1;
b = 2.8 + 1;
c = true;
d = "hello world";
e = 'h';
f = 0;
```

### 2.1.6 Flow control statement

#### Program control statement

- One is "if" sentence for realizing dual switches. Another one is switch statement for realizing multiple switches.

##### 1.1 if statement

(1) One switch only

```
if(conditional expression){
    statement block // when
conditional expression is true (i.e. the value is
true) , run this statement block
}
```

Example:

```
int a;

a = 0;
```

```
// Execute the statement in {} only when a=0
// As in the above statement, the value of a is
changed to 0, now expression "a=0" is true ,
a=a+1 will be executed..
```

```
if(a = 0){
```

```

a = a + 1;
}

```

(2) With two switches

```

if(conditional statement){
    Statement block1 // when
conditional statement is true, execute statement
block 1
}
else{
    Statement block 2 // When
conditional statement is not true, execute
statement block 2
}

```

Example:

```

int a;

a = 0;

if(a < 0){
    a = a + 1; // execute when a is
less than 0
}else{
    a = a - 1; // execute when a is
not less than 0 (i.e. a is equal to or more than 0)
}

```

(3) With multiple switches (similar as switch statement)

```

if(conditional expression1){
    Statement block1 // execute
statement block 1 when conditional expression1
is true
}
else if(conditional expression2){
    Statement block 2 // execute
statement block 2 when conditional expression2
is true
}
else if(conditional expression3){

```

```

statement block 3 // execute
statement block 3 when conditional expression 3
is true

```

```

}
...
else{
    statement block n // execute
statement block n when above conditional
expressions are not true
}

```

Example:

```

int a;
int b;

a = 3;
b = 9;

if(a = 3){
    a = a + 1; // execute when a =
3
}else if(b = 9){
    b = b + 1; // execute when b =
9
}
if(a > 3 || b > 9){
    a = a - 1; // execute when a>3
or b>9
    b = b - 1;
}else{
    a = 0; // execute when
all above conditions are not met
    b = 0;
}

```

1.2 Switch statement

```

switch(Expression){
    case constant1:
        block 1 // execute block 1
when the value of expression is equal to constant
1
        break;

```

```

    case constant 2:
        block 2 // execute block 2
when the value of expression is equal to constant
2
    break;
...
    case constant n:
        block n // execute statement
block n when the value of expression is equal to
constant n
    break;
    default:
        statement block n // when
the value of expression is not equal to above all

```

```

constant, execute default statement block n
}

```

the type of constant n should be in accordance with the type of expression value. Assuming "int a;" is defined, which is an integer constant, so the values of "a", "a+4" and "a\*(a+4)" are all integer. At this point, constant should be integer.

Example:

```
int a = 3;
```

```

switch(a-2){
    // when a-2=1, execute the statement
under"case 1".
    case 1:
        a = a+1;
        break;

    // when a-2=2, execute the statement
under"case 2".
    case 2:

        a = a+2;
        break;

```

```

// If none of the above conditions is met,
execute the statement under"default".

```

```
default:
```

```
    a = a+3;
```

```
}
```

Note: The "break" behind each case statement means jump out of switch statement. If without "break", the statement will go on running. Example, if leaving out the "break" behind "case 1", then after executing statement "a=a+1", the program will go on running the statement "a=a+2" in "case 2". If without "break" in the end of "case 2", then the statement in "default" will also be executed.

## 2、 Loop statement

### 2.1 do...while statement

```

do{
    loop body
}while(loop conditional expression);

```

Firstly, execute loop body once, and then judge if the loop conditional expression is true. If it's false, the loop ends. If it's true, re-execute loop body and re-judge if the loop conditional expression is true. In this way, repeatedly execute it until the loop conditional expression is false.

The feature of "do...while" statement is no matter the value of loop conditional expression is true or false, loop body will be executed at least once.

Example:

```
int i = 0;
```

```

do{
    i = i + 1;
}while(i < 10 );

```

Statement "i = i + 1;" after repeatedly running it for 10 times, i=10. At this point, loop conditional

statement  $i < 10$  is not true anymore. The loop ends.

## 2.2 While statement

```
while(loop conditional expression){
    loop body
}
```

Firstly, judge if the loop conditional expression is true. If it's false, the loop ends; If it's true, execute loop body. And re-judge if the loop conditional expression is true...In this way, the loop won't be ended until the loop conditional expression is false.

The difference between "while" statement and "do...while" statement is that the loop body won't be executed unless the loop conditional expression is true.

Example:

```
int i = 0;
```

```
while( i < 10 ){
    i = i + 1;
}
```

Statement "i = i + 1;" loop will be executed for 10 times.

## 2.3 For statement

```
for(expression1; loop conditional expression;
expression 2){
    loop body
}
```

"For" loop body statement is pretty complicated, at the meantime it is the most flexible loop statement. The sequence of executing "for" statement:

- (a) Execute "expression 1"
- (b) Judge if "loop conditional expression" is true. If not, jump to step (e)

(c) Execute "loop body"

(d) Execute "expression 2", jump to step (b)

(e) Loop ends

Example:

```
int i = 10;
int n = 0;
```

```
for( i=1; i<=10; i=i+1){
    n = n * i;
}
```

## 2.4 Loop control statement

In the loop body of all loop statements, "continue" and "break" can be used for relevant control over the loop.

(1) "continue" statement

This statement is used for giving up this loop, and immediately going to next loop.

Modify the example 2.3 as follows:

```
int i = 10;
int n = 0;
```

```
for( i=1; i<=10; i=i+1){
    if( i == 5 )
        continue;

    n = n * i;
}
```

As a result, this program segment calculates that "n" is the multiplication of 1 to 4 and 6 to 10. As in the 5th loop (i.e. i=5), "continue" statement has been executed, resulting giving up this loop. Therefore, "n=n\*i" under "continue" hasn't been executed.

(2) "break" statement

This statement is for ending the whole loop.

Modify example 2.3 as follows:

```
int i = 10;
int n = 0;
```

```
for( i=1; i<=10; i=i+1){
    if( i == 5 )
        break;

    n = n * i;
}
```

As a result, this program segment calculates that “n” is the multiplication of 1 to 4, as in the 5th loop (i.e. i=5) “break” statement is executed, resulting the end of the whole loop. Therefore, the 6th, 7th...10th loop won't be executed.

### 2.1.7 WAIT statement

#### 1、 Define WAIT statement

WAIT statement is for defining the statement block to be executed after a pointed period of

time, with the following format:

