BEST++

Programmer's Reference Manual

808-893 Rev. 7/05

Manual Revisions

The *BEST++ Programmer's Reference Manual* is catalog number 808-893, Rev. 7/05. This manual replaces the preliminary *BEST++ Programmer's Reference Manual*, catalog number 808-893, Rev. 3/96.

Section/Chapter	Changes
Introduction	1. On Page 1, added the following note: It is important to note that you cannot write to Comfort Controller maintenance decisions. You can, however, read from Comfort Controller maintenance decisions without negative consequence. Also added this note to the description of CONNECT on Page 61, under Internally Connecting to a Decision, and on Page 66 under Externally Connecting to a Function's Decision in a Comfort Controller.
The BEST++ Programmer's Environment	2. In Table 1, BEST++ Terms, revised the first sentence of the description of the Decompile task so that it now reads: To reverse compile a program and produce a BEST++ program source file that you can view and edit. 3. Under File, revised the first sentence of the explanation of the Save Folder As... command so that it now reads: Use this command to save the open folder and all of the .BST programs listed in the folder under a new name.
Creating an Application	4. On Page 25, inserted new Paragraph 1, referring out to the ComfortWORKS Programmer's Environment information that is contained in Appendix D.
Reserved Words, Logical Operators and Math Functions	5. In the explanation of CONNECT, on page 58, under External CONNECTs (CONNECTs to another controller on the same CCN), under CONNECTing to a decision in a UT203 FID, 32MP Gateway, or VVT Gateway, modified the 32 MP GW and VVT GW syntax so that they each now read: 32MP GW NETWORK_DECISION *var* (*CCN_element#, CCN_bus#, 0, table#, index#*) VVT GW NETWORK_DECISION *var (CCN_element#, CCN_bus#, 0, device, decision#)* Revised corresponding explanation under BEST++ Syntax on bottom of Page 64.

Section/Chapter	Changes
Reserved Words, Logical Operators and Math Functions	6. In the explanation of COUNTER, re-wrote the second paragraph so that it now reads: The maximum number to which a counter can count is 65,535. When the counter reaches this value, it automatically resets to 0. 7. In the explanation of INPUTFROM on Page 97, revised the description of status 2. 8. In the explanation of MAX and MIN, in the Syntax and Examples, replaced all occurrences of the . (period) character with the , character (comma). 9. In the explanation of ROUNDDOWN and ROUNDUP, revised examples 1 and 2. 10. In the explanation of RUN, revised the interpretation so that it now reads: A task named CHILLER is activated at its first statement. If the task is already active, execution continues and is not affected. If the task is not active, execution begins from the first statement of the task. 11. In the explanation of TASK, revised the third paragraph on Page 138 so that it now reads: *reschpor* is the amount of time that must elapse after a download or Power On Reset (POR) occurs before the task will repeat. 12. In the explanation of TIMER, revised the third paragraph so that it now reads: A timer can count up to 4,294,967,295 seconds and then automatically resets to 9 and continues incrementing.
Debugging System	13. On Page 174, under Debug Dialog Box, revised the second paragraph so that it now reads: The left box displays a list of all local user names in the program. Use the scroll bars to view names that do not fit in the box.

Section/Chapter	Changes
Appendix A	14. On Page 197, revised the second paragraph so that it now reads: When BEST++ writes a value to an internal hardware of software point, it will write the value as dictated by the *forcepri*. When forcing variables over the CCN network, all writes will cause BEST forces, regardless of the *forcepri* setting.
Appendix D	15. Added this appendix, ComfortWORKS Programmer's Environment.

Introduction

Introduction

Operating Characteristics

Carrier's Building Environmental Systems Translator (BEST^{++}) is a custom programming language that provides you with the ability to supplement or enhance the Comfort Controller's standard control algorithms.

The Comfort Controller-resident algorithms provide the type of control necessary for most applications. The purpose of BEST^{++} is to allow you to further customize and extend the Comfort Controller's capabilities to meet any unique control requirements with little or no additional hardware.

Notes: The BEST^{++} Programmer's Environment requires a minimum of 550 Kbytes of memory to run successfully. Any TSR (Terminate and Stay Resident) programs, such as SMARTDRV, must be removed.

It is important to note that you cannot write to Comfort Controller maintenance decisions. You can, however, read from Comfort Controller maintenance decisions without negative consequence.

Because it is derived from the English language and similar to BASIC, BEST^{++} is easy to use.

Other BEST^{++} features that add to ease of use include:

- a programmer's environment that gives you the capability to select commands from a menu bar, similar to the way you do in Windows-based applications.

- a paste function that provides a foolproof way to insert reserved word, logical and mathematical operator, and symbol syntax into your program. This paste function also provides on-line help for each reserved word, logical and mathematical operator, and symbol.

- a full-screen text editor that gives you the capability to write new or edit existing BEST^{++} programs.

- two debugging features — one that displays the current status and values of variables for a specified BEST^{++} program, and another

that allows access to all items in the connected controller, including schedules, setpoints, and standard algorithms.

- a method to quickly compile and download all programs at one time to the Comfort Controller.

- the ability to use local and global variables within BEST^{++} programs. Local variables are those defined in a program for use only in that program. Global variables are those defined in the global dictionary for use by all the BEST^{++} programs in the same folder as the global dictionary.

Purpose of and Prerequisites for this Manual

The *BEST^{++} Programmer's Reference Manual* should be used as a reference tool when writing BEST^{++} programs for the Comfort Controller.

Prior to using this manual, you should attend some formal BEST^{++} training.

Note: The examples presented in this manual are for illustrative purposes only. They are not necessarily true applications of the BEST^{++} programming language. Do not attempt to implement them as shown. All examples are in customary US engineering units.

Structure and Content of this Manual

This manual is divided into the following chapters:

Introduction
The BEST^{++} Programmer's Environment
Creating an Application
Rules for Creating Names in BEST^{++}
Statements
Reserved Words, Logical Operators, and Math Functions
Mathematical and Relational Operators
Symbols
Task Execution
Debugging System
Syntax Error Messages and Report Warnings
Appendix A — Comfort Controller Force Priorities
Appendix B — Metric Units

Appendix C — Comparison of FID BEST and BEST^{++} PE Commands

The Introduction contains the information contained here as well as a description of the BEST^{++} custom programming language.

The BEST^{++} Programmer's Environment chapter provides the following instructions:

- How to access the BEST^{++} Programmer's Environment and select a command from the menu bar

The Creating an Application chapter provides the following step-by-step instructions:

- How to create, compile, and download a new BEST^{++} program
- How to edit an existing BEST^{++} program

The Rules for Creating Names in BEST^{++} chapter lists the rules for creating variable names and provides examples of valid and invalid names.

The Statements chapter introduces you to a BEST^{++} statement and provides rules for creating statements. This chapter also discusses the benefits of indenting statements and inserting blank lines. This chapter also includes BEST^{++} syntax rules.

The Reserved Words, Logical Operators, and Math Functions chapter provides the following information for each BEST^{++} reserved word, logical operator, and math function: description, syntax, examples of usage, and usage rules. For easy reference, this chapter is organized alphabetically.

The Mathematical and Relational Operators chapter provides the following information for each BEST^{++} mathematical and relational operator: description, syntax, examples of use, and usage rules. For easy reference, the operators in this chapter are organized in order of execution, with mathematical operators appearing before relational operators.

The Symbols chapter provides the following information for each BEST^{++} symbol: description, syntax, examples of use, and usage

rules. For easy reference, the symbols in this chapter are organized according to their order of execution.

The Task Execution chapter discusses single BEST^{++} task execution, multiple task execution, and BEST^{++} program interaction with Comfort Controller algorithms. This chapter also discusses the effect of cycling Comfort Controller power during BEST^{++} task execution.

The Debugging System chapter describes the menu items within the BEST^{++} Debugger and System Debugger.

The Syntax Error Messages and Report Warnings chapter lists each error code along with its meaning.

Appendix A lists, in decreasing order, the priority of forces within a Comfort Controller.

Appendix B provides instructions for specifying customary US or metric engineering units.

Appendix C lists the FID BEST Programmer's Environment commands and their comparable BEST^{++} Programmer's Environment commands.

Appendix D provides information on the ComfortWORKS' Programmer's Environment.

Terminology

The following terms are used throughout this manual.

Table 1
BEST^{++} Terms

Term	Description
Clipboard	A temporary storage location used to transfer data between places in a program. You place data on the clipboard using the Cut or Copy command, and you retrieve data from the clipboard by using the Paste command. The clipboard contains only the most recent cut or copied text.
Comfort Controller	A field installed device that is a solid-state, microcontroller-based controller. It regulates building equipment using closed-loop, direct digital control.

(continued)

Table 1
BEST ++ Terms *(continued)*

Term	Description
Compile	To translate BEST++ program syntax to machine (microcontroller) language.
Debug	To edit a program and correct any errors detected during the compile process. Additionally, the debugging feature enables you to monitor program operation within the controller to ensure the program is operating as intended.
Decompile	To reverse compile a program and produce a BEST++ program source file that you can view and edit. Additionally, the decompile process allows you to verify that the compiled program was correctly translated.
Folder	A list of all BEST++ programs that reside in a specific controller. There must be only one folder per controller. A folder allows all of the programs contained in it to share one common dictionary (the global dictionary) and gives you the capability to compile and download all programs within a folder at one time. For example, if a folder contains the names of ten programs and you want to compile and download all ten of them, you could easily compile them all at once with the Compile All command.
Global Dictionary	The program where you specify point and variable names, timers, counters, and arrays to be used by the programs in a folder. These names automatically become common to all BEST++ programs listed in the same folder as the global dictionary. You can add to the dictionary as often as necessary.

(continued)

Table 1
BEST ++ Terms *(continued)*

Term	Description
	Note: You must initialize the Comfort Controller after making a change to the global dictionary. For information on initializing, refer to the Debugging System chapter in this manual. When you create a new folder, BEST++ will automatically create a .BST file with the same name as the folder and insert the PROGRAM statement with the description "Global Dictionary." **Note:** You must keep the global dictionary as the first program in the folder, and you may not change its name.
Menu Bar	A menu displayed across the top of a screen or dialog box. To access the menu bar, press the Alt key. Use the right or left arrow key to highlight the desired menu item and press Enter to display that menu item's command list. Use the up or down arrow key to select the command.

The BEST++ Programmer's Environment

The BEST++ Programmer's Environment

About this Chapter

This chapter provides the following information about BEST++ programs:

- General information
- File structure
- Writing
- Accessing the BEST++ Programmer's Environment (PE) and selecting a command from the menu bar

This chapter also describes each of the commands that appear in the BEST++ menu bar.

You can choose menu items and commands in the Programmer's Environment using the mouse or the keyboard. The instructions in this manual are written with the assumption that you are not using a mouse, and details the keystrokes required to use BEST++ with a keyboard.

Note: If you intend to use a mouse with the Network Service Tool, you must ensure that you have loaded the mouse driver software that was provided with the mouse.

You can use the mouse to move the cursor around the screen and select menu bar items, commands in menus, or text by clicking once on the left mouse button.

Using the Keyboard

The table below lists the Network Service Tool's keys that you can use to move around in BEST^{++}. Keys that must be pressed at the same time are separated with a plus sign (+).

Table 2
Network Service Tool's BEST^{++} Keys

Press	To
Alt	access the menu bar.
Ctrl + C	copy highlighted text.
Ctrl + O	open a program.
Ctrl + S	save a program.
Ctrl + P	print a list of all the programs specified in a folder.
Ctrl + X	cut highlighted text.
Ctrl + V	paste highlighted text.
Enter	select what is highlighted.
Esc	close a menu or dialog box without making a selection or saving changes.
Shift + Tab	move to the previous text box, button, or option in a dialog box.
Shift + arrow key	highlight text you wish to edit.
Tab	move to the next text box, button, or option in a dialog box.
Up, Down Arrows	move up and down in lists or to move the cursor within a program.
Left, Right Arrows	move between menu items on the menu bar or to move the cursor within a program.
F3	access the Find dialog box. Use this dialog box to enter the text string to be searched.
Shift + F3	access the Find and Change dialog box.
F8	Activate BEST^{++} Debug.
F9	Activate System Debug.

Where You Write BEST++ Programs

You write, edit, compile, and debug BEST++ programs using the BEST++ Programmer's Environment. When you select the BEST++ function, you enter the Programmer's Environment. A Comfort Controller is not necessary to write programs or correct syntax errors while in the Programmer's Environment.

After your programs are successfully compiled and debugged, you download the program to the targeted Comfort Controller. The program will execute and continue to run even if you exit the Programmer's Environment or disconnect from the Comfort Controller.

BEST++ File Structure

Folders: A folder consists of a list of .BST files. These files include a global dictionary and associated local programs. The global dictionary should contain CONNECT, TIMER, and ARRAY statements. CONNECTs or other definitions in the global dictionary can be accessed by all of the programs in the folder. Any CONNECTs defined in the local programs cannot be seen by the global dictionary or any other local program. They are strictly for use in the program in which they are defined. You save all folders (.FLD file extension) on the Network Service Tool's hard drive in a directory of your choice.

Programs: The programs contain your BEST++ code, which can be downloaded to a Comfort Controller when compiled. BEST++ automatically saves the compiled program (.BPP file extension) in the same directory as the source program. You do not have to save all programs in the same directory as the folder. However, the Global Dictionary must be in the same directory as the folder.

File Extensions

BEST++ files are named using the file extensions listed and described in the table below.

Table 3
File Extensions

File Extension	Description
.BST	A source program, a file that contains the uncompiled version of your program.
.BPP	A compiled program, a file that contains the compiled version of the program ready to download.

(continued)

Table 3
File Extensions
(continued)

File Extension	Description
.FLD	A folder, a file that contains the name and location of the global dictionary and all associated programs.
.LST	A list file, generated by the Make List File command after a program is compiled. This file is a listing of the program with line numbers and syntax error messages that were detected during the BEST^{++} program compile. This list file may also contain warnings that are generated when BEST^{++} corrects certain errors detected within a program. **Caution:** Do not edit the list file because the changes will not be saved to the source file. Make all corrections in the source file.