```
WAIT TIME NAME{
Statement block
}
```

In which TIME must be integer constant, indicating how many millisecond to be waiting for.

NAME must be character string constant, indicating the name of WAIT statement, which is not necessary. We call the WAIT statement with name as named WAIT statement. In program, we may use “CANCL\_WAIT (name of WAIT statement)” to cancel corresponding WAIT statement. We call the WAIT statement without name as anonymous statement, which cannot be cancelled by using “CANCEL\_WAIT()”.

Note: 1. The naming rule for WAIT statement is the same as that for variable. For details, please refer to the naming rule for identifiers in [“Type and value”](#) chapter.

2. In program, any name appears in

named WAIT statement must be exclusive.

3. WAIT statement can be only for the devices defined by DEFINE\_DEVICE, the constants defined by DEFINE\_CONSTANT, the variables defined DEFINE\_VARIABLE, the function defined by DEFINE\_FUNCTION or the variable defined by the WAIT statement. Any variable appearing in other places cannot be used, neither the system internal objects like BUTTON, LEVEL and DATA.

Example:

```
DEFINE_EVENT
    BUTTON_EVENT(TP,1)
    {
        PUSH()
        {
            WAIT 1000 "tp_push_1"

            {
                TRACE("TP      PUSH
JoinNumber=1");
            }
        }
    }
```

Upon triggering the button for “TP” channel 1, WAIT statement will start timing immediately. After elapsing 1000ms, it will execute the “TRACE” statement in WAIT statement.

Example of anonymous WAIT:

```
DEFINE_EVENT
    BUTTON_EVENT(TP,1)
    {
        PUSH()
        {
            WAIT 1000
            {
                TRACE("TP      PUSH
JoinNumber=1");
            }
        }
    }
```

}

## 2. Nesting of WAIT statement

Sometimes, it is necessary to have such function: wait for a moment for TIME1 and execute operation 1, and then wait for a moment for TIME1 and then execute operation 2. For realizing this function, it is necessary to use nesting of WAIT statement.

```

WAIT TIME1 NAME1{
  Operation 1
}
// NAME2 WAIT statement nesting within NAME1
WAIT statement
WAIT TIME2 NAME2{
  Operation2
}
}

```

Internal statement won't start timing until the external WAIT statement is finished (i.e. timing has reached TIME1). And operation 2 won't be executed until the internal WAIT statement is finished. Therefore, actually operation2 has waited for TIME1+TIME2.

## 3. Canceling WAIT statement

For any named WAIT statement, function CANCEL\_WAIT(name of WAIT statement) can be used for cancelling the operation.

To cancel the operation results in:

- (1) If WAIT statement is on the run, then the timing will stop and the whole execution of WAIT will be given up.
- (2) If WAIT statement has finished timing, and is executing the statement block in WAIT statement, then it will stop and give up executing the whole WAIT statement.

Example:

```

DEFINE_EVENT
  BUTTON_EVENT(TP,1)
{

```

```

PUSH()
{

```

```

    WAIT 1000 "tp_push_1"
    {
        TRACE("TP      PUSH
JoinNumber=1");
    }
}

BUTTON_EVENT(TP,2)
{
    PUSH()
    {
        CANCEL_WAIT( "tp_push_1" );
    }
}

```

Upon triggering the button for "TP" channel 2, the "tp\_push\_1" WAIT statement will be cancelled.

### 2.1.8 Function

Function is a segment of codes with a certain feature for repeated use. If a segment of codes with the same function appears repeatedly in the program, then these codes should be written as a function, and called for the places previously stuffing with these codes.

#### 1. Definition of function

```

Return   value   type           function
name(parameter1,parameter2,...,parameter n ){
    function body
}

```

#### (1) Return value type

A function is with a return value. Return value type can be defined When defining the function. It can only be one of the following data types: "void", "int", "char", "double", "byte", "boolean" and "string", in which "void"-indicating void value, is the exclusive type for return value of function, i.e. no actual return value for the function.

## (2) Function name

The naming rule is the same as that for the variable. For details, please refer to [Type and Value](#) chapter.

## (3) Parameters

In a function, there are some parameters, but it can be none. The full definition of parameter is with "type parameter\_name". if there are multiple parameters, each parameter should be separated by ",". Like the variable type, only one out of "int","char","double","byte","boolean"and"string" can be used.

The naming rule of parameter is the same as the variable. For details, please refer to [Type and Value](#) chapter.

## (4) Function value with return

For those functions without void return value, a "return" statement is needed for returning corresponding type of value. "return" statement can be placed at any position of the function body. "return" statement can also be used for those functions with void return value.

For details, please refer to the following examples:

### 2. Example of defining function

#### (1) Function with void return value, without parameter

Function fun1 outputs character string "fun1"

```
void fun1(){
    TRACE("fun1");
}
```

Assume to define integer variable "n" in DEFINE\_VARIABLE

Function "fun2" plus or minus 1 to adjust variable n toward 10.

```
void fun2()
{
```

```
    if( n > 10 )
    {
        n = n -1;
        return;
    }
    else if( n < 10 )
    {
        n = n + 1;
        return;
    }
}
```

#### (2) Function with one parameter

Function Square calculates the square of integer "n" transferred from parameters, and returns the result.

```
int Square( int n ){
    return n*n;
}
```

Function GetWage takes the floating point "base" from parameters as salary base, and multiples it by 0.92 to have outstanding salary and with calculation result returned.

```
double GetWage( double base ){
    double money = 0;
    double rate = 0.92;
    money = base * rate;
    return money;
}
```

Function PrintComputerName outputs the character string computer\_name from parameters.

```
void PrintComputerName( string
computer_name ){
    TRACE( "Computer Name: " +
computer_name );
}
```

#### (3) Function with multiple parameters

Function Add calculates the parameter a1+a2,

and has the result returned.

```
int Add( int a1, int a2 ){
    return a1 + a2;
}
```

Function Max finds the maximum value out of a1, a2 and a3, and has the result returned.

```
int Max( int a1, int a2, int a3 ){
    int max = 0;

    if( a1 > a2 )
        max = a1;
    else
        max = a2;

    if( a3 > max )
        max = a3;

    return max;
}
```

### 3. Calling function

For the functions defined in above examples, the calling should be in the following way:

(1) Calling the function without parameter

Calling fun1

```
fun1();
```

Calling fun1

```
fun2();
```

(2) Calling the function with parameter

Calling function Square

```
int s = 0;
s = Square(20);           // calculate the square
of 20 and save the result into the variable s
```

```
int a = 4;
s = Square(a);           // calculate the square
of a and save the result into the variable s
s = Square(a+3);        // calculate the square of
"a+3" and save the result into the variable s
```

Calling function GetWage

```
double money = GetWage( 1000 );           //
```

Calculate the salary on the base of 1000, and save the result into variable money

Calling function PrintComputerName

```
PrintComputerName("IBM ThinkPad");
```

Calling function Add

```
int sum = 0;
sum = Add(10,20);           // Calculate the sum
of 10+20, and save the result into the variable
sum
Add(sum,30);               // Calculate the
sum of 10+20+30 without saving the result
```

Calling function Max

```
int num1 = 10;
int num2 = 4;
int num3 = 109;
int max_num = Max(num1,num2,num3);
// Take the maximum value out of
num1,num2,num3, and save it into the variable
max_num
```

```
max_num = Max(Max(1,2,3),4,5);
// Take the maximum value out of 1,2,3,4,5, and
save it into the variable max_num, please note
that how to use the return value of a function as
the parameter of another function.
```

(3) Using return value

For the function with non-void return value, a variable with the same type of return value can be used for saving the return value of the function, as the above example of calling function.

## 2.2 Code Organization

### 2.2.1 Composition of code organization

```
// Define device block
DEFINE_DEVICE
```

```
// Define constant block
DEFINE_CONSTANT

// Define variable block
DEFINE_VARIABLE

// Define function block
DEFINE_FUNCTION

// Program initialization statement block
DEFINE_START

// Program loop statement block
DEFINE_PROGRAME

// Define event block
DEFINE_EVENT
```

Instruction:

(1) Each block has its own function, e.g. DEFINE\_DEVICE is special for defining device. All devices being used in the program must be defined in this block.

(2) Changing the order within the blocks is not recommended.

### **DEFINE\_DEVICE: Define device block**

Defining device must be in this block

Format of defining device

```
Device name=[type of carrier device]:[carrier device ID]:[ element device type];
```

#### 1、 Device name

The naming rule for device is the same as that for variable. For details, please refer to [Type and Value](#) chapter.

#### 2、 Type of carrier device

Note: The carrier device refers to a big device, on which there are other devices. For example, main

board is a big device, on which there are serial port device, relay, infrared port and IO port etc.

As to the touch screen, carrier device integrates with element device. T is for carrier type; TP is for element device type.

[Device type]	Description
M	Main board
T	Touch screen device
N	CRNET Device
L	CRLINK Device

#### 3、 Carrier device ID

Integer constants like: 1001,2001,3003.....

#### 4、 Element device type

Note: Element devices refer to the small devices on carrier device, such as serial port, light and

volume etc.

#### I、 Type of device element

[Type of device element]	Description
RELAY	Relay
COM	COM port
TP	Touch screen
IR	Infrared
IO	Input/output
LITE	Light modulator
VOL	Volume control
WM	Panel on wall
DMX512	512 Marquee

II、 Element devices are all with corresponding channel numbers

#### 5、 Example

##### DEFINE\_DEVICE

```
RELAY =M:1001:RELAY; // Define the relay on main board 1001
```

```
COM =M:1001:COM; // Define the COM port devices on main board 1001
```

```
    Tp1      =T:1002:TP; // Define the touch
screen with ID 1002
```

```
    RELAY2   =S:2001:RELAY; //Define the
```

```
relay on cascading connection board 2001< br >
RELAY3      =N:2004:RELAY; // Define CRNET<
/ span >Relay on device 2004
```

### DEFINE\_COMBINE: Define device block

Note:

This block is especially for multiple touch screens. If in practice multiple touch screens are needed, this module may provide good matching.

Format:

```
[tp1,tp2]; //tp1 equal to tp2
```

Example:

```
[tp1,tp2,tp3];
```

```
[tp1,tp4];
```

```
[tp5,tp6];
```

Indicating tp1 == tp2==tp3==tp4  
tp5==tp6 but unequal to tp1, tp2,tp3, tp4.

### DEFINE\_VARIABLE: Define variable block

The definition of variable must be in this block

Format of defining variable:

```
Variable type Variable name;
```

Example:

```
DEFINE_VARIABLE
```

```
    int x = 222;           // Define integer
```

variable and initializing

```
    float f = 2.5;        //Define float
```

variable and initialize

```
    string s = "hello";   //Define character
```

string and initialize

```
    char c;               //Define
```

character variable

```
    boolean b;           //Define
```

Boolean variable

```
    byte w;              //Define byte
```

variable

### DEFINE\_CONSTANT: Define constant block

Constant definition must be placed into this block

Format of constant definition:

```
Name of constant= integer constant;
```

Example:

```
DEFINE_CONSTANT
```

```
    const_1 = 222;
```

### DEFINE\_FUNCTION: Define function block

Function definition must be placed into this block.

Example:

```
DEFINE_FUNCTION
```

```
    int fun1()
    {
        return 200;
    }
```

### DEFINE\_TIMER: Define timer block

Timer definition must be placed into this block.

Note: To meet some actual needs, it is necessary

to define a timer for executing some action at set intervals.

Format:

```
Timer function name()
{
    // execute actio
}
```

Call:

```
startTimer(function name, interval unit is ms);
cancelTimer("function name"); //note: function
name should be in quotation marks
```

Example:

```
DEFINE_TIMER
```

```
Timer setVol()
```

```
{
    if(65536 >vol_value)
    {
        SET_VOLTOTOL(dev_vol,2,vol_value);
        vol_value = vol_value +100;
    }else
    {
        cancelTimer("setVol");
    }
}
```

```

}

// Call execute every 1 second
startTimer(setVol,1000);
// close

```

```
cancelTimer("setVol");
```

Note: Called from setVol body, close cancelTimer function, prevent from wasting resources.

### DEFINE\_START: Program initialization statement block

When start to execute the program, the statement here is firstly executed. Program initialization statement block can be used for initialization function like initializing variables, executing the following initializing operations etc.