How to Access the Programmer's Environment from the Network Service Tool and Select a Command

Follow the steps below to access the BEST^{++} Programmer's Environment from the Network Service Tool's menu and then select the PE menu bar.

1. Press *8* from the Network Service Tool's Main Menu to access the Comfort Controller BEST^{++} Programmer's Environment (PE). The PE menu bar consists of the following items: File Edit BEST Options Help

2. Press the Alt key to access the PE menu bar or use the mouse to select a menu item.

3. Type the first letter of the desired menu item to access it, or use the right or left arrow key to access an item.

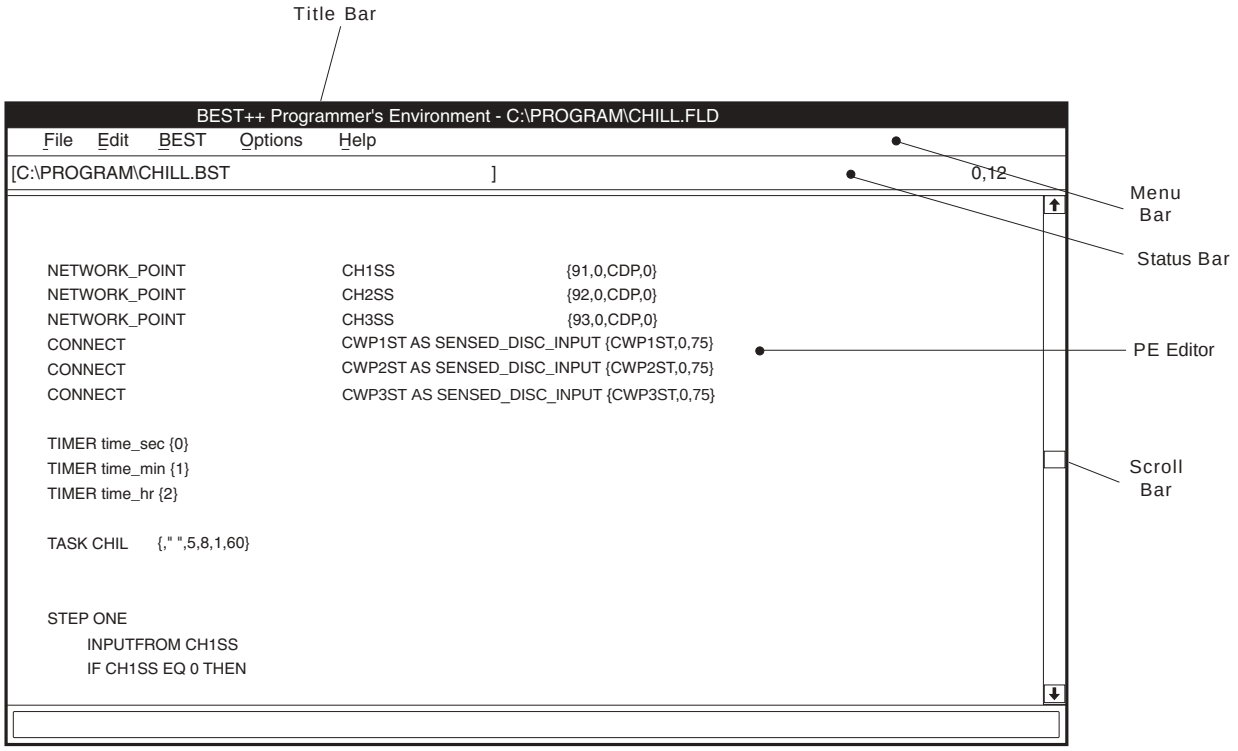
4. Type the highlighted letter of the command you wish to access, or use the up or down arrow key to highlight a command from the menu item's list of commands. Then press Enter to select the command.

   To exit a menu without making a selection, press the Escape key.

The Programmer's Environment Window

The previous section How to Access the Programmer's Environment from the Network Service Tool and Select a Command described how to display the PE window and how to select a command from the menu bar. This section describes the PE window layout and the menu bar commands in more detail. The figure below shows the PE window.

Figure 1
Programmer's Environment Window



Title Bar	Contains the product name *BEST*$^{++}$ *Programmer's Environment* and identifies the folder that is currently open.
Menu Bar	Contains the PE menu items File, Edit, BEST, Options, and Help used to write and debug BEST^{++} application programs.
Status Bar	Provides user feedback when commands are being executed. It is subdivided from left to right into three sections: The first section is a pull down menu that allows you to select any program in the folder currently open. When selected, the program is displayed on the screen. The second section identifies the status of an action in process, such as reading or writing to a target device. The third section on the far right displays the target device address.
PE Editor	The workspace for editing BEST^{++} programs.
Scroll Bar	If all of the program being edited cannot be displayed at once, clicking below the box in the scroll bar enables you to move down.

Programmer's Environment Menu Items

File

Command	Use this command to
New Folder	create a new folder. When selected, this command displays the New Folder dialog box, which contains a list of the existing folders in the current directory. Refer to File Dialog Box.
Open Folder	open an existing folder. When selected, this command displays the Open Folder dialog box, which contains a list of existing folders in the current directory. Refer to File Dialog Box.
Save Folder	save the open folder. When selected, this command displays the Save As dialog box if the folder was not already saved. Refer to File Dialog Box.
Save Folder As ...	save the open folder and all of the .BST programs listed in the folder under a new name. The saved folder is given the Save As name. When selected, this command displays the Save As dialog box. When you save a folder using this command, all programs in the folder will be copied to the desired path. Refer to File Dialog Box.
Delete Folder	delete a selected folder. This command deletes the folder only; it does

(continued)

File *(continued)*

Command	Use this command to
	not delete any programs listed in the folder. When selected, this command displays the Delete dialog box. Refer to File Dialog Box. **Note:** When deleting a folder, you must be in another folder.
New Program	create a new program. When selected, this command displays the New Task dialog box. This dialog box lists the existing programs and allows you to name the new program. Refer to File Dialog Box.
Open Program	open an existing program to view or edit. If the program is not part of the open folder, you will be prompted to add it to the open folder.
Save Program	save the program in the PE editor window.
Save Program As ...	copy the open program and save it as a different name or to a different directory. When selected, this command displays the Save As dialog box. Refer to File Dialog Box.
Remove Program	remove the open program from the currently open folder. A confirmation dialog box will appear. Press Enter to remove the program from the open folder. This process only removes a program name from the open folder's program list. It does *not* delete the program.

(continued)

File *(continued)*

Command	Use this command to
Delete Program	delete a selected program. When selected, this command displays the Delete dialog box, which contains a list of the existing programs. Refer to File Dialog Box.
Print All	print all programs listed in the opened folder. This command prints the actual programs, not just a list of program names.
Print Program	print the program in the Programmer's Environment editor window.
Exit	exit the Best^{++} Programmer's Environment.

File Dialog Box

The File dialog box allows you to define a folder or program name and its storage or retrieval location.

This dialog box is displayed whenever you access a folder or program. It is also activated when you perform a Save As. Use the Tab key to move within the dialog box.

File Name:	Enter a unique name (up to 8 alphanumeric characters, with first 7 characters unique) of a program or folder to save or retrieve. You must maintain the file name extensions provided by BEST^{++}.
File Name list	Select a name from the list of existing programs or folders in the selected drive and directory. The name that you select will appear in the file name entry box.

(continued)

File Dialog Box
(continued)

Drive	Displays the current drive for storing or retrieving files. When you use the pull down menu to select a new drive, the directory list is updated.
Directory	Displays the working directory on the current drive. When you select a new directory, the file name list is updated.
OK	Confirms selections and exits the dialog box.
Cancel	Exits the dialog box and restores the previous settings. Any changes made are not saved.

Edit

You must highlight the text that you wish to cut, copy, paste, or delete.

Note: If using the keyboard to cut, copy, paste, and delete text, use the arrow key(s) to move your cursor to the beginning of the text you wish to highlight. Highlight the text by holding down the Shift key and pressing the appropriate arrow key.

If you are using a mouse to cut, copy, paste, and delete text, place your cursor at the beginning of the text you wish to highlight and hold down the mouse button. Drag the cursor until all of the text you want to affect is highlighted.

Command	Use this command to
Cut	remove the highlighted text. Shortcut key: Control + X. The removed text is saved to the clipboard for future use with the Paste command. The clipboard contains only the most recently cut or copied text.

(continued)

Edit *(continued)*

Command	Use this command to
Copy	copy the highlighted text and save it to the clipboard. Shortcut key: Control + C.
Paste	insert previously cut or copied text from the clipboard into your program at the current cursor location. Shortcut key: Control + V.
Delete	permanently delete the highlighted text. Shortcut key: F7. Deleted text is *not* stored on the clipboard, and therefore cannot be retrieved using the Copy or Paste commands.
Find	locate a designated text string. Shortcut key: F3. **Dialog Box Options:** *Find What:* Enter the text string you wish to find. *Find Next:* Press Enter to locate the designated text string in the program you are editing. *Cancel:* Exit the Find dialog box without executing the Find command.
Replace	replace one text string with another. Shortcut key: Shift + F3 **Dialog Box Options:** *Find What:* Enter the text string you wish to find and change.

(continued)

Edit *(continued)*

Command	Use this command to
	Change To: Enter the text that will replace the Find What text string. *Change Next:* Select to locate and replace the first instance of the Find What text string. *Change All:* Select to change all instances of the Find What text string. *Cancel:* Select to exit the Change dialog box without executing a change command.
Paste Function	insert the text of the selected keyword into your program where the cursor is currently located. Refer to Paste Function Dialog Box below. allows you to view the syntax, identify configuration and maintenance parameters, and get on-line help for each keyword. By pasting the syntax into a program at the cursor location, you can use paste function as an alternate way of entering text in a program. The reserved words are grouped according to functionality. Select Paste Function to display the list of reserved word groups as a submenu, and use the up or down arrow to highlight the group in which you are interested. Press Enter again to display the reserved words in the selected group.

(continued)

Group	Displays a List of
All	all supported reserved words, mathematical, relational and logical operators, and program control
Operators	all supported mathematical, relational, and logical operators
Math Functions	mathematical functions
Program Control	reserved words used to control flow of program logic

Paste Dialog Box Command Keys

The top window where the cursor is located allows you to enter the first character of the reserved word or symbol.

The center window contains the reserved words in alphabetical order. A scroll bar is provided for quick scanning.

The bottom window contains the syntax, functions or help information for the highlighted keyword, depending on the command key selected:

Syntax	Display the syntax for the selected reserved word.
Funcs	Display the configuration and maintenance parameters for the selected reserved word.
Help	Display the syntax and on-line help for the selected reserved word.

(continued)

Paste Dialog Box Command Keys *(continued)*

Paste	Paste the syntax for the selected reserved word into the program you are editing at the current cursor location.
Exit	Close the Paste Function dialog box.

BEST

Command	Use this command to
Set Controller Address	designate the address (CCN bus and system element number) of the target controller. When you select this command, the Set Controller Address dialog box displays. After you enter and confirm the address, it displays in the status bar. **Dialog Box Options:** *Bus #:* Carrier Comfort Network bus number for the desired Comfort Controller. Valid numbers range from 0 to 239. *Element #:* Carrier Comfort Network element number for the desired Comfort Controller. Valid numbers range from 0 to 239. *OK:* To save the information and exit from the Set Controller Address dialog box.
Compile Program	compile the program that is open in the Programmer's Environment editor. When you select this command, the Compiled Code dialog box lists the results of the compile process, including any syntax errors detected.

(continued)

Command	Use this command to
Compiled Code Box Options:	
	Print: Select to print the information in the Compiled Code box. *Exit:* Select to close the Compiled Code box.
Compile All	compile all of the programs contained in the open folder in the order listed. When you select this command, the Compiled Code dialog box displays the results of each compile process. **Compiled Code Box Options:** *Print:* Select to print the information in the Compiled Code box. *Exit:* Select to close the Compiled Code box.
Decompile Program	decompiles the program that is open in the Programmer's Environment editor. You can use this as a debugging tool to evaluate how the compiler interpreted your program. Selecting this command displays a decompiled version of the program.
Make List File	creates a decompiled version of the currently open program that includes line numbers. You can use this command when debugging your program. Refer to the Debugging System chapter of this manual.

(continued)

Command	Use this command to
	Listing Text Box Options:
	Print: Select to print the Listing text box.
	Exit: Select to close the Listing text box.
Show Compiler Window	redisplay the Compiler Status window, which you can hide by clicking the left mouse button when the mouse is pointing outside the window.
Download Program	download the last compiled version of the program open in the PE editor. You should always compile a program before you download it, or you may download a previously compiled version that does not contain your most recent changes.
Download All	download all of the programs in the open folder. Programs are downloaded in the order listed in the folder. You should compile all programs before you download them.
	Dialog Box Options:
	Yes: Select to confirm correct target comfort controller.
	No: Select to return to the main screen without downloading the programs in the currently displayed folder.
BEST^{++} Debug	debug programs. Refer to the Debugging System chapter.
System Debug	debug Comfort Controllers. Refer to the Debugging System chapter.

Options

Command	Use this command to
Color Display	when selected, set up the Programmer's Environment to use a color monitor. Otherwise, it assumes a black and white monitor.
Report Warnings	when selected, display warnings along with syntax error messages. BEST^{++} generates warnings when it automatically makes assumptions within a program, i.e., replacing = with *EQ* in IF... THEN... ENDIF statements. If you have requested a list file of errors using *Make List File?*, BEST^{++} saves the warnings to the .LST file along with the errors. Refer to the Syntax Error Messages chapter for a list of the warnings and error messages that BEST^{++} can display. **Note:** A program containing warnings will compile and download correctly. However, the program could operate improperly.
Metric	when selected, display configuration and maintenance data in metric engineering units instead of customary US engineering units. Also displays metric equivalents for Units Name. Refer to the Debugging System chapter of this manual.
Delete Template After Download	when selected, delete a source file's template after downloading the compiled version to the controller. This prevents the program from being uploaded and decompiled.

(continued)

Options *(continued)*

Command	Use this command to
	A template is an image of the source file that provides the capability to upload the file from the controller back to the Network Service Tool. **Caution:** A downloaded file whose template has been deleted cannot be uploaded to the Network Service Tool, nor can it be edited. If you plan to upload and edit files, do not delete the templates after downloading.

Help

Command	Use this command to
Keywords	access the help for reserved words. This help is the same help displayed when you select Paste Function from the Edit menu and then select *Program Control*.
Advanced Features	This command is the same as Edit/Paste Function/All.
Units Name	display the allowable entries for analog input and analog output display units for use with constants.
About . . .	display copyright information and currently installed Programmer's Environment and database version numbers.

Creating an Application

Creating an Application

The step-by-step instructions in this section are written with the assumption that you are using the Network Service Tool's Programmer's Environment. If you are using the ComfortWORKS' BEST^{++} Programmer's Environment, refer to Appendix D.

Before writing any BEST^{++} programs, you should decide how to structure your application. If it will require multiple programs that share data, you will need a global dictionary that defines these common parameters.

A folder contains a list of all the programs composing an application, the global dictionary, and their physical locations. Every application, even those having only one program, should be contained in a folder. The folder is not compiled or downloaded to the Comfort Controller, so it uses no additional space.

How to Create or Open a Folder

When you open a new folder, the Programmer's Environment creates a global dictionary program with the same name. Follow the steps below to create or open a folder.

1. Select *File* from the PE menu bar.
2. Select either *New Folder* or *Open Folder*, depending on whether you want to save the program in an existing or new folder, and press Enter.

   Depending on your selection, either the Open or New Folder dialog box will display.
3. Enter a unique folder name using a .FLD extension and a directory or, if saving the new program to an existing folder, select an existing folder name and directory from the list. Then select *OK*.

   Note: The folder name and each program name can be up to 8 characters long, but the first 7 characters must be unique.

   BEST^{++} will automatically open a program and assign to it the same up to 8 character name you assigned to the folder. This program, which BEST^{++} automatically opens for you, is the global dictionary. BEST^{++} also automatically inserts the PROGRAM statement.

How to Add a New BEST++ Program to a Folder

You write BEST++ programs as a sequence of statements. A statement is a combination of BEST++ reserved words, mathematical functions, operators, symbols, variables, and constants.
You can write BEST++ programs using either customary US or metric engineering units. For information on using metric engineering units, refer to Appendix B.

Follow the steps below to add a new program to your folder.

1. Select *File* from the PE menu bar.
2. Select *New Program* from the File menu.
3. Enter a unique name for the new program. You can change the directory and drive where the program will be saved. Then select *OK*. The program name must be different from the folder name and must have a .BST extension.

   The PE will automatically insert the PROGRAM statement and add the program to your folder.
4. Enter the TASK statement, followed by your task. End the task with an ENDTASK statement.

   A sample program is shown below.

Figure 2
Sample Program

```
PROGRAM sample {SAMPLE, "Custom Program sample"}

~ This program is just a sample.

TASK SAMPLE

x = y + z
c = a + b

ENDTASK
```

How to Add an Existing BEST++ Program to a Folder

Follow the steps below to add an existing program to your folder.

1. Select *File* from the PE menu bar.
2. Select *Open Program* from the File menu.
3. Select the program to be added from the list displayed. You can change the directory and drive if your program is not in the current directory.

4. Select *OK*. The program opens and a dialog box prompts, *Add Task to Task List?*. Answering *Yes* adds the selected program to the folder.

How to Compile a BEST++ Program

Follow the instructions below to compile a BEST++ program.

1. Select *Compile Program* from the BEST menu.

   When the compiling is complete, BEST++ displays the Compile Code window. This window displays information such as the number and type of syntax errors, if any, and the program size. Refer to the figure below which illustrates what the Compile Code window will display for the program shown in Figure 2.

   For a list of syntax error messages and their code numbers, refer to the Syntax Error Messages chapter. If errors are not detected, the number of errors will equal *0*.

Figure 3
Compile Code Window

```
I:\PROGRAM\SAMPLE.BST
Objects Generated      10
Object File (.BPP) written
Download E2ROM Size        =      181
  Program E2ROM Size       =      222
      Total E2ROM Size     =      403
      Program RAM Size     =      64

Compilation Complete - Number of Errors  0
```

2. Press the Alt key to activate the Compile Code menu bar. Then select either *Print* or *Exit*.

3. If no errors were detected during the compile, you can download the program to a controller.

   If errors were detected during the compile process, select *Make List File* from the BEST menu to create a list file that displays the program and allows you to see exactly where the syntax error occurred. If necessary, you can expand the window to full screen by clicking on the up arrow. Refer to the figure below, which illustrates a program containing a syntax error message and the corresponding list file.

Caution: Do not edit the list file because the changes will not be saved to the source file. Make all corrections in the source file.

Figure 4
Sample Program, Corresponding List File, and Corrective Action

Sample Program

```
PROGRAM sample {SAMPLE, “Custom Program sample”}

~ This program is missing the TASK statement.

x = y + z
c = a + b

ENDTASK
```

List File

```
BEST++ - Program Listing I:\PROGRAM\SAMPLE.BST
SE2PROM Size      =  181
E2PROM Size       =  222
          RAM Size =  64

1 Errors Detected

01    PROGRAM sample {SAMPLE, “Custom Program sample”}
02
03    ~ This program is missing the TASK statement.
04
05
06
07    x = y + z
              *** Line 7, Error 32 (=), TASK missing
08    c = y + z
09
10
11    ENDTASK

Symbol Table
line #
07    VARIABLE    x
07    VARIABLE    y
07    VARIABLE    z
08    VARIABLE    c
```

Figure 4
Sample Program, Corresponding List File, and Corrective Action (continued)

Action
Press the Alt key to activate the menu bar. Then select either *Print* or *Exit*. Fix the errors in your source (.BST) program and re-compile.

How to Download a BEST++ Program

When you open a program, the last compiled version of that source file is available to download. If you changed the program, you must compile it before downloading. When you download a compiled program to a Comfort Controller, the compiled program is copied to that Comfort Controller.

Follow the instructions below to download a BEST++ program.

1. Verify that the correct address for the target controller appears in the upper right corner of the screen, just below the menu bar. If it is correct, proceed to Step 2.

   If you need to change the address, select *Set Controller Address* from the BEST menu. Then specify the bus number and system element number of the controller to which you want to download the program. Then press *OK*. The new address appears on the screen.

2. Select *Download Program* from the BEST menu. Depending on whether the download is successful, BEST++ will display *Download Complete* or *Download Failed*.

   If the download failed, perform the following troubleshooting measures:

   - Verify that the CCN Communication Bus is connected correctly.
   - Verify that you can communicate with the controller from Carrier Controls.
   - Select *Database Control (UPDATEDB)* from *System Debug*. Verify that there are no errors and that enough memory is available for your program. Refer to the Debugging System chapter of this manual.

How to Edit a BEST++ Program in a Connected Controller

Follow the instructions below to edit a BEST++ program (.BST file) that resides only in a connected controller.

1. Upload the program by selecting *System Debug* from the BEST menu to upload the list of programs from the selected controller. Then select *BEST++ Sources* and select the program you want to view. Press the Alt key to activate the menu, then select *Upload*. Press the Alt key again, and select *Exit*.

   Note: BEST++ automatically decompiles the program as it is uploaded.

2. Modify, save, compile, and download the program.

   Note: The program will not contain remarks. These are saved only in the original source program.

Rules for Creating Names in BEST^{++}

Rules for Creating Names in BEST++

Follow the rules below when creating names for the following items in BEST++:

- Tasks
- Variables
- Steps
- Labels
- Timers
- Counters
- Arrays

Rule 1 **Names must consist of consecutive alphanumeric characters with no spaces, mathematical, relational, or logical operators.**

Invalid
CHLR 1
SFS*1

Valid
CHLR1
SFS_1
SUPPLY_FAN_STAT_1

Suggestion: Use the underscore character (_) to represent a space.

Rule 2 **Letters, numbers, and the following special symbols are acceptable:**

@	#
$	%

_ (underscore)

Rule 3 **The first character of a name can be a letter, number, or any of the special symbols listed in Rule 2.**

Note: A name cannot consist of only numbers. There must be at least one character that is not a number.

Valid
$CHILLER
CHILLER7
7CHILLER

Invalid
763

Rule 4 **Names can be in either uppercase or lowercase. However, BEST++ is case sensitive. Typing the same name inconsistently will cause it to be interpreted as two different variables.**

Invalid

```
VARIABLE FAN
IF TEMP1 > 70 THEN TURNON Fan ENDIF

x = 2y + 3
IF X > 20 THEN GOTO 5 ENDIF
```

Valid

```
VARIABLE fan
IF TEMP1 > 70 THEN TURNON fan ENDIF

X = 2y + 3
IF X > 20 THEN GOTO 5 ENDIF
```

Rule 5 **The following items cannot be used as variable names:**

- **Reserved words**
- **Names of tasks, arrays, steps, labels, timers, and counters**
- **Math functions**
- **Logical, relational, and mathematical operators**

Invalid

```
MONTH = x
```

Valid

```
x =  MONTH
```

Rule 6 **Within a Comfort Controller, you should create a unique name for each variable. If you want a CONNECTed variable to represent the same constant, value, or decision in more than one program, you must do these two things:**

1. **Define the variable in the global dictionary.**
2. **Be consistent when typing the name in each program, i.e., be case sensitive. Type the name exactly as it appears in the global dictionary.**

Note: Using the same name to represent different constants, values, or decisions will produce the following error message during program compilation: *duplicate user name*.

Invalid

```
CONNECT TEMP1 AS TEMP_INPUT {,1}
CONNECT TEMP1 AS TEMP_INPUT {,2}
```

Valid

```
CONNECT TEMP1 AS TEMP_INPUT {,1}
CONNECT TEMP2 AS TEMP_INPUT {,2}
```

Rule 7 **Create variables that clearly identify the values they are representing.**

Poor

A3 to represent pump #1
ABC to represent a cooling coil valve

Better

PUMP1 to represent pump #1
CCV to represent a cooling coil valve

Rule 8 **To avoid possible confusion when assigning names to points, use the same point names that are used in the Network Service Tool.**

	Network Service Tool	BEST++
Poor	SFS HCV	SUPFANST HTGCV
Better	SFS HCV	SFS HCV

Rule 9 **Unlike BEST, when you load a BEST++ program the values of all unCONNECTed variables are initialized to (start out at) *0*, unless you explicitly initialize the variable to a particular value using { }.**

Example VARIABLE TEMP1

Interpretation: When this line is encountered, BEST++ will initialize TEMP1 to *0*.

Example VARIABLE TEMP1 {6}

Interpretation: When this line is encountered, BEST++ will initialize TEMP1 to *6*. The curly brackets *{}* indicate initialization.

Statements

Statements

Purpose

A BEST^{++} program is made up of statements that define and control a program. A statement is a combination of reserved words, mathematical functions, operators, symbols, variables, and constants.

BEST^{++} statements enable you to:

- define the beginning and end of a TASK.
- control the flow of the task with STEPs, DELAY statements, and GOTO statements.
- pass control to and from standard Comfort Controller algorithms.
- RUN and HALT other tasks.
- give reference names to points.
- read and write to points and configuration data.
- identify the beginning and end of LOOPs within the task.
- create and control timers and counters.
- insert helpful comments within the program.
- use a variety of built-in programming functions.
- use algebraic and logical expressions.

Examples of Statements

Nine statements are shown in the sample task below:

Figure 5
Statements in a Sample Program

```
PROGRAM CHILLERS {chillers, East Plant,} ~ Statement 1

TASK  CHILLER1                           ~ Statement 2

:X                                       ~ Statement 3

VAR1 = TEMP1 + TEMP2                     ~ Statement 4

IF TASKTIMER > 3600                      ~ Statement 5
        THEN HALT CHILLER
                ENDIF

DELAY CHILLER1 (10)                      ~ Statement 6

SQRT  (3)                                ~ Statement 7

GOTO  X                                  ~ Statement 8

ENDTASK                                  ~ Statement 9
```

Creating Statements

Statements are entered in the Programmers Environment in one of two ways:

- Typing it in manually, pressing Enter when finished. A tilde (~) is not required to continue a statement onto the next line.
- Inserting the syntax for all reserved words, operators, and math functions into your task by using the Paste Function command.

Separating Statements

Multiple BEST^{++} statements can be written on a single program line. The statements can be separated by one or more spaces or by a semi-colon (;).

Example

IF OAT > 70 THEN TURNON FAN; COUNT = COUNT + 1 ENDIF

Interpretation: In the above program line, if OAT is greater than 70, FAN will be turned on, and the counter will be incremented by 1. If OAT is less than or equal to 70, FAN will not be turned on and the counter will not be incremented by 1. The semicolon indicates to BEST^{++} to treat the two action statements as one.

Use of Indentation and Blank Lines

For the purpose of readability and organization, especially if the program is rather large, it is good practice to use indentation and multiple lines to type a statement.

Caution: The maximum number of lines in a BEST++ program is 600. In the examples shown below, Example 1 uses one program line. Example 2 uses two lines, and Example 3 uses three lines.

BEST++ interprets the following three statements in the same way:

Example 1

```
IF ALARM EQ 1 THEN GOTO FOUR
```

Example 2

```
IF ALARM EQ 1
        THEN GOTO FOUR
```

Example 3

```
IF ALARM EQ 1
        THEN
                GOTO FOUR
```

Example 4

This is an example of a program typed using indents and blank lines:

```
                                TASK MSI

~REM    BUS   0         CTRLR   1
~REM    2/11/96         THIS PROGRAM LOOKS AT THE
~REM                    STATUS OF TWO PUMPS AND
~REM                    TURNS A THIRD STATUS POINT
~REM                    ON IF EITHER PUMP IS ON
~REM                    AND TURNS IT OFF
~REM                    IF BOTH PUMPS ARE OFF.
~REM                    THE STATUS OF EACH POINT IS
~REM                    CHECKED EVERY 5 SECONDS.

STEP ONE
IF PMPS1 EQ 1 OR PMPS2 EQ 1
   THEN TURNON PMPS
ENDIF
IF PMPS1 EQ 0 AND PMPS2 EQ 0
   THEN TURNOFF PMPS
ENDIF
DELAY MSI (5)
REPEAT

ENDTASK
```

Syntax Rules

The following are syntax rules that you must adhere to when creating BEST++ statements. In most cases, if these rules are broken, a syntax error message appears on the screen when the program is compiled.