```
DEFINE_START
```

```
    x=20;           // initializing variable
```

```
    ON_RELAY(relay,1); // turn on relay way 1
```

**Event definition must be placed into this block.**

#### 1. Button event

Grammar rule: the number of parameters can be 0, 1 or 2. That's to say, when defining device name and joinNumber, event is only effective for appointed devices. When the number of parameter is 0, event is effective for all devices. In the event, there are functions for responding the actions of pressing, releasing, holding and full button process. Event processing code must be placed into corresponding function.

```
BUTTON_EVENT([device] [,JoinNumber])
```

```

{
    PUSH()
    {
        // action executed when pressing button
    }
    RELEASE()
    {
        // action when releasing button
    }
}

```

```

}
    HOLD(<TIME>[,TRUE|FALSE])
    {
        // action repeatedly executed when
        // pressing button for a set time/or in an interval.
    }
}

```

```

    REPEAT()
    {
        // Action repeatedly executed when
        // pressing button
    }
}

```

Example:

```
DEFINE_EVENT
```

```
    // responding device tp, JionNumber=3 的
    // BUTTON event
```

```
    BUTTON_EVENT(tp,3)
```

```

    {
        PUSH()
        {
            TRACE("BUTTON_EVENT(tp,3)
            push\n");
            int i = 34;
            ON_RELAY(relay1);
            ON_RELAY(relay2);
        }

        HOLD(2000)
        {
            TRACE("HOLD");
        }

        REPEAT()
        {
            TRACE("REPEAT");
        }
    }

    // responding all BUTTON event in device tp
    BUTTON_EVENT(tp)
    {

```

```

    PUSH()
    {
        TRACE("BUTTON_EVENT(tp,3)
push\n");
    }
}

// responding all BUTTON event
BUTTON_EVENT()
{
    RELEASE()
    {
        TRACE("BUTTON_EVENT(tp,3)
push\n");
    }
}

```

## 2. Draw bar event

Grammar rule: the number of parameter can be 0, 1 or 2. When with 2 parameters, that's when defining device name and channel number, the event is only effective for appointed device, appointed joinNumber; When with 1 parameter, that's for appointed device, the event is effective for the appointed device. When with 0 parameter, the event is effective for all devices.

```

LEVEL_EVENT([device] [, JoinNumber])
{
    // action executed when drawing bar
}

```

Example:

```

DEFINE_EVENT
    // responding to LEVEL event of device tp,
    JoinNumber= 3
    LEVEL_EVENT(tp,2)
    {
        TRACE("LEVEL_EVENT(tp,2)\n");
        OFF_RELAY(relay3);
        ON_RELAY(relay2);
    }

```

```

// responding to all LEVEL events of device tp
LEVEL_EVENT(tp)
{
    TRACE("LEVEL_EVENT(tp)\n");
    OFF_RELAY(relay1);
    ON_RELAY(relay2);
}

// responding to all LEVEL events
LEVEL_EVENT(){
    TRACE("LEVEL_EVENT()\n");
    OFF_RELAY(relay2);
    ON_RELAY(relay3);

    TRACE( LEVEL.DeviceType + " " +
        LEVEL.DeviceID + " " +
        LEVEL.Module + " " +
        LEVEL.Channel + " " +

        LEVEL.Value + " " +
        LEVEL.JoinNumber + "\n");
}

```

## 3. Data event

```

DATA_EVENT([device])
{
    ONLINE()
    {
        // action executed when receiving online
        data order from the device
    }
    OFFLINE ()
    {
        // action executed when receiving offline
        data order from the device
    }
    ONERROR ()
    {
        // action executed when receiving data
        error order from the device
    }
    ONDATA()
    {

```

```
// action executed when receiving data
from the device
}
}
```

Example:

**DEFINE\_EVENT**

```
// responding to DATA event in com device
DATA_EVENT(com,1)
{
    ONLINE()
    {
        TRACE("DATA_EVENT(relay1)\n");
    }
}
```

```
// responding to DATA events in all com
```

devices

```
DATA_EVENT(com)
{
    ONLINE()
    {
        TRACE("DATA_EVENT(relay1)\n");
    }
}
```

### **DEFINE\_PROGRAM: Program loop statement block**

After starting the program, these statements will be constantly in loop execution. Monitoring work can be applied here, such as monitoring the status of some devices.

## **2.2.2 Control device function**

### **1、SEND\_IRCODE**

```
void SEND_IRCODE (String dev,int
channel,String str)
```

Function: send infrared data

Parameters:

dev - :infrared device

channel - : device number

str - : infrared data HEX String

Example: ,

```
IR_M = M:1000:IR; //Define the infrared
devices on main board IR_M
```

```
//Send infrared code to channel 1 at IR_M
```

```
// In which
```

```
IRCODE<"StanderIRDb:3M:CODEC:VCS3000:P
OLYCOM1:6289:6 (MNO)"> is the 3M company in
standard library,
```

```
// CODEC type, VCS3000 type, POLYCOM1
controlled terminal, (MNO) infrared code under
infrared sample number 6289.
```

```
// call the function, extract the infrared code from
standard library and send it out.
```

```
SEND_IRCODE(IR_M,1,IRCODE<"StanderIRDb:
```

```
3M:CODEC:VCS3000:POLYCOM1:6289:6
(MNO)">);
```

### **2、ON\_RELAY**

```
void ON_RELAY(String dev,int channel)
```

Function: turn on relay

Parameters:

dev - :relay device

channel - :device channel number

Example:

```
RELAY_M = M:1000:RELAY; //Define the relay
No.1000 on main board
```

```
ON_RELAY(RELAY_M,2); // Turn on the 2nd
way of relay No.1000 on main board
```

### **3、OFF\_RELAY**

```
void OFF_RELAY(String dev,int channel)
```

Function: Turn off relay

Parameters:

dev - : Relay device

channel - :Device channel

Example:

```
RELAY_M = M:1000:RELAY; //Define the
relay No.1000 on main board
OFF_RELAY(RELAY_M,2); // Turn off the 2nd
way of relay No.1000 on main board
```

#### 4、SET\_COM

```
void SET_COM(String dev,
int channel,
long sband,
int databit,

int jo,
int stopbit,
int dataStream,
int comType)
```

Function: configure COM port

Parameters:

dev - : device name

channel - : device channel No.

sband - : Baud rate

databit - : Data bit 1~8

jo - : parity bit 0: none, 1: odd number, 2: even number, 3: mark, 4: space

stopbit- : stop bit 10,15,20, corresponding 10=1,15=1.5, 20=2

dataStream - : data stream: 0: none, 1: xon/xoff, 2: hardware

comType - : serial port communication 232、485、422 are not these 3 values, as default is 232

Example:

```
Com_m = M:1000:COM; //define the serial port
No.1000 on main board
```

```
// configure way 1 at serial port Com_m (i.e.
define the 1st serial port No.1000on main board)
```

```
//Baud rate is 9600, data bit is 8, no parity, stop
bit is 1, no data stream, communication way is
232
```

```
SET_COM(Com_m,1,9600,8,0,10,0,232);
```

#### 5、SEND\_COM

```
void SEND_COM(String dev,int channel, String
```

```
str)
```

Function: sending data from serial port

Parameters:

dev - : serial port device

channel - : device channel number

str - : serial port data, two format supported

1 : Directly transfer character string data (send complete character string to serial port),

2 : Convert 16-bit character string (for character string starting with 0x or 0X, send the string after converting it into 16 bit. If send

0x3132, character string of "12" will be received at serial port.)

Example:

```
Com_m = M:1000:COM; //define serial port
No.100 on main board
```

```
SEND_COM(Com_m,1,"1234"); // send character
```

```
string "1234" to the 1st way of serial port on main
board
```

```
SEND_COM(Com_m,1,"0x31323334"); // send
character string "1234" to 1st way of serial port on
main board."
```

#### 6、SEND\_IO

```
void SEND_IO(String dev,int channel,int val)
```

Function: control IO port

Parameters:

dev - : io device

channel - : device channel number

val - : (Note: get value as per specific peripheral device) generally 0 or 1.

Example:

```
Io_m = M:1000:IO; //define the IO No.1000 on
main board
```

```
SEND_IO(Io_m,1,0); // output low level to 1st way
of Io_m
```

#### 7、READ\_IO

```
int READ_IO(String dev,int channel)
```

Functionj: control IO port

Parameters:

dev - : io device

channel - : device channel number

Return: Return to high/low level status at IO channel, with value of 0 or 1. It's wrong with others.

Example:

```
lo_m = M:1000:IO; // define IO No.1000 on main board
```

```
int iostate =READ_IO(lo_m,1); //read way 1 status of lo_m
```

### 8、SEND\_LITE

```
void SEND_LITE(String dev,int channel,int val)
```

Function: control light

Parameters:

dev - :light device

channel - :device channel number

val - : analog quantity (Note: get value as per specific peripheral device)

Example:

```
lite_n = N:8:LITE;; //define light device with CRNET No.8
```

```
SEND_LITE (lite_n,1,65535); //Send analog quantity65535 to way 1 of lite_n
```

### 9、SEND\_DM512

```
void SEND_DM512(String dev,int channel,int val)
```

Function: control DMX512

Parameters:

dev - : light device

channel - : device channel number

val - : analog quantity (Note: get value as per specific peripheral device)

Example:

```
liit_L = L:7:DMX512; //define DMX512 device with CRLINK(CAN) NO.7
```

```
SEND_DM512(liit_L,1,65535);// send analog quantity 65535 to way 1 of liit_L
```

### 10、SEND\_ACAR

```
void SEND_ACAR(String dev,int channel,int val)
```

Function: control voltage output from analog card

Parameters:

dev - :analog card device

channel - : device channel number

val - : analog quantity (Note: get value as per specific peripheral device)

(Normal range from double type -12 (v) to 12 (v))

Example:

```
acar_L = L:7:ACAR; //define ACAR device with
```

```
CRLINK(CAN) No.7
```

```
SEND_ACAR( acar_L,1,-12);// send analog quantity-12 to way 1 of liit_L, i.e. configure -12 output from analog card
```

### 11、SEND\_QACAR

```
Void SEND_QACAR (String dev,int channel)
```

Request voltage value in analog card. After request, DataEVENT on analog card will be triggered, from where voltage value can be acquired. For detailed cases, see BYTES\_TO\_INT of other functions.

Parameters:

dev - : Analog card device

channel - : device channel number

Example:

```
Acar_m = M:8:ACAR; //define analog card No.8 on main board
```

```
SEND_QACAR (Acar_m,1); // read voltage value of way 1 of Acar_m
```

### 12、ON\_VOL

```
void ON_VOL(String dev,int channel)
```

Function: Turn volume

Parameters:

dev - :volume device

channel - : device channel number

Example:

```
vol_N = N:9:VOL; // define volume device vol_N
with CRNET device No.9
ON_VOL(vol_N,1); // turn on way 1 of vol_N
```

### 13、OFF\_VOL

```
void OFF_VOL(String dev,int channel)
```

Function: turn off volume

Parameters:

dev - : volume device

channel - :device channel number

Example:

```
vol_N = N:9:VOL; // define volume device vol_N
with CRNET device No.9
ON_VOL(vol_N,1); //turn off way 1 of vol_N
```

### 14、SET\_VOLTOTOL

```
void SET_VOLTOTOL(String dev,int channel,int
val)
```

Function: master volume control

Parameters:

dev - : volume device

channel - :device channel No.

val - : analog quantity (Note: get value as per specific peripheral device)

Example:

```
vol_N = N:9:VOL; // define volume device vol_N
with CRNET device No.9
SET_VOLTOTOL(vol_N,1,600);//configure the
master volume value of way 1 in vol_N device as
600
```

### 15、SET\_VOLHIGHT

```
void SET_VOLHIGHT(String dev,int channel,int
val)
```

Function: treble volume control

Parameters:

dev - :volume device

channel - :device channel No.

val - : analog quantity (Note: get value as per specific peripheral device)

Example:

```
vol_N = N:9:VOL; // define volume device vol_N
with CRNET device No.9
SET_VOLHIGHT (vol_N,1,600);// configure the
treble volume of way 1 in vol_N device as 600.
```

### 16、SET\_VOLLOW

```
void SET_VOLLOW(String dev,int channel,int val)
```

Function: bass volume control

Parameters:

dev - : volume device

channel - :device channel number

val - : analog quantity (Note: get value as per specific peripheral device)

Example:

```
vol_N = N:9:VOL; // define volume device vol_N
with CRNET device No.9
SET_VOLLOW (vol_N,1,600);// configure the
bass volume of way 1 in vol_N device as 600.
```

### 17、UP\_WM

```
void UP_WM(String dev,int channel)
```

Function: Send pop-up message to on-wall panel, for information exchange when using touch screen along with on-wall panel to control over the same device.

Parameters:

dev - : panel device

channel - :device channel No.

Example:

```
wm_N = N:14:WM; // define on-wall device with
CRNET device No.14
UP_WM(wm_N,1);//request pop-up status in way
1 of wm_N device
```

### 18、DOWN\_WM

```
void DOWN_WM(String dev,int channel)
```

Function: Send pressing message to on-wall panel, for information exchange when using touch screen along with on-wall panel to control over the same device.

Parameters:

dev - : panel device

channel - : device channel No.

Example:

```
wm_N = N:14:WM; // define on-wall device with
CREAT device No.14
```

```
DOWN_WM(wm_N,1);// Request way 1 of wm_N
device is at down state
```

## 19、DEV\_REG

```
void DEV_REG(String dev, int channel)
```

Function: Device registration, mainly for on-wall

devices of pgm2 generation

Parameters:

dev – input device

channel - : device channel No.

Example:

```
wm_N = N:14:WM; // define on-wall panel device
with CRNET device No.14
```

```
DEV_REG(wm_N,1);// register way 1 of on-wall
panel wm_N
```

## 20、DEV\_QUERY

```
void DEV_QUERY(String dev, int channel)
```

Function : device query Function : device registration, mainly for the query of on-wall panel device of pgm2 generation

Parameters:

dev – input device

channel - :device channel No.

Example:

```
wm_N = N:14:WM; // define on-wall panel device
with CRNET device No.14
```

```
DEV_QUERY (wm_N,1);//Query for way 1 of
```

wm\_N of on-wall panel

## 2.2.3 Other functions

### 1、TRACE

```
void TRACE(string msg)
```

Function: print message

Parameter: msg - : character string indicates the content of message.

Example:

```
TRACE("This is my program!");
```

```
//print character string"This is my program!"
```

### 2、START\_TIMER

```
void START_TIMER(String name,int time);
```

Start a named timer with execution interval in ms

unit, used along with CANCEL\_TIMER(XXX)

Parameter: name -: Timer name, i.e. XXX in

```
START_TIMER(XXX)
```

Example:

- Define Timer triggered function

```
TIMER testTimer()
```

```
{
```

```
    SEND_LITE (lite_n,1,65535); //send analog
quantity 65535 to way 1 in lite_n
```

```
}
```

- Run a timer, and set interval as 1000 ms.

```
START_TIMER(testTimer,1000);
```

- Cancel a running timer

```
CANCEL_TIMER("testTimer");// Note: double
quotation marks are needed here.
```

### 3、START\_TIMER

```
void START_TIMER(String name,int time ,int
year,int mouth,int day,int hh,int minute,int
second);
```

Start a named timing executor at year, mouth, day, hh, minute, second, with executing interval in ms

unit.

Used along with CANCEL\_TIMER(XXX)

parameter: name -: Timer name, i.e.XXX in START\_TIMER(XXX)

Example:

- Define Timer triggered function

```
TIMER testTimer()
{
    SEND_LITE (lite_n,1,65535); //send analog
quantity 65535 to way 1 in lite_n
}
```

- Run a timer and set interval by day, and set timer starting time as 2010,10,26,14,00,00.

```
START_TIMER(testTimer,1000*3600*24
2010,10,26,14,00,00);
```

- Cancel a running or initiated (by

```
START_TIMER(testTimer,1000*3600*24
2010,10,26,14,00,00)) but without executing
timer.
```

```
CANCEL_TIMER("testTimer");// Note: Double
quotation marks are needed here.
```

#### 4、CANCEL\_TIMER

```
void CANCEL_TIMER(String name);
```

Function: cancel a named timer, used along with START\_TIMER(XXX,t).

Parameter: name -: Timer name, i.e. XXX in START\_TIMER(XXX)

Example:

- Define Timer triggered function

```
TIMER testTimer()
{
    SEND_LITE (lite_n,1,65535); // send analog
quantity 65535 to way 1 in lite_n
}
```

- Run a timer and set internal as 1000 ms
- ```
START_TIMER(testTimer,1000);
```

- Cancel a running timer

```
CANCEL_TIMER("testTimer");// Note: Double
quotation marks are needed here.
```

#### 5、WAIT

Format: WAIT numeric constant or WAIT numeric constant "name of statement block"

Function: Similar with SLEEP function, delay the operation in WAIT statement block for a certain period of time (minimum unit is ms in WAIT). Unlike SLEEP function, this statement block won't influence other operations by touch screen.

Classification: Depending on definition formats, there are anonymous WAIT statement block and named WAIT statement. The anonymous one will be assigned name by system. CANCEL\_WAIT

can't be used for cancelling the running operation. Named WAIT statement block may call CANCEL\_WAIT function to cancel the running WAIT statement block.

Example:

- anonymous

```
WAIT 1000
{
    ON_RELAY(relay_M,2);//delay for
1s and turn on the way 2 in relay_M
}
```

- Named

```
WAIT 3000 "xxx"
{
    ON_RELAY(relay_M,2);// delay for
3s and turn on the way 2 in relay_M}
```

- Cancel

```
CANCEL_WAIT("xxx ");
```

#### 6、CANCEL\_WAIT

```
void CANCEL_WAIT(string name)
```

Function: cancel a named WAIT statment

Parameter: name -: name of WAIT statement

Example:

```
// button event processing in tp channel 3
BUTTON_EVENT(tp,3)
{
    PUSH()
    {
        WAIT 4000 "ty" // This WAIT
statement should be "ty"
        {
            TRACE("WAIT 4000 \"ty\"\n");
        }
    }
}