Note: BEST++ does not generate error messages for logical errors.

Uppercase vs. Lowercase

Rule 1

In BEST++ statements, reserved words, logical operators, and functions must always be uppercase. A syntax error results if you do otherwise.

Any part of a statement, other than those specified above, can be either uppercase or lowercase. However, we recommend that you always use upper case.

Invalid

```
if OAT > 70 and FAN eq 0 then turnon FAN endif
```

Valid

```
IF oat > 70 AND fan EQ 0 THEN TURNON fan ENDIF
IF OAT > 70 AND FAN EQ 0 THEN TURNON FAN ENDIF
```

Rule 2

You must be consistent in spelling and in the use of capitalization when creating and referencing variables. For example Fan, fan, and FAN would be interpreted as three different variables.

Invalid

```
IF OAT > 70 AND FAN = 0 THEN TURNON fan.
```

Valid

```
IF OAT > 70 AND FAN = 0 THEN TURNON FAN.
```

Note: In this manual, statements are typed in uppercase.

Use of Spaces

For readability purposes you can use spaces almost anywhere in a statement. In certain instances, spaces must be used as listed in the rules below.

Rule 3 **No spaces are needed after the tilde (~), if it is used to begin a remark.**

Valid ~REM This program determines the average
~REM temperature of four sensors.

Valid ~This program determines the average temperature of
~four sensors.

Rule 4 **At least one space must appear before and after reserved words and logical operators.**

Invalid WHENTEMP1>70TURNONFAN1

Valid WHEN TEMP1>70 TURNON FAN1

Guideline **To enhance the readability of a program, you can place space(s) before and after mathematical or relational operators. Including these spaces however, is not necessary; doing so only enhances the appearance of the program.**

Valid x=temp1+temp2 + temp3/ 3

Use of Commas

Rule 5 **Commas must be used in an initialization string to separate initialization values.**

Valid TASK mytask {MYTASK, "My task", 5, 8, 1, 6, 0}

Rule 6 **Commas can be used in initialization strings as placeholders for default values.**

Valid TASK mytask {, , , , , 6, 0}

Rule 7 **Numbers must be typed without commas.**

Invalid x = 1,000

Valid x = 1000

Use of Parentheses

Rule 8 **Parentheses must be used in pairs. In other words, a left parenthesis must accompany a right one, and vice versa.**

Parentheses can be used to define the order of execution in a mathematical expression.

Invalid D = A + B) - C

Valid D = (A + B) - C

Interpretation: BEST^{++} first executes what is inside the parentheses. A and B are added and then C is subtracted from the sum. The variable D is assigned the value of the difference.

Rule 9 **If parentheses are nested, the operation within the innermost pair is executed first.**

Valid D = ((A + B) * C)

Interpretation: BEST^{++} first executes the operation within the innermost parentheses. First A and B are added, then the sum is multiplied by C. The variable D is assigned the value of the product.

Rule 10 **Multiple non-nested parentheses in a statement are evaluated from left to right.**

Valid E = (A + B) / (C - D)

Interpretation: A and B are added before D is subtracted from C because BEST^{++} first executes the operation within the leftmost set of parentheses. The sum of A and B is then divided by the difference of D from C. The result is assigned to the variable E.

Rule 11 **In a mathematical expression, a mathematical operator must separate two sets of parentheses that are side by side. For instance, BEST^{++} does not assume that you intend to multiply the two expressions. BEST^{++} does not give you an error message if you omit the mathematical operator, but an incorrect value will be calculated.**

Invalid X = (A - B) (C * D)

Valid X = (A - B) * (C * D)

Interpretation: The result of A minus B is multiplied by the product of C and D. The resulting value is assigned to the variable X.

Reserved Words, Logical Operators, and Math Functions

Reserved Words, Logical Operators, and Math Functions

About this Chapter

This chapter discusses each BEST++ reserved word, logical operator, and math function. For easy reference, they are listed alphabetically. The information for each reserved word, operator, and math function includes a description, syntax, examples of use, and usage rules.

Reserved Words

Reserved words, logical operators, and mathematical functions are part of the BEST++ language and are used to make up BEST++ statements. They cannot be used as variable, task, step, label, array, counter, or timer names. The various types of reserved words follow.

- **definition**

  ANALOG_MAINTENANCE
  ARRAY
  CONNECT
  COUNTER
  DIM
  FLOAT_CONFIGURATION
  TIMER
  VARIABLE

- **input/output**

  AUTO
  INPUTFROM
  OUTPUTTO
  RELEASE
  TURNOFF
  TURNON
  UPDATE

- **task control**

  CALL
  DELAY
  ENDTASK
  EXIT
  GOTO
  HALT
  IF... THEN... ELSE... ENDIF...
  LOOP... ENDLOOP
  PROGRAM
  REPEAT
  RETURN
  RUN
  STEP
  SUBROUTINE
  TASK
  WHEN

- **calendar/clock**

  DOM
  DOW
  DOY
  HOUR
  MINUTE
  MONTH
  SECOND

- **counter control**

  DECREMENT
  INCREMENT
  RESET

- **timer control**

  RESET
  START
  STOP

- **general purpose**

  REM

Logical Operators

Logical operators are used to test two or more logical conditions or to generate a logical (TRUE/FALSE) result. Logical operators cannot be used as variable, task, step, label, array, counter, or timer names. They are listed below in order of precedence of execution in a statement.

NOT
AND
OR
XOR

Along with a detailed description of each operator, truth tables for the logical operators are provided in this chapter.

Math Functions

Math functions are routines that calculate a value. Math functions cannot be used as variable, task, step, label, array, timer, or counter names. The BEST^{++} math functions are:

ABS
ACOS
ASIN
ATAN
COS
COSH
EXP
FRACTION
LOG
LOG10
MIN
MAX
PID
POWER
REMAIN
ROUNDDOWN
ROUNDUP
SIN
SINH
SQRT
SWITCH
TAN
TANH

ABS

When used in a BEST^{++} statement, this function provides the absolute value of the constant, variable, or mathematical expression following it.

Syntax

ABS *x* or ABS (*x*)

where *x* is a constant, variable, or expression.

Example 1

Y = ABS -6

Interpretation: The absolute value of the -6 is calculated and assigned to the variable Y.

Example 2

TVAL = ABS (TEMP1 - TEMP2)

Interpretation: The variable TVAL is assigned the absolute value of the difference of TEMP1 and TEMP2.

Usage Rules

1. ABS cannot appear on the left side of the equal sign in an assignment statement.

   For example, ABS X = Y is invalid.

   ABS may, however, be used on the left side of an equal sign used as a relational operator.

   For example: IF ABS (X) EQ 10 THEN GOTO THREE ENDIF

2. A constant, variable, or mathematical expression must follow ABS.

3. If the variable is part of a mathematical expression, the use of parentheses ensures the proper order of evaluation of the terms. For example, TVAL = ABS TEMP1 - TEMP2 differs from the example with parentheses shown above. Without parentheses, the value of TEMP2 is subtracted from the absolute value of TEMP1, and the result is assigned to the variable TVAL.

ACOS

When used in a BEST^{++} statement, the ACOS (arc cosine) function provides the inverse cosine of an angle. The input (*x*) is the ratio of an adjacent side of a right angle triangle and its hypotenuse. The output (result of ACOS *x*) is the angle subtended by the sides expressed in radians. One radian equals $^{360}/_{(2\pi)}$ or 57.29577 degrees.

Syntax

ACOS *x*

where *x* is a constant, variable, or mathematical expression ranging from -1 to 1. Invalid statements return a value of *0*. For example, ACOS 5 is invalid.

Example

PI = ACOS (-1)

Interpretation: When BEST^{++} encounters this line in the task, it calculates the value of PI as 3.141593.

ANALOG_ MAINTENANCE

This reserved word is used in a BEST^{++} statement to define maintenance variables. Maintenance variables give you the ability to display BEST^{++} variable or statement values on the maintenance screen without having to create software points.

Syntax

ANALOG_MAINTENANCE *maintname* (*x*) {*name, "description", units*}

where *maintname* is a unique BEST^{++} user-configurable name for the maintenance decision

x is the BEST^{++} variable or statement value that will be displayed on the maintenance screen

name is the user-configurable, up to 8 character name for the maintenance decision

description is the user-configurable, up to 24 character description of the maintenance decision

units is the engineering unit name or number for the maintenance decision

Example 1

ANALOG_MAINTENANCE msetpoint (setpoint) {SETPOINT, "Calculated Setpoint", 1}

Interpretation: A maintenance point that displays the value of *setpoint*, in units of ·F, will be created for the program.

Example 2

ANALOG_MAINTENANCE msetpointC ((setpoint-32)*1.8) {METRICSP,"Setpoint in deg", 56}

Interpretation: A maintenance point that displays the value of *setpoint*, will be created for the program and converted to Celsius. Engineering units will not be displayed.

Usage Rules

1. The *maintname* cannot be used as a BEST^{++} variable. The following example is invalid:

   x = msetpointC *2

2. If *name* is omitted, the default name of “VALUE” will be used.

3. If *description* is omitted, the default description of “Value” will be used.

4. Use a quotation mark before and after *description* if it consists of spaces or any characters (+, -, etc.).

5. Use analog_maintenance to display discrete values (0 and 1) in addition to analog values.

AND

This operator is used in a BEST^{++} statement to test two logical conditions.

It provides a result of TRUE if both the conditions tested are true and a result of FALSE if either of the conditions is false.

For analog variables, BEST^{++} interprets any number other than zero as TRUE. For discrete variables, 1 = TRUE, 0 = FALSE.

A statement may contain multiple ANDs.

Note: Do not confuse this logical operator with the mathematical operator +. The logical operator AND cannot be used to add numbers.

AND cannot be used to link statements together. Use the semicolon (;) instead.

Invalid: TURNON FAN1 AND FAN2
Valid: TURNON FAN1; TURNON FAN2

Truth Table

x_1	x_2	x_1 AND x_2
T	T	T
F	T	F
T	F	F
F	F	F

Syntax

x_1 AND x_2

where each x is a logical condition or variable.

Example 1

IF (TEMP1 > 72) AND (TEMP2 > 72) THEN RUN CHILLER ENDIF

Interpretation: If and only if both TEMP1 and TEMP2 are greater than 72 degrees when this statement is executed, then a task named CHILLER is activated.

Example 2

IF FAN1STAT AND FAN2STAT THEN TURNON FAN3 ENDIF

Interpretation: If both FAN1STAT and FAN2STAT are ON when this statement is executed, then FAN3 is turned on.

Example 3

IF FANSTAT1 AND PUMPSTAT AND FANSTAT2 THEN TURNON FAN3 ENDIF

Interpretation: If FANSTAT1 and PUMPSTAT and FANSTAT2 are ON when this statement is executed, then FAN3 is turned on.

Usage Rules

1. A variable, statement, or variable must immediately precede the word AND.

2. A variable, statement, or variable must immediately follow the word AND.

3. If multiple ANDs are used in a statement, they are evaluated pair-wise, from left to right. A AND B AND C is evaluated as (A AND B) AND C.

ARRAY

This reserved word defines an array name and the number of elements within the array. An array created using an ARRAY statement can consist of one of the following: hardware points, software points, HVAC functions, system functions, schedules, or alarms. Comparatively, an array created using a DIM statement can only consist of numbers. BEST^{++} allows only one dimensional arrays to be created.

Syntax

ARRAY *x* AS A *reserved word* {*y*}

where *x* is the name of the array.

reserved word is the array's element type. For a list of the element types, use the Paste Function command. Select ARRAY. Its element types will display in the Reserved Word list. Select an element type from the list to display how it is used in the ARRAY syntax.

y is a positive integer that denotes the number of elements in the array. The allowable entries are 1 to 99. The default value is 10.

Example

```
ARRAY PUMPS AS A DISC_OUTPUT {3}
CONNECT PUMP1 AS A DISC_OUTPUT {PUMP1}
CONNECT PUMP2 AS A DISC_OUTPUT {PUMP2}
CONNECT PUMP3 AS A DISC_OUTPUT {PUMP3}
TASK PUMPSTRT
PUMPS[1].ADDRESS = ? PUMP1   ~ SET ELEMENT 1
PUMPS[2].ADDRESS = ? PUMP2   ~ SET ELEMENT 2
PUMPS[3].ADDRESS = ? PUMP3   ~ SET ELEMENT 3

I = 1
IF PUMPS[1] EQ 1 THEN
        TURNON PUMPS[I + 1]
        TURNOFF PUMPS[I + 2]

ENDIF
```

Caution: You must use the PUMPS[1].ADDRESS, PUMPS[2].ADDRESS, PUMPS[3].ADDRESS syntax exactly as shown in the example to connect to the pumps. These statements must be inside the task where they will be executed at least once when the program runs.

Interpretation: An array of discrete output points named PUMPS with three elements is created. The elements are referenced as PUMPS[1], PUMPS[2], and PUMPS[3].

Using the indirect assignment symbol (?), the first array element is assigned to the CONNECTed DO at address 1. The second array element is assigned to the CONNECTed DO at address 2, and the third array element is assigned to address 3.

The value of the first pump is read and if it is on (= 1), the next pump is turned on and the following pump is turned off.

Conceptually, this array can be illustrated as follows:

ARRAY PUMPS

Element	Value of Element
PUMPS[1]	Value of PUMP1
PUMPS[2]	Value of PUMP2
PUMPS[3]	Value of PUMP3

See Also

DIM

Usage Rules

1. The number of elements in an array must be an integer value greater than zero.

2. You cannot assign an array to another array.

3. You must assign each item within an array separately.

ASIN

When used in a BEST^{++} statement, the ASIN (arc sine) function provides the inverse sine of an angle. The input (*x*) is the ratio of an opposite side of a right angle triangle and its hypotenuse. The output (result of ASIN *x*) is the angle subtended by the sides as expressed in radians. One radian equals $^{360}/_{(2\pi)}$ or 57.29577 degrees.

Syntax

ASIN *x*

where *x* is a constant, variable, or mathematical expression ranging from -1 to 1. Invalid statements return a value of *0*. For example, ASIN 5 is invalid.

Example

HALF PI = ASIN (1)

Interpretation: When BEST^{++} encounters this line in the task, it calculates the value of PI/2 .

ATAN

When used in a BEST^{++} statement, the ATAN (arc tangent) function provides the inverse tangent of an angle. The input (*x*) is the ratio of the opposite side and adjacent side of a right angle triangle. The output (result of ATAN *x*) is the angle subtended by the side as expressed in radians. One radian equals $^{360}/_{(2\pi)}$ or 59.29577 degrees.

Syntax

ATAN *x*

where *x* is a constant, variable, or mathematical expression.

Example

QUARTERPI = ATAN (1)

Interpretation: When BEST^{++} encounters this line in the task, it calculates the answer as $^{\pi}/_{4}$ or .7853982 degrees.

AUTO

This reserved word is used in a BEST^{++} statement to remove a force from an internally CONNECTed point or an array element. BEST^{++} can remove forces equal to or less than the value entered as *forcepri* in the TASK statement if a level 8 force (BEST^{++}) is not involved. For information on *forcepri*, refer to TASK in this chapter and Appendix A (Comfort Controller Force Priorities).

A task with a higher priority for execution, as defined in the TASK statement, will force over a task with a lower priority.

For example, you have two BEST^{++} tasks that force the same point:

TASK1 — *forcepri*: 4 (Building Supervisor)

TASK2 — *forcepri:* 8 (BEST^{++}, which is lower than 4)

TASK1 will force over TASK2. BEST^{++} will hold TASK2's force in reserve.

A task can AUTO any point with the same or higher force level, that is, a task with force level 4 can AUTO a point with force level 7, but a task with level 7 cannot AUTO a point with level 4.

If TASK1 AUTOs the force, BEST^{++} will take off the level 4 force and put back the level 8 force.

If TASK2 AUTOs the force, it will take off the reserved level 8 force but leave the level 4 force.

Syntax

AUTO *x*

where *x* is a configured and CONNECTed input/output point.

Example

```
CONNECT PUMP AS A DISC_OUTPUT {PUMP}
IF CHILLEDWATER > 45 THEN AUTO PUMP ENDIF
```

Interpretation: When BEST^{++} encounters this line and CHILLEDWATER is greater than 45, BEST^{++} removes the force (ON/OFF) on PUMP and returns it to automatic control.

CALL

This reserved word is used to execute a subroutine. Subroutines are useful when the same mathematical procedure is performed in multiple places in a task or tasks. If your task requires the use of subroutines, use the CALL statement to invoke the subroutine. The task in which the CALL statement resides is known as the CALLing task.

Syntax

x = CALL *y* (*arg1*,*arg2*,*arg3*,*arg4*)

where *x* is any variable.
y is the name of the subroutine.
arg1, *arg2*, *arg3*, and *arg4* are the values(arguments) passed to the subroutine. Arguments are optional, and up to four are permitted.

Example

```
~  SUBROUTINE to filter a value between a and b
~        x is the value to be filtered
~        if x is less than a then result is 0
~        x increases proportionally from 0 to c between a and b
~        if x is greater than b the result is c
SUBROUTINE FILTER (x,a,b,c)
        IF x < a THEN
                result = 0
        ELSE
                IF x >6 THEN
                        result = c
                ELSE
                        result = (b - a) * (x - a) / c
                ENDIF
        ENDIF
RETURN (result)
TASK subtest
input = 7
filteredinput = CALL FILTER (input,-5,13.7,100)
ENDTASK
```

Interpretation: Refer to the remarks in the example.

See Also

RETURN, SUBROUTINE

Usage Rules

1. CALL must always appear to the right of the = sign.
2. If you do not use arguments, you must still type ( ) after the name of the subroutine.
3. A SUBROUTINE Must end with a RETURN statement.

CONNECT

Any hardware or software point, configuration or maintenance decision, HVAC function, system function, schedule, or alarm in a controller that is to provide input to a BEST^{++} task or that is to be controlled by a BEST^{++} task must be identified by a variable name (must be CONNECTed).

Once an item is CONNECTed to a variable name, it will be read or controlled whenever the connected variable name is encountered in the BEST^{++} task.

There are two basic types of CONNECTS — internal and external. If the item to be read or controlled is internal to (exists in) the controller in which the task resides, use an internal CONNECT. If the item to be read or controlled is external to (does not exist in) the controller in which the task resides, use an external CONNECT.

Compiling FID BEST CONNECT Statements

When compiling existing FID BEST programs, the BEST^{++} compiler automatically converts the old syntax to an appropriate BEST^{++} syntax.

FID BEST and BEST^{++} Syntax

To help familiarize FID BEST users with BEST^{++} CONNECT statements, the explanation of CONNECT includes both FID BEST and BEST^{++} CONNECT syntax.

Comparison of BEST and BEST^{++} CONNECT Syntax

Below is a comparison of FID BEST and BEST^{++} CONNECT syntax. Descriptions, examples, and usage rules begin on the following page.

Internal CONNECTs (CONNECTs within a controller)

CONNECTing to a point by variable name:
FID BEST CONNECT *PTNAME* AS A *type* TO *point#*
BEST++ CONNECT *var* AS A *vartype* {*PTNAME*}

Note: *point#* is a unique number greater than 99 (1-64 hardware, 65-99 software). *PTNAME* is the actual point name as configured in the controller.

CONNECTing to a point by point number
FID BEST CONNECT *var* AS A *type* TO *point#*
BEST++ CONNECT *var* AS A *vartype* {, *point#*}

Note: *point#* is the actual hardware or software channel number as configured in the controller.

CONNECTing to a decision:
FID BEST CONNECT *var* AS A DECISION TO *table# point#, decision#*
BEST++ CONNECT *var* AS A *vartype* {*VARNAME}*

Note: *VARNAME* is the actual name of a hardware or software point, HVAC function, system function, schedule, or alarm as configured in the controller. You can access a function, or maintenance or configuration decision within a function, by appending *.decname* to *var* in a statement later in the task — do not append it to *var* in the CONNECT statement.

CONNECTing to a task:
FID BEST not supported
BEST++ CONNECT *var* AS A *task* {*TASKNAME*}

Note: *TASKNAME* is the actual name of the task.

External CONNECTs (CONNECTs to another controller on the same CCN)

CONNECTing to a point by variable name:
FID BEST CONNECT *PTNAME* AS A *type* TO *point#* IN *CCN_element#*
BEST++ NETWORK_POINT *var {CCN_element#, CCN_bus#, PTNAME}*

Note: *point#* is a unique number greater than 99. *PTNAME* is the actual point name as configured in the controller.

CONNECTing to a point by point number:
FID BEST CONNECT *var* AS A *type* TO *point#* IN *CCN_element#*
BEST++ NETWORK_POINT var {*CCN_element#, CCN_bus#, , point#}*

Note: *point#* is the actual hardware or software channel number as configured in the controller. You must CONNECT to a point in a UT203 FID or VVT Gateway by variable name instead of by point number.

CONNECTing to a decision in a UT203 FID, 32MP Gateway, or VVT Gateway:
FID BEST CONNECT *var* AS A DECISION TO *table# point#, decision#* IN *CCN_element#*
BEST++
UT203 FID NETWORK_DECISION *var* {*CCN_element#, CCN_bus#, table#, point#, decision#}*
32 MP GW NETWORK_DECISION *var* {*CCN_element#, CCN_bus#, 0, table#, index#}*
VVT GW NETWORK_DECISION *var* {*CCN_element#, CCN_bus#, 0, device#, decision#}*

CONNECTing to a point's decision by decision name in a Comfort Controller:
FID BEST not supported
BEST++ NETWORK_CONNECT *var* AS A *vartype.decname {CCN_element#, CCN_bus#, PTNAME}*

CONNECTing to a function's decision in a Comfort Controller:
FID BEST not supported
BEST++ NETWORK_CONNECT *var* AS A *functiontype.subfunction.decname {CCN_element#, CCN_bus#, FUNCNAME}*

Note: In this external CONNECT syntax, *functiontype* is the algorithm, system function, schedule, or alarm type. *subfunction* is the name of an HVAC function or alarm function. *decname* is a maintenance or configuration decision within the subfunction. *FUNCNAM*E is the actual name of the function (as described by *functiontype*) as configured in the controller.

Internal CONNECTs

Internally CONNECTing To a Point by Variable Name

Read the information below to CONNECT to an internal point by variable name.

FID BEST Syntax

CONNECT *PTNAME* AS A *type* TO *point#*

where *PTNAME* is the actual point name as configured in the controller.

type is the point type: AI, AO, DI, or DO.

point# is a unique number greater than 99.

FID BEST Example

CONNECT SPT AS A AI TO 101

BEST++ Syntax

CONNECT *var* AS A *vartype* {*PTNAME*}

where *var* is the variable name you are assigning to the point.

vartype is the point type. For a list of *vartype*s, use the Paste Function Command to display the syntax for CONNECT. *vartypes* will appear in the Reserved Word List. For information about the Paste Command, refer to the Edit Menu Summary in The BEST++ Programmer's Environment chapter.

PTNAME is the actual point name as configured in the controller.

BEST++ Example

CONNECT SPT AS A TEMP_INPUT {SPT}

Internally CONNECTing To a Point by Point Number

Read the information below to CONNECT to a point by point number.

FID BEST Syntax

CONNECT *var* AS A *type* TO *point#*

where *var* is the variable name you are assigning to the point.

type is the point type: AI, AO, DI, or DO.

point# is the actual hardware (1-64) or software (65-99) channel number.

FID BEST Example

CONNECT SUPFAN AS A DO TO 8 USING SAFETY LIMITS 0,0

BEST++ Syntax

CONNECT *var* AS A *vartype* {, *point#*}

where *var* is the variable name you are assigning to the point.

vartype is the point type. For a list of *vartype*s, use the Paste Function Command to display the syntax for CONNECT. *vartypes* will appear in the Reserved Word List. For information about the Paste Command, refer to the Menu Command Summary in The BEST++ Programmer's Environment chapter.

point# is the actual hardware (1-64) or software (65-99) channel number.

BEST++ Example

CONNECT supfan AS A DISC_OUTPUT {,8}

Internally CONNECTing To a Decision

Read the information below to CONNECT to configuration or maintenance decisions.

Note: It is important to note that you cannot write to Comfort Controller maintenance decisions. You can, however, read from Comfort Controller maintenance decisions without negative consequence.

FID BEST Syntax

CONNECT *var* AS A DECISION TO *table# point#, decision#*

where *var* is the variable name you are assigning to the decision.

table# is the controller table number associated with the decision type. The decision types are as follows:
0 = Global decision (applicable only to UT203 FIDs)
1 = DO, 2 = AO, 3 = AI, 4 = DI, 5 = PI, 6 = TS, 7 = SS, 8 = HOL

point# is the actual hardware or software channel number.

decision# is the decision number.

FID BEST Example

CONNECT STPTLO AS A DECISION TO 701, 2

BEST++ Syntax

CONNECT *var* AS A *vartype* {*VARNAME}*

where *var* is the variable name you are assigning to the decision.

vartype is the point type or function type. For a list of *vartype*s and function types, use the Paste Function Command to display the syntax for CONNECT. *vartypes* and function types will appear in the Reserved Word List. For information about the Paste Command, refer to the Menu Command Summary in The BEST++ Programmer's Environment chapter.

VARNAME is the actual name of a hardware or software point, HVAC function, system function, schedule, or alarm as configured in the controller. You can access a decision within a function by appending *.decname* to *var* later in the task. Do not append it in the CONNECT statement. You can access a function or maintenance or configuration decision of a subfunction (HVAC or alarm function) by appending *.subfunction.decname* to *var* later in the task. Do not append it in the CONNECT statement.

To connect to the Occupied High parameter of a Setpoint Schedule, select EDIT from the toolbar, then PASTE FUNCTION, ALL, and highlight CONNECT.
Click on the SYNTAX button and you will see the syntax for CONNECT in the bottom box. Highlight SETPOINT in the upper box and click on the FUNCTIONS button. Scroll down the bottom box until you see ~ C or ~ M on the right. You will use setpointname.OCCHGH in your program. Connect to the setpoint schedule in your dictionary as follows:

BEST++ Example

CONNECT *setpointname* AS A SETPOINT {*SETPTxx*}

where *SETPTxx* is the actual setpoint schedule name. In the program, use setpointname.OCCHGH when referring to the Occupied High Setpoint parameter.

In the global dictionary: CONNECT SCHED01 AS A SETPOINT {SETPT01}

In the task:

```
STEP ONE
IF SCHED01.OCCHGH <> 75 THEN SCHED01.OCCHGH = 75 ENDIF
DELAY taskname FOR 30
REPEAT
```

External CONNECTs

Use external CONNECTs when connecting to a function (hardware or software point, configuration or maintenance decision, HVAC function, system function, schedule, or alarm) in another device. The device you are connecting to does not have to be on the same bus.

Externally CONNECTing To a Point by Variable Name

Read the information below to CONNECT to a point in another controller by variable name. You can connect to the same point name in multiple controllers using NETWORK_POINT.

FID BEST Syntax

CONNECT *PTNAME* AS A *type* TO *point#* IN *CCN_element#*

where *PTNAME* is the actual point name as configured in the controller.

type is the point type: AI, AO, DI, or DO.

point# is a unique number greater than 99.

CCN_element# is the system element number of the Comfort Controller containing the point to which you are CONNECTing.

FID BEST Example

CONNECT SPT AS A AI TO 101 IN 3

BEST++ Syntax

NETWORK_POINT *var {CCN_element#, CCN_bus#, PTNAME}*

where *var* is the variable name you are assigning to the point.

CCN_element# is the system element number of the Comfort Controller containing the function to which you are CONNECTing.

CCN_bus# is the CCN bus number of the Comfort Controller containing the point to which you are CONNECTing.

PTNAME is the actual point name as configured in the controller.

BEST++ Example

NETWORK_POINT SPT AS {2,5,SPT}

Externally CONNECTing To a Point by Point Number

Read the information below to CONNECT to a point in another controller by point number.