// cancel the WAIT statement named "ty".
BUTTON_EVENT(tp,4)

{
    PUSH()
    {
        CANCEL_WAIT("ty");
    }
}
```

## 7、 SLEEP

void SLEEP(int time)

Function : let the program sleep for a while

Parameters:

time – sleeping time. Unit: ms

Example:

```
BUTTON_EVENT(tp,4)
{
    PUSH()
    {
        SLEEP(1000); //sleep for 1s
here (Note: Main thread will sleep for 1 second)
    }
}
```

## 8、 BYTES\_TO\_STRING

String BYTE\_TO\_STRING(byte[] bb)

Function : bit converted to character string

Parameters:

bb : bit array object

Returns:

System default coding character string

Example:

```
DEFINE_VARIABLE
string tt;
string xxx;
byte bb[0];
DEFINE_EVENT
BUTTON_EVENT(TP1,2)
{
    PUSH()
    {
        tt = "xxxxx123sdf456";

        bb = STRING_TO_BYTES(tt);
        xxx = BYTES_TO_STRING(bb);

        TRACE(xxx );
    }
}
```

(Note: Here the variable is suggested to be defined under DEFINE\_VARIABLE label.)

## 9、 STRING\_TO\_BYTES

Byte[] STRING\_TO\_BYTES(string str)

Function : Character string converted into dynamic bit array.

Parameters:

str: Character string object

Returns:

System default coding bit array.

Example: See above BYTE\_TO\_STRING

## 10、 STRING\_EQ

boolean STRING\_EQ(String src,String des)

Function : compare 2 character strings strictly case sensitive.

Parameters:

src –compare character 1

des – compare character 22

Returns:

src means des returns to true and others return to false.

Example:

**DEFINE\_VARIABLE**

```
string t1;
```

```
string t2;
```

**DEFINE\_EVENT**

```
BUTTON_EVENT(TP1,2)
```

```
{
```

```
    PUSH()
```

```
    {
```

```
        t1 = "123";
```

```
        t2 = "122";
```

```
        if( STRING_EQ(t1,t2) )
```

```
        {
```

```
            TRACE("t1 == t2" );
```

```
        }
```

```
        else
```

```
        {
```

```
            TRACE("t1 != t2" );
```

```
        }
```

```
    }
```

```
}
```

// Press the button of touch screen TP1 joinmuber=2, with following result.

// t1 != t2

## 11、STRING\_EQNOCASE

**boolean STRING\_EQNOCASE(String src,String des)**

Function : Comparing two character strings, which is not case sensitive.

Parameters:

src – source character string

des – target character string

Returns:

Under the condition of case non-sensitive, src is equal to des returning to true, while others returning to false.

Example:

```
t1 = "abc";
```

```
t2 = "ABc";
```

```
if( STRING_EQNOCASE(t1,t2) )
```

```
{
```

```
    TRACE("t1 == t2" );
```

```
}
```

```
else
```

```
{
```

```
    TRACE("t1 != t2" );
```

```
}
```

// Output result

// t1 == t2

## 12、STRING\_STARTWITH

**boolean STRING\_STARTWITH(String s1,String s2)**

Function: see if the heads are matched

Parameters:

S1- character string 1

S2 –character string 2

Returns:

Both S1 and S2 start from the begin, see whether S1 character string exists in S2?

Example:

```
t1 = "12";
```

```
t2 = "123";
```

```
if( STRING_STARTWITH(t1,t2) )
```

```
{
```

```
    TRACE("t1 == t2" );
```

```
}
```

```
else
```

```
{
```

```
    TRACE("t1 != t2" );
```

```
}
```

// Output result

```
// t1 == t2
```

### 13、STRING\_ENDWITH

```
boolean STRING_ENDWITH(String S1,String S2)
```

Function: see if the ends are matched

Parameters:

S1 – character string 1

S2 – character string 2

Returns:

Both S1 and S2 start from the end. See if S1 character string exists in S2.

Example:

```
t1 = "1234";
```

```
t2 = "XXXX1234";
```

```
if( STRING_ENDWITH(t1,t2) )
```

```
{
    TRACE("t1 == t2" );
}
```

```
else
```

```
{
    TRACE("t1 != t2" );
}
```

```
// Output result
```

```
// t1 == t2
```

### 14、atoi

```
int atoi(String a)
```

Function: character type converted into int type.

Parameters:

a - :String data type input

Returns:

: int data type returned.

Example:

```
DEFINE_VARIABLE
```

```
string t1;
```

```
int i2;
```

```
DEFINE_EVENT
```

```
BUTTON_EVENT(TP1,2)
```

```
{
    PUSH()
```

```
{
    t1 = "1234";
    i2 = atoi(t1);
}
```

### 15、itoa

```
String itoa(int i)
```

Function : Convert int type data into String (character string)

Parameters:

i - : Input int value to be converted

Returns:

: return character string, failed return

Example:

```
DEFINE_VARIABLE
```

```
string t1;
```

```
int i2;
```

```
DEFINE_EVENT
```

```
BUTTON_EVENT(TP1,2)
```

```
{
    PUSH()
    {
        i2= 1234;
        t1= itoa(i2);
    }
}
```

### 16、BYTES\_ADD

```
byte[] BYTES_ADD(byte[] first,byte[] second)
```

Function : Add parameter 2 to the end of parameter 1, making up of a new byte and return.

Parameters:

first - : parameter1, byte array

second - : parameter 2, byte array

Returns:

: return new byte array

Example:

```
DEFINE_VARIABLE
```

```
byte c1Bytes[0];
```

```
int c1Blen;
```

```
DEFINE_START
```

```

    RESET_BYTE( c1Bytes ); //Empty c1Bytes,
this is a good habit
DEFINE_EVENT
    DATA_EVENT(com_M,1)
    {
        ONDATA()
        {
            c1Bytes =
BYTE_ADD(c1Bytes,DATA.Data); // constantly
add the data received from serial port 1 to the end
of c1Bytes (Here DATA.Data is the internal object
of DATA_EVENT, and serial port data)
//Note:
returning with result of adding two byte arrays
            c1Blen =
GET_BYTES_LENGTH(c1Bytes); // Acquire the
length of c1Bytes
            if(c1Blen > 30 )