FID BEST Syntax

CONNECT *var* AS A *type* TO *point#* IN *CCN_element#*

where *var* is the variable name you are assigning to the point.

type is the point type: AI, AO, DI, or DO.

point# is the actual hardware or software channel number.

CCN_element# is the system element number of the controller to which you are CONNECTing.

FID BEST Example

CONNECT SPT AS A AI TO 8 IN 3

BEST++ Syntax

NETWORK_POINT *var {CCN_element#,CCN_bus#,,point#}*

where *var* is the variable name assigned to the CONNECTed object.

CCN_element# is the system element number of the Comfort Controller containing the function to which you are CONNECTing.

CCN_bus# is the CCN bus number of the Comfort Controller containing the function to which you are to CONNECTing.

The extra comma is a placeholder for *PTNAME*.

point# is the actual hardware or software channel number.

Note: When CONNECTing to a point by point number in a UT203 FID, add *1* to *point#*. For example, if CONNECTing to hardware channel 8, *point#* must be 9.

When CONNECTing to a point by point number in a VVT Gateway, add *4* to *point#*. For example, if CONNECTing to hardware channel 8, *point#* must be 12.

BEST++ Example

NETWORK_POINT SPT {2,5,,8}

Externally CONNECTing to a Decision in a UT203 FID, 32MP Gateway, or VVT Gateway

Read the information below to CONNECT to a decision in a UT203 FID, 32MP Gateway, or VVT Gateway.

FID BEST Syntax

CONNECT *stptlo* AS A DECISION TO *table# point#, decision#* IN *CCN_element#*

where *var* is the variable name you are assigning to the decision.

table# is the 203 FID or Gateway table associated with the decision type. Valid 203 FID types are 0 = Global decision, 1 = DO, 2 = AO, 3 = AI, 4 = DI, 5 = PI, 6 = TS, 7 = SS, 8 = HOL. For 32MP Gateway and VVT Gateway information, refer to the respective Overview and Configuration manuals.

point# is the actual hardware or software channel number.

decision# is the decision number.

CCN_element# is the system element number of the UT203 FID or Gateway containing the decision to which you are CONNECTing.

FID BEST Example

CONNECT *STPTLO* AS A DECISION TO 701, 2 IN 3

BEST++ Syntax

UT203 FID: NETWORK_DECISION *var* {*CCN_element#, CCN_bus#, table#, point#, decision#}*

32MP Gateway: NETWORK_DECISION *var* {*CCN_element#, CCN_bus#, 0, table#, decision#}*

VVT GW: NETWORK_DECISION *var* {*CCN_element#, CCN_bus#, 0, device#, decision#}*

where *var* is the variable name you are assigning to the decision.

CCN_element# is the system element number of the UT203 FID or Gateway containing the decision to which you are CONNECTing.

CCN_bus# is the CCN bus number of the UT203 FID or Gateway containing the decision to which you are CONNECTing.

table# is the 203 FID or Gateway table associated with the decision type. Valid 203 FID types are 0 = Global decision, 1 = DO, 2 = AO, 3 = AI, 4 = DI, 5 = PI, 6 = TS, 7 = SS, 8 = HOL. For 32MP Gateway and VVT Gateway information, refer to the respective Overview and Configuration manuals.

point# is the actual hardware or software channel number.

decision# is the decision number

index# is the index number

BEST++ Example

NETWORK_DECISION stptlow {3, 0, 7, 1, 2}

Externally CONNECTing to a Point's Decision by Decision Name in a Comfort Controller

Read the information below to CONNECT to a point's decision by decision name within a Comfort Controller.

FID BEST Syntax

Not supported

BEST++ Syntax

NETWORK_CONNECT *var* AS A *vartype.decname {CCN_element#, CCN_bus#, PTNAME}*

where *var* is the variable name you are assigning to the decision.

vartype is the point type. For a list of *vartypes*, use the Paste Function Command to display the syntax for CONNECT. *vartypes* will appear in the Reserved Word List. For information about the Paste Command, refer to the Edit menu summary in The BEST++ Programmer's Environment chapter.

.*decname* is name of the decision.

CCN_element# is the system element number of the Comfort Controller containing the decision to which you are CONNECTing.

CCN_bus# is the CCN bus number of the Comfort Controller containing the decision to which you are CONNECTing.

PTNAME is the actual point name as configured in the controller.

BEST++ Example

NETWORK_CONNECT supfan AS A DISC_OUTPUT.FORCE {3, 0, SUPFAN}

Externally CONNECTing to a Function's Decision in a Comfort Controller

Read the information below to CONNECT to a function's decision in a Comfort Controller.

Note: It is important to note that you cannot write to Comfort Controller maintenance decisions. You can, however, read from Comfort Controller maintenance decisions without negative consequence.

FID BEST Syntax

Not supported

BEST++ Syntax

NETWORK_CONNECT *var* AS A *functiontype.subfunction.decname {CCN_element#, CCN_bus#, FUNCNAME}*

where *var* is the variable name you are assigning to the decision.

functiontype is the algorithm, system function, schedule, or alarm type. For a list of the *functiontype*s, use the Paste Function Command to display the syntax for CONNECT. *Functiontype*s will appear in the Reserved Word List. For information about the Paste Command, refer to the Edit menu summary in the BEST++ Programmer's Environment chapter.

.*subfunction* is name of the HVAC or alarm function.

.*decname* is the name of the maintenance or configuration

decision within the subfunction to which you are CONNECTing.

CCN_element# is the system element number of the Comfort Controller containing the decision to which you wish to CONNECT.

CCN_bus# is the CCN bus number of the Comfort Controller containing the decision to which you wish to CONNECT.

FUNCNAME is the actual name of the algorithm, system function, schedule, or alarm as configured in the Comfort Controller.

BEST++ Example

```
NETWORK_CONNECT DIALARM AS A
DI_STATE_ALARM.DISTALM.EXCDLIM {3, 0, DSALM01}
```

CONNECT Usage Rules

1. All variable names established with the CONNECT statement must be unique within a BEST++ program.

2. Although CONNECT statements can appear anywhere after the PROGRAM statement in a program, it is good practice to place all CONNECTs in the global dictionary and send (download) the global dictionary to the Comfort Controller before any other programs that reference the CONNECTed variable names are sent.

3. A connected variable cannot be assigned a value beyond the range of the point or decision to which it is connected.

   For example, if a variable is connected to an AI YSI 10K Thermistor, it cannot be assigned a value less than -40 or greater than 244.

4. You can connect to a device on a different bus, i.e., BEST++ can communicate through a bridge.

5. When connecting to an external device, you must use OUTPUTTO and INPUTFROM to perform the write and read operations to and from the device.

6. If a CONNECT is made to a configuration parameter in EEPROM, it should not be written to more than once per hour. EEPROM allows about 100,000 writes to each location.

COS

When used in a BEST^{++} statement, the COS (cosine) function provides the cosine of an angle. The input (*x*) is the angle, expressed in radians, subtended by the adjacent side of a right angle triangle and the hypotenuse. The output (result of COS *x*) is the ratio of the adjacent side and the hypotenuse.

Syntax

COS *x*

where *x* is a constant, variable, or mathematical expression.

Example

RATIO = COS (*x* + .3)

Interpretation: When BEST^{++} encounters this line in the task, it adds .3 to *x* and calculates the cosine of the angle as a number between -1 and 1.

COSH

When used in a BEST^{++} statement, the COSH (hyperbolic cosine) function provides the hyperbolic cosine of a number. The formula for calculating COSH is:

COSH(c) = (EXP (x) - EXP (-x)) / 2

Syntax

COSH *x*

where *x* is a constant, variable or mathematical expression.

Example 1

HYPERCOS = COSH (.3)

Interpretation: When BEST^{++} encounters this line, it calculates the hyperbolic cosine of .3 as 1.0453.

COUNTER

This reserved word is used to create a counter that can be INCREMENTed by a value of 1, DECREMENTed by a value of 1, and RESET to 0. When BEST^{++} encounters the name of the counter, it returns the value currently in the counter. BEST^{++} can compare this value to a variable or a constant.

The maximum number to which a counter can count is 65,535. When the counter reaches this value, it automatically resets to 0.

Syntax

COUNTER *x*

where *x* is the counter name.

Note: The following two methods for incrementing, decrementing, and resetting x provide the same result. Both methods are acceptable in BEST^{++}.

COUNTER x	VARIABLE x	{0}
INCREMENT x	x = x + 1	
DECREMENT x	x = x - 1	
RESET x	x = 0	

Example 1

```
COUNTER NUM_CHILLERS
IF CHILLER_STATUS EQ 1 THEN INCREMENT NUM_CHILLERS
ENDIF
```

Interpretation: When BEST^{++} encounters the first line, it creates a counter named NUM_CHILLERS. When BEST^{++} encounters the second line and CHILLER_STATUS equals 1 (ON), BEST^{++} increments NUM_CHILLERS by 1.

Example 2

```
COUNTER NUM_CHILLERS
IF NUM_CHILLERS > 0 THEN LEAD_PUMP = 1 ENDIF
```

Interpretation: When BEST^{++} encounters the first line, it creates a counter named NUM_CHILLERS. When BEST^{++} encounters the second line and CHILLER_STATUS is greater than zero, BEST^{++} sets LEAD_PUMP equal to 1 (ON).

See Also INCREMENT, DECREMENT, RESET

Usage Rules

1. A counter can be created anywhere in a program.

2. A counter cannot be assigned a value.

   Invalid:

   ```
   COUNTER COUNT1
   COUNT1 = 90
   ```

3. Each counter name must be unique within a given program.

4. Counter names must conform to the rules for variable names.

5. You can create as many counters as you want.

6. Once a counter is created, all further reference to the counter is by its name.

7. Counters created in the global dictionary can be accessed by all programs in the controller.

DECREMENT

This reserved word is used to decrease the value of a counter by one.

Note: If you DECREMENT a counter that is equal to 0, its value will be equal to 65535.

Syntax

DECREMENT *x*

where *x* is the counter name.

Example

DECREMENT COUNTER1

Interpretation: When BEST^{++} encounters this line, it decreases the current value of the counter named COUNTER1 by one. Note that COUNTER1 must have been previously defined as a COUNTER.

See Also

COUNTER, INCREMENT, RESET

Usage Rules

1. DECREMENT can appear at the beginning of a line, after a THEN statement, or after a semicolon (;).
2. The name of a counter must immediately follow the word DECREMENT.
3. The counter specified in the DECREMENT statement must be defined in the same controller as the DECREMENT statement.

This reserved word is used in a BEST^{++} statement to suspend the execution of a specified task for a user-defined number of seconds.

DELAY

When it is encountered, the task is delayed for the specified time and, after the delay, continues executing at the next program line. DELAY can be used to delay the task that the statement is in, or to delay another task.

When the DELAY statement is encountered and executed, the specified delay time period begins and cannot be reset. When the delay time expires, the specified BEST^{++} task continues executing at the line of the task where it stopped.

The time delay is expressed in seconds (1 through 32767).

Syntax

DELAY *x* FOR *y*

where *x* is the task name to delay.

y is the number of seconds (wait time) the task is to be delayed.

Alternate Syntax

DELAY *x* FOR *y*
DELAY *x* (*y*)
DELAY (*x* * *y*)
DELAY *x* FOR *var*

Examples 1 - 4

DELAY CHILLER FOR 30
DELAY CHILLER (30)
DELAY (CHILLER * 30)
DELAY CHILLER FOR *y*

Note: where *y* = any assigned number.

Interpretation: Examples 1 - 4 illustrate the alternate methods for typing the DELAY statement syntax.

IF SFS EQ 0 THEN TURNON PMP1; DELAY AHU FOR 40; TURNON PMP2;

Example 5

GOTO ONE ENDIF

Interpretation: If SFS is off, then PMP1 will be turned on. The task named AHU will be inhibited for 40 seconds, and then it will turn on PMP2. The task will then GOTO Step ONE.

Usage Rules

1. Only one BEST^{++} task name can be specified after the word DELAY.

2. The BEST^{++} task that will be DELAYed may or may not be the same one that contains the DELAY statement.

   Before DELAYing another BEST^{++} task in a different source file, you must first CONNECT that task. For example:

   CONNECT PUMP_TASK AS A TASK {PMPTSK}

   TASK ONE
   DELAY PUMP_TASK FOR 10
   ENDTASK

3. The task that will be DELAYed must reside in the same controller as the one containing the DELAY statement.

4. Only one wait time can be specified. Specify the wait time in seconds. Allowable entries are 1 through 32767. Variable names and mathematical expressions are valid entries.

   Note: If a task is to be unconditionally prohibited from execution, or if the wait time is to be indefinite, use the HALT statement.

This reserved word is used to define an array name and the number of elements within the array. An array created using a DIM state-

DIM

ment can only consist of numbers. BEST^{++} allows only one dimensional arrays to be created.

The amount of Comfort Controller memory used by an array is proportional to its size. Each array element uses the same amount of memory as a variable.

Note: You cannot use a DIM statement to create an array of hardware and software points. To create an array of hardware and software points or any other function (configuration or maintenance decisions, HVAC functions, system functions, schedules or alarms), use an ARRAY statement.

Syntax

DIM *x*{*y*}

where *x* is the name of the array.

y is a positive integer that denotes the number of elements in the array. The allowable entries are 0 to 99; the default value is 10.

Example

```
DIM CHILLER{3}
CHILLER[1] = 7; CHILLER[2] = 14; CHILLER[3] = 0
 IF CHILLER[3] EQ 1 THEN GOTO TEN
```

Interpretation: An array named CHILLER with 3 elements has been created. The elements are referenced as CHILLER[1], CHILLER[2], and CHILLER[3].

The first element of the array is assigned the value *7*. The second element of the array is assigned the value *14*. The third element of the array is assigned the value *0*.

The value of the third element of the array is compared to the value 1 when this statement is executed. If the value of the third element is equal to 1, the task is directed to go to step or label TEN.

The value in the third element of the array is 0 and not 1. Therefore, the task is not directed to go to TEN.

Conceptually, this array can be illustrated as follows:

DIM CHILLER	
Element	Value of Element
CHILLER[1]	7
CHILLER[2]	14
CHILLER[3]	0

ARRAY

See Also

Usage Rules

1. The number of elements in an array must be an integer value 0 to 99.
2. There should be no space between the name of the array and the number of elements.
3. You cannot assign an array to another array.
4. You must assign each item within an array separately.

```
Invalid    DIM CHILLER[3]
           DIM PUMP{3}
           CHILLER = PUMP

Valid      DIM CHILLER{3}
           DIM PUMP{3}
           CHILLER[1] = PUMP[1]
           CHILLER[2] = PUMP[2]
           CHILLER[3] = PUMP[3]
```

The statement on the right side of the equals sign (=) is evaluated before being assigned to the array element.

```
Valid      DIM CHILLER {3}
           CHILLER [3] = SPT
           (where SPT = 75
           so CHILLER [3] = 75
```

DOM is used in BEST^{++} statements to obtain the **D**ay **O**f the **M**onth. Whenever it is encountered, a value from 1 to 31 is pro-

DOM

vided. The value may then be compared to a variable or constant, or assigned to a variable.

The value assigned to DOM is maintained by the internal calendar of the Comfort Controller.

Syntax

DOM operator *x*

where *x* is any variable or constant between 1 and 31.

Alternate Syntax

x relational operator DOM
x = DOM

Example 1

IF DOM > 15 THEN TURNON FAN1 ENDIF

Interpretation: FAN1 is turned on if this statement is executed on the 16th through the 31st of any month.

Example 2

SERVCDAY = 15
IF DOM EQ SERVCDAY THEN TURNOFF PUMP1 ENDIF

Interpretation: A variable called SERVCDAY has been assigned the value 15 and PUMP1 is turned off whenever this statement is executed on the 15th day of every month.

Example 3

NOSERVC = DOM
IF NOSERVC EQ 15 THEN TURNON PUMP1 ENDIF

Interpretation: A variable NOSERVC is equal to the current day of the month and PUMP1 is turned on when the day of the month is the 15th.

1. DOM cannot be assigned a value.

Usage Rules

For example: DOM = 3 is improper usage. Apart from that, DOM may appear anywhere in a line of a program.

Note: DOM EQ 3 could be used as a condition to be tested in an IF... THEN... statement. This is because you are not actually assigning a value *to* DOM. You are only testing to see if DOM *is equal* to a value.

2. DOM must be preceded or followed by one of the following operators:

   < < = > > = EQ < > = IF

3. If DOM precedes one of the above operators, then a variable, constant, or mathematical expression must immediately follow the operator.

4. DOM cannot be used as a variable. In order to operate on the value of DOM it must be assigned to a variable.
   For example: X = DOM.

This reserved word is used in a BEST^{++} statement to obtain the numerical value of the **D**ay **Of** the **W**eek. Whenever it is encoun-

DOW

tered, a value from 1 to 7 is provided. This value may then be compared to a variable or constant, or it may be assigned to a variable.

The value assigned to DOW is maintained by the internal calendar of the Comfort Controller.

Numerical values correlate to days of the week as follows:

1	=	Monday	5	=	Friday
2	=	Tuesday	6	=	Saturday
3	=	Wednesday	7	=	Sunday
4	=	Thursday			

Note: DOW cannot be used to assign a value to the Comfort Controller clock.

Syntax

DOW relational operator *x*	*or*
x relational operator DOW	*or*
x = DOW	(used for assignment)

where *x* is any variable or constant between 1 and 7.

Example 1

IF DOW > 4 THEN TURNON FAN1 ENDIF

Interpretation: FAN1 is turned on each Friday, Saturday, and Sunday whenever this statement is executed.

Example 2

IF DOW EQ 5 THEN TURNOFF FAN1 ENDIF

Interpretation: FAN1 is turned off every Friday when this statement is executed.

Example 3

NOWORK = DOW

Interpretation: A variable called NOWORK is assigned the value of DOW.

1. DOW cannot be assigned a value.

Usage Rules

For example, DOW = 3 is improper usage. Apart from this, DOW may appear anywhere in a line of a program.

Note: DOW EQ 3 could be used as a condition to be tested in an IF... THEN... statement. This is because you are not actually assigning a value *to* DOW. You are only testing to see if DOW *is equal* to a value.

2. DOW must be preceded or followed by one of the following operators:

   < < = > > = EQ < > = IF

3. If DOW precedes one of the above operators, then a variable, constant, or mathematical expression must immediately follow the operator.

4. DOW cannot be used as a variable. In order to operate on the value of DOW it must be assigned to a variable.
   For example: X = DOW.

This reserved word is used in BEST^{++} statements to obtain the numerical **D**ay **O**f the **Y**ear. Whenever it is encountered, a value

DOY

from 1 to 366 is provided. This value may then be compared to a variable or constant, or it may be assigned to a variable.

The value assigned to DOY is maintained by the internal calendar of the Comfort Controller.

Syntax

DOY relational operator *x*

Alternate Syntax

x relational operator DOY
x = DOY

Example 1

IF DOY > 121 THEN TURNOFF PUMP1 ENDIF

Interpretation: PUMP1 is turned off when this statement is executed after May 1st.

Example 2

IF (DOY > 121) AND (DOY < 305) THEN TURNON FAN1 ENDIF

Interpretation: FAN1 is turned on when this statement is executed between May 1st and November 1st.

Example 3

NOWORK = DOY
IF (NOWORK > 358) AND (NOWORK < 364) THEN TURNOFF FAN ENDIF

Interpretation: FAN is turned off when this statement is executed between December 24 and December 30.

Example 4

X = DOY + 30

Interpretation: X is assigned the value DOY + 30.

1. DOY cannot be assigned a value.

Usage Rules

For example, DOY = 3 is improper usage. Apart from this, DOY may appear anywhere in a line of a program.

Note: DOY EQ 3 could be used as a condition to be tested in an IF... THEN... statement. This is because you are not actually assigning a value *to* DOY. You are only testing to see if DOY *is equal* to a value.

2. DOY must be preceded or followed by one of the following operators:

   < < = > > = EQ < > = IF

3. If DOY precedes one of the above operators, then a variable, constant, or mathematical expression must immediately follow the operator.

4. DOY cannot be used as a variable. In order to operate on the value of DOY it must be assigned to a variable.
   For example: X = DOY.

This reserved word is used in a BEST^{++} statement to define the ending of a group of statements that will be executed together, in

ENDLOOP

sequence, for a specified number of times.

Syntax

LOOP *x* FROM *y* TO *z*
ENDLOOP *x*

where *x* is the name of the loop.
y is the start value.
z is the end value.

Example

```
LOOP ONE FROM 1 TO 10
IF ONE > 5 THEN TURNON FAN[ONE]
ELSE TURNOFF FAN[ONE]
ENDIF
ENDLOOP ONE
```

Interpretation: When BEST^{++} encounters the LOOP statement, it executes the lines of the task between the LOOP and ENDLOOP statements ten times. It then executes the ENDLOOP statement.

See Also

LOOP

Usage Rules

1. You must terminate a loop with an ENDLOOP statement.
2. The loop name must immediately follow ENDLOOP.
3. You can use the loop name only once in a BEST program. The following is invalid:

```
LOOP i FROM 1 TO 10
        SUM = SUM + i
ENDLOOP i
LOOP i FROM 7 TO 23
        SUM = SUM + i/2
ENDLOOP i
```

ENDTASK is a reserved word that defines the end of a BEST^{++} task.

ENDTASK

Syntax

ENDTASK

Example

```
TASK TEMPAVG
STEP ONE
TOTALDEG = POINT1 + POINT2 + POINT3
AVERAGE = TOTALDEG/3
DELAY TEMPAVG FOR 30
REPEAT
ENDTASK
```

EXIT is a reserved word that causes the BEST^{++} task in which it appears to stop running for the current reschedule interval.

EXIT

When this word is encountered, the task immediately stops executing.

The task does not resume execution until it is reactivated or rescheduled. This can be done with another task containing a RUN or RESCHEDULE statement.

EXIT is similar to the reserved word ENDTASK, which is discussed later in this chapter.

Syntax

```
EXIT
```

Example 1

```
WHEN X > 70
TURNON FAN1
EXIT
```

Interpretation: When the first statement shown in this example is encountered, the task waits for X to be greater than 70. When it is, FAN1 is turned on and the task stops running.

Example 2

```
IF TEMP1 > 72 THEN EXIT ENDIF
```

Interpretation: When this statement is executed, the task is stopped if the value of TEMP1 is greater than 72 degrees.

Usage Rule

EXIT does not require that a task name be specified. It assumes that the task to be stopped is the one in which it is included.

This function is used in a BEST^{++} statement to raise the numerical value of e to the value of a constant, variable, or mathematical expres-

EXP

sion. EXP is the inverse of LOG.

Syntax

EXP *x*

where *x* is a constant, variable, or mathematical expression.

Example 1

e = EXP (1)

Interpretation: When BEST^{++} encounters this line, it calculates the value of e as 2.71828.

See Also

LOG, POWER

FLOAT_CONFIGURATION

This reserved word allows a configuration variable to be defined. This may be useful in a BEST^{++} task to allow the user to alter the value of a BEST^{++} decision from the configuration screen without having to create and force software points.

Syntax

FLOAT_CONFIGURATION *configname*
{name,"description", units, lowlimit, highlimit, initvalue}

where *configname* is a unique BEST^{++} user-configurable name for the configuration decision.

name is a user-defined, up to 8 character description of the configuration decision.

description is a user-defined, up to 24 character description of the configuration decision.

units is the engineering unit name or number for the configuration decision.

lowlimit is the smallest allowable entry.

highlimit is the largest allowable entry.

initvalue is the initial value of the variable *configname*.

Example 1

FLOAT_CONFIGURATION setpoint
{SETPOINT, "User Setpoint", 1, 50, 80, 70}

Interpretation: A configuration point will be created for the program which will display the value 70 on the configuration screen. The user may change the value between 50 and 80 and send it to the controller. Any statement that uses the BEST^{++} name setpoint will use the last configured value.

Usage Rules

1. If *name* is omitted, the default name of "VALUE" will be used.
2. If *description* is omitted, the default description of "Value" will be used.
3. Use a quotation mark before and after *description* if it consists of spaces or any characters (+, -, etc.).

FRACTION

This function is used to provide the fractional portion of the argument. The fractional portion is the numeric value to the right of the decimal point.

Syntax

FRACTION (*x)*

where *x* is any constant, variable, or mathematical expression.

Example

N = FRACTION (SPT)

Interpretation: When BEST^{++} encounters this line, it calculates the value of N. If SPT = 79.61, then N = .61.

See Also

REMAIN, ROUNDDOWN, ROUNDUP

GOTO

This reserved word is used in BEST^{++} statements to control the sequence of execution within a program.

It tells a BEST^{++} task to leave the current statement and go to a named step or label in the same task.

Syntax

GOTO *x*

where *x* is a step or label in the task that contains the GOTO statement.

For information on creating step or label names, refer to STEP in this chapter and to colon (:) in the Symbols chapter.

Example 1

GOTO S1

Interpretation: The task is directed to step or label S1.

Example 2

```
STEP ONE
IF TEMP1 > 72 THEN GOTO TWO
TURNOFF FAN
GOTO ONE

STEP TWO
TURNON FAN
GOTO ONE
```

Interpretation: If TEMP1 is greater than 72 degrees when this line is executed, the execution of the task is directed to step TWO.

Usage Rules

1. GOTO may appear anywhere in a task, including the first line after the task name.

2. The name of a single step or label within the same task must immediately follow the word GOTO.

HALT

This reserved word is used in BEST^{++} statements to stop the execution of an active task in the assigned Comfort Controller.

The task is stopped immediately. The task does not resume execution until it is reactivated by a HALT statement.

HALT is similar to the reserved word EXIT, which is discussed earlier in this chapter.

Syntax

HALT *x*

where *x* is a task in the assigned Comfort Controller.

Example

HALT CHILLER

Interpretation: A task named CHILLER is stopped.

Usage Rules

1. HALT must be immediately followed by a single task name.

2. Before HALTing another BEST^{++} task, you must first CONNECT that task in the global dictionary. For example:

```
CONNECT PUMP_TASK AS A TASK {PMPTSK}
TASK ONE
HALT PUMP_TASK
ENDTASK
```

HOUR

The reserved word HOUR is used in BEST^{++} statements to obtain the numerical hour of the day. Whenever it is encountered in a task, a value from 0 to 23 is provided. The value may then be compared to a variable or constant, or it may be assigned to a variable.

The value assigned to HOUR is maintained by the internal clock of the Comfort Controller.

Numerical values correlate to the hour of the day as follows:

0	=	midnight	12	=	noon
1	=	1 a.m.	13	=	1 p.m.
2	=	2 a.m.	14	=	2 p.m.
3	=	3 a.m.	15	=	3 p.m.
4	=	4 a.m.	16	=	4 p.m.
5	=	5 a.m.	17	=	5 p.m.
6	=	6 a.m.	18	=	6 p.m.
7	=	7 a.m.	19	=	7 p.m.
8	=	8 a.m.	20	=	8 p.m.
9	=	9 a.m.	21	=	9 p.m.
10	=	10 a.m.	22	=	10 p.m.
11	=	11 a.m.	23	=	11 p.m.

Syntax

HOUR relational operator *x*

where *x* is any variable or constant between 0 and 23.

Alternate Syntax

x relational operator HOUR
x = HOUR

Example 1

IF HOUR > 7 THEN TURNON FAN1 ENDIF

Interpretation: FAN1 is turned on each day at or anytime after 7 a.m.

Example 2

COUNTIME = HOUR

Interpretation: The variable COUNTIME is assigned the current value of HOUR.

Usage Rules

1. HOUR cannot be assigned a value.

   For example: HOUR = 3 is improper usage. Apart from this, HOUR may appear anywhere in a line of a task.

   Note: HOUR EQ 3 could be used as a condition to be tested in an IF... THEN... statement. This is because you are not actually assigning a value *to* HOUR. You are only testing to see if HOUR *is equal* to a value.

2. HOUR must be preceded or followed by one of the following operators:

   < < = > > = EQ < >

3. If HOUR precedes one of the above operators, then a variable, constant, or mathematical expression must immediately follow the operator.