                {
                    //Process the data at serial port
1 on main board

                    TRACE(BYTES_TO_HEX(c1Bytes));
//Empty c1Bytes, receive the
data arrived next time, otherwise c1Bytes will
constantly add serial port data to the end.
                    RESET_BYTE( c1Bytes );
                }
            }
        }
    }
}

```

### 17、GET\_BYTES\_LENGTH

```
int GET_BYTE_LENGTH(byte[] bsrc)
```

Function: acquire the length of dynamic byte array

Parameters:

bsrc - :dynamic byte array

Returns:

int: Return with the length of byte array.

Example: see above BYTE\_ADD example

### 18、BYTES\_TO\_HEX

```
String BYTE_TO_HEX(byte[] bb)
```

Function: convert dynamic byte array into character string in hexadecimal format

Parameters:

bb - : dynamic byte array

Returns:

: return with character string in hexadecimal format

Example: see BYTE\_ADD example

### 19、HEX\_TO\_BYTES

```
byte[] HEX_TO_BYTES(String s);
```

Function: convert hexadecimal character string into dynamic byte array

Character : hexadecimal character string"FF00EFFF"

Return: dynamic byte array

Example:

```
byte[] b = HEX_TO_BYTES("FF01EF");
```

Result after execution

```
b[0]:0xFF;b[1]:0x01;b[2]=0xEF;
```

### 20、GET\_YEAR

```
int GET_YEAR()
```

Function: acquire the "year" in the clock of system

Returns:

: return with year in int type, like 2010

Example:

```
int year = GET_YEAR();
```

### 21、GET\_MONTH

```
int GET_MONTH()
```

Function: acquire the "month" in the clock of system

Returns:

: return month in int type, like 7

Example:

```
int month = GET_MONTH();
```

### 22、GET\_DATE

```
int GET_DATE()
```

Function: acquire the "day" in the clock of system

Returns:

: return with day in int type, like 26

Example:

```
int day= GET_DATE();
```

### 23、GET\_HOUR\_OF\_DAY

```
int GET_HOUR_OF_DAY()
```

Function: acquire the hour in the clock of system

Returns:

: Return with the 'hour" in int type, like 23

Example:

```
int day = GET_HOUR_OF_DAY();
```

### 24、GET\_MINUTE

```
int GET_MINUTE()
```

Function: acquire the minute in the clock of system

Returns:

: Return with the 'minute" in int type, like 59

Example:

```
int minute = GET_MINUTE();
```

### 25、GET\_SECOND

```
int GET_SECOND()
```

Function: acquire the second in the clock of system

Returns:

: Return with the 'second" in int type, like 59

Example:

```
int second= GET_SECOND();
```

### 26、GET\_DAY\_OF\_WEEK

```
int GET_DAY_OF_WEEK()
```

Function: acquire the day of week in the clock of system

Returns:

: Return with the "day" in int type, like 1

Example:

```
int week = GET_DAY_OF_WEEK ();
```

### 27、INT\_TO\_DOUBLE

```
double INT_TO_DOUBLE(int i)
```

Function: convert int type into double type

Returns:

Return with the converted double-type value

Example:

```
Double d = INT_TO_DOUBLE (10);
```

### 28、DOUBLE\_TO\_INT

```
Int DOUBLE_TO_INT(double i)
```

Function: convert double-type into int-type

Returns:

Return with the converted int-type value

Example:

```
int i = INT_TO_DOUBLE (10.12);// i value is 10
```

### 29、STRING\_TO\_DOUBLE

```
double STRING_TO_DOUBLE (string s)
```

Function: convert string-type into double-type

Returns:

Return with the converted double-type value

Example:

```
double d = STRING_TO_DOUBLE ("10.12");//d = 10.12
```

### 30、DOUBLE\_TO\_STRING

```
Int DOUBLE_TO_STRING (double i)
```

Function: convert double-type into string-type

Returns:

Return with the converted string-type value

Example:

```
string s = DOUBLE_TO_STRING (10.12);//s value is"10.12"
```

### 31、BYTES\_TO\_INT

```
Int BYTES_TO_INT (byte[] b)
```

Function: Process the front 4 digits of the byte array as an int value, if b is less than 4 digits, then convert all digits.

Returns:

Return with the converted int-type value

Example: return with voltage data in analog card, actual voltage in analog card (millivolt)

```

DATA_EVENT(mcar,2)
{
    ONDATA()
    {
        double          dmv          =
BYTES_TO_INT(DATA.Data) //Triggered here
when use SEND_QACAR to send request,

        SEND_COM(COM,1,DOUBLE_TO_STRING
(dmv));
    }
}

```

#### 2.2.4 Sample program

When creating a new project, a void program

template will be automatically created as follows:

```
DEFINE_DEVICE
```

```
DEFINE_CONSTANT
```

```
DEFINE_VARIABLE
```

```
DEFINE_FUNCTION
```

```
DEFINE_START
```

```
DEFINE_PROGRAME
```

```
DEFINE_EVENT
```

This program has not any content, only for compiling and running, but doing nothing. The following codes realize a simple device shut-down event.

```
DEFINE_DEVICE
```

```

tp = T:1002:TP:192.168.1.1; // touch
screen
relay1 = S:2001:RELAY:192.168.1.1;
relay2 = N:2001:RELAY:192.168.1.1;

```

```
DEFINE_CONSTANT
```

```
const_1 = 22;
```

```
DEFINE_VARIABLE
```

```
int x = 222;
```

```
DEFINE_FUNCTION
```

```

int fun1(){
    return x;
}

```

```
DEFINE_MUTUALLY_EXCLUSIVE
```

```
DEFINE_START
```

```

x=20; // initialize variable
ON_RELAY(relay1,1);

```

```
DEFINE_PROGRAME
```

```
DEFINE_EVENT
```

```

BUTTON_EVENT(tp,3)
{
    PUSH()
    {
        int i = fun1();
        ON_RELAY(relay1,1);
        ON_RELAY(relay2,1);
    }
}

```

For other samples, please refer to Control Device Function and other Functions.





CREATOR CORPORATION (CHINA)  
Copyright by CREATOR

Last Revision: 09/2011