4. HOUR cannot be used as a variable. In order to operate on the value of HOUR it must be assigned to a variable.

   For example: X = HOUR.

IF... THEN... [ELSE...]ENDIF

These reserved words are used in BEST^{++} statements to test or evaluate a condition before proceeding to take a specified action. The condition may be a relational or logical expression. IF... THEN... provides task control as follows:

If the condition is TRUE, then the action is completed. If the condition is FALSE, the action is not completed and the task will read the ELSE statement if one exists. If an ELSE statement does not exist, the next line after ENDIF is executed.

Syntax

IF *x* THEN *y* [ELSE statement] ENDIF

where *x* is any logical or relational expression.

y is any statement(s).

ELSE is an optional statement(s) that executes only when the IF statement is false.

ENDIF is a reserved word that defines the end of the IF statement.

Example 1

IF TEMP1 > 72 THEN RUN CHILLER ENDIF

Interpretation: If the value of TEMP1 is greater than 72 degrees when this line is executed, then a task named CHILLER is activated. If the value of TEMP1 is less than or equal to 72 degrees, then the task named CHILLER is not activated and program execution continues with the statement immediately following ENDIF.

Example 2

IF TEMP1>72 THEN TURNON PUMP1 ELSE TURNOFF PUMP1 ENDIF

Interpretation: If the value of TEMP1 is greater than 72 degrees when this line is executed, then PUMP1 is turned on. If the value of TEMP1 is less than or equal to 72 degrees, PUMP1 is turned off.

Example 3

IF (TEMP1 > 72) OR (TEMP2 > 72) OR (TEMP3 > 72) THEN GOTO S4 ENDIF

Interpretation: If TEMP1, TEMP2, or TEMP3 is greater than 72 degrees when this line is executed, then the task is directed to S4 of the same program. If all of the TEMPs are less than or equal to 72 degrees, then the task does not go to S4. Instead, it executes the statement immediately following ENDIF.

Example 4

IF TEMP1 > 72 THEN PSI = 16; TURNON PUMP1; TURNOFF PUMP2 ENDIF

Interpretation: If the value of TEMP1 is greater than 72 degrees when this line is executed, then PSI is assigned the value 16, PUMP1 is turned on, and PUMP2 is turned off. If the value of TEMP1 is less than or equal to 72 degrees, then the values of PSI, PUMP1, and PUMP2 remain unchanged.

Usage Rules

1. IF must begin a condition statement and must be immediately followed by a logical expression.
2. The word THEN must follow the IF... expression. No other statements should be inserted before the THEN expression.
3. When multiple conditions are to be tested, IF is stated only once, with each additional condition included in the test by use of one of the following words: AND, OR, or NOT.
4. The ELSE statement is optional. It executes only when the IF statement is false. If no action is required, the ELSE statement can be omitted.
5. ENDIF must terminate all IF... THEN... statements.
6. You can nest up to twenty IF... THEN... ENDIF statements. Refer to the example below.

```
IF TEMP1 > 72 THEN
        IF TEMP2 > 72 THEN
                TURNOFF PUMP2
        ENDIF
        TURNOFF PUMP1

        IF TEMP3 > 72 THEN
                TURNOFF PUMP3
        ENDIF
        TURNOFF PUMP4
ENDIF
```

7. When an IF THEN statement spans more than one line, you should place the conditional arguments in parentheses ( ).

INCREMENT

This reserved word is used to increase the value of a counter by one.

Syntax

INCREMENT *x*

where *x* is the counter name.

Example

INCREMENT COUNTER1

Interpretation: When BEST^{++} encounters this line, it increases the current value of the counter named COUNTER1 by one.

See Also

COUNTER, DECREMENT, RESET.

Usage Rules

1. INCREMENT can appear at the beginning of a line, after a THEN statement, or after a semicolon (;).

2. The name of a counter must immediately follow the word INCREMENT.

3. The counter specified in the INCREMENT statement must be in the same controller as the INCREMENT statement.

INPUTFROM

This reserved word is used in BEST^{++} statements to read the value of variables that reside in another system element. INPUTFROM is only valid with variables that have been connected using NETWORK_CONNECT, NETWORK_POINT and NETWORK_DECISION.

Syntax

INPUTFROM *x*

operation on *x*, where *x* is the name of an externally CONNECTed variable.

Example

```
NETWORK_POINT OAT     {12,0,OAT}

INPUTFROM OAT

IF OAT.STATUS EQ 0 AND OAT >= 75 THEN
    TURNOFF FAN
ENDIF
```

where:
- 0 = successful communication
- 1 = communication is ok, but data is invalid (out of range, not communicating to Communication bus, or hardware error)
- 2 = Communication ok but couldn't find variable

Interpretation: A connect was made to point OAT of System Element #12 on Bus #0. The INPUTFROM causes a network message to be generated to read OAT. When the response has been received, if the communication status of OAT (OAT. STATUS) is good and the value is >= to 75, the fan is turned off.

Usage Rules

1. INPUTFROM is only valid with variables that are externally CONNECTed.

2. INPUTFROM cannot appear on the right side of an assignment. The following example is invalid:

   x = INPUTFROM OAT

3. The ?VALUE and ?STATUS flags previously used in BEST are no longer required.

This reserved word is used in a BEST^{++} statement to return the

LOG

natural logarithm of a numeric expression. LOG is the inverse of EXP.

Syntax

LOG *x*

where *x* is any constant, variable, or mathematical expression greater than zero.

Example

y = LOG (e)

Interpretation: When BEST^{++} encounters this line, it calculates *y* as 1, where e = 2.71828.

Usage Rules

The value of *x* should be greater than zero. If the value of *x* is less than zero, the result of LOG *x* will equal zero.

This reserved word is used in a BEST^{++} statement to return the

LOG10

logarithm to the base 10 of a number.

Syntax

LOG10 *x*

where *x* is any constant, variable, or mathematical expression greater than zero.

Example 1

POWER = LOG10 5

Interpretation: When BEST^{++} encounters this line, it calculates the answer as $10^{POWER} = 5$.

Example 2

POWER = LOG10 Y

Interpretation: When BEST^{++} encounters this line, it calculates the answer as $10^{POWER} = Y$.

Usage Rules

The value of *x* should be greater than zero. If the value of *x* is less than zero, the result of LOG10 *x* will equal zero.

This reserved word is used in a BEST^{++} statement to define the

LOOP

beginning of a group of statements that will be executed together, in sequence, for a specified number of times.

Syntax

LOOP *x* FROM *y* TO *z*
ENDLOOP *x*

where *x* is the name of the loop.
y is the start value.
z is the end value.

Example

```
LOOP ONE FROM 1 TO 10
IF ONE > 5 THEN TURNON FAN[ONE]
ELSE TURNOFF FAN[ONE]
ENDIF
ENDLOOP ONE
```

Interpretation: When BEST^{++} encounters the LOOP statement, it executes the lines of the task between the LOOP and ENDLOOP statements ten times. It then executes the ENDLOOP statement.

See Also

ENDLOOP

Usage Rules

1. A loop name can only be used once in any BEST^{++} program.
2. The loop count value may be read by referencing the loop name.
3. You cannot write a value to a loop name.
4. You must terminate a loop with an ENDLOOP statement.
5. The loop name must immediately follow LOOP and ENDLOOP.
6. You can nest up to twenty LOOP statements. Refer to the example below.

```
LOOP I FROM 1 TO 8
    IF PUMP[I] EQ 1 THEN
        LOOP J FROM 1 TO 8
                TURNON PUMP[J]
        ENDLOOP J
    ENDIF
ENDLOOP I
```

This reserved word is used to return the maximum of two values.

MAX

Syntax

MAX (x,y)

where *x* and *y* are any constant, variable or mathematical expression.

Example 1

MAX (x,3)

Interpretation: If *x* is larger than 3, then the value of MAX is equal to *x*. Otherwise, the value of MAX is 3.

Example 2

MAX (-Y,2*Z)

Interpretation: If -Y is larger than 2 * Z, then the value of MAX is equal to -Y. Otherwise, the value of MAX is 2 * Z.

The reserved word MINUTE is used in BEST^{++} statements to obtain

MIN

Syntax

MIN (x,y)

where *x* and *y* are any constant, variable or mathematical expression.

Example 1

MIN (X,3)

Interpretation: If *x* is smaller than 3 then the value of MIN is equal to *x*. Otherwise, the value of MIN is 3.

Example 2

MIN (-Y,2*Z)

Interpretation: If -Y is smaller that 2 * Z then the value of MIN is equal to -Y. Otherwise, the value of MIN is 2 * Z.

The reserved word MONTH is used in BEST^{++} statements to obtain

MINUTE

the numerical value for the minute of the hour. Whenever this word is encountered in a task, a value from 0 to 59 is provided. This value may then be compared to a variable or constant, or it may be assigned to a variable.

The value assigned to MINUTE is maintained by the internal clock of the Comfort Controller.

Syntax

MINUTE relational operator *x*
or
x relational operator MINUTE
or
x = MINUTE
or
y = MINUTE math operator *x*

where *x* is any variable or constant from 0 to 59.

Example 1

IF MINUTE > 15 THEN TURNOFF FAN1 ENDIF

Interpretation: If the number of minutes past the hour is greater than 15 when this line is executed, then FAN1 is turned off.

Example 2

DELTIME = MINUTE * 2

Interpretation: The variable DELTIME is assigned twice the value of MINUTE when this line is executed.

Example 3

IF (HOUR EQ 8) AND (MINUTE EQ 15) THEN TURNON FAN1 ENDIF

Interpretation: If the time is anywhere from 8:15:00 to 8:15:59 when this line is executed, then FAN1 is turned on.

1. MINUTE cannot be assigned a value.

Usage Rules

For example: MINUTE = 3 is improper usage. Apart from this, MINUTE may appear anywhere in a line of a task.

Note: MINUTE EQ 3 could be used as a condition to be tested in an IF... THEN... statement. This is because you are not actually assigning a value *to* MINUTE. You are only testing to see if MINUTE *is equal* to a value.

2. MINUTE must be preceded or followed by one of the following operators:

   < < = > > = EQ < >

3. If MINUTE precedes one of the above operators, then a variable, constant, or mathematical expression must immediately follow the operator.

4. MINUTE cannot be used as a variable. In order to operate on the value of MINUTE it must be assigned to a variable. For example: X = MINUTE.

This reserved word is used to return the minimum of two values.

MONTH

the numerical value of the month of the year. Whenever this word is encountered, a value from 1 to 12 is provided. This value may then be compared to a variable or constant, or it may be assigned to a variable.

The value assigned to MONTH is maintained by the internal calendar of the Comfort Controller.

Syntax

MONTH relational operator *x*
or
x relational operator MONTH
or
x = MONTH
or
y = MONTH math operator *x*

where *x* is any variable or constant from 1 to 12.

Example 1

IF MONTH > 9 THEN TURNON PUMP1 ENDIF

Interpretation: PUMP1 is turned on when this line is executed after September.

Example 2

IF (MONTH < 4) AND (DOW < > 6) THEN TURNON PUMP1 ENDIF

Interpretation: PUMP1 is turned on every day except Saturday during the months of January, February, and March.

Example 3

y = MONTH * 2

Interpretation: The variable *y* is assigned twice the value of MONTH when this line is executed.

1. MONTH cannot be assigned a value.

Usage Rules

For example: MONTH = 3 is improper usage. Apart from this, MONTH may appear anywhere in a line of a task.

Note: MONTH EQ 3 could be used as a condition to be tested in an IF... THEN... statement. This is because you are not actually assigning a value *to* MONTH. You are only testing to see if MONTH *is equal* to a value.

2. MONTH must be preceded or followed by one of the following operators:

   < < = > > = EQ < >

3. If MONTH precedes one of the above operators, then a variable, constant, or mathematical expression must immediately follow the operator.

4. MONTH cannot be used as a variable. In order to operate on the value of MONTH it must be assigned to a variable. For example: X = MONTH.

NOT

This logical operator is used in BEST^{++} statements to invert a logical condition, or to test for the inverse of a logical condition. This operator may be used in conjunction with other logical operators to form one statement.

For analog variables, BEST^{++} interprets any number other than zero as TRUE. For discrete variables, ON = TRUE, OFF = FALSE.

Syntax

NOT *x*

where *x* is a condition or variable.

Truth Table

x	NOT *x*
T	F
F	T

Example 1

IF NOT PUMPSTAT THEN RUN PUMPCHEK ENDIF

Interpretation: If PUMPSTAT is not ON when this line is executed, then a task named PUMPCHEK is activated.

Example 2

IF ((NOT FANSTAT1 OR NOT FANSTAT2) AND NOT FANOFF) THEN RUN FANSRUN ENDIF

Interpretation: If either FANSTAT1 or FANSTAT2, or both are not ON, and FANOFF is OFF when this line is executed, then a task named FANSRUN is activated.

Example 3

INVX = NOT X

Interpretation: If variable X is nonzero, then variable INVX will be set to zero. If X is zero, then INVX will be set to 1.

Usage Rule

A variable or constant must immediately follow the logical operator NOT.

OR

This logical operator is used in BEST^{++} statements to link two or more logical conditions.

For analog variables, BEST^{++} interprets a zero as FALSE, and any number other than zero as TRUE.

x OR *y*

where *x* and *y* are any constant, variable, or expression

Truth Table

x	*y*	*x* OR *y*
T	T	T
F	T	T
T	F	T
F	F	F

Example

IF (TEMP1 > 72) OR (TEMP2 > 72) THEN RUN CHILLER
ENDIF

Interpretation: If either TEMP1 or TEMP2 is greater than 72 degrees when this line is executed, then a task named CHILLER is activated.

Usage Rules

1. One logical condition must immediately precede the logical operator OR.

2. One logical condition must immediately follow the logical operator OR.

OUTPUTTO

This reserved word is used in BEST^{++} statements to write to (or change the value or status of) variables that reside in another system element. OUTPUTTO is only valid with variables that have been externally CONNECTed using NETWORK_CONNECT, NETWORK_POINTS, AND NETWORK_DECISION.

Syntax

OUTPUTTO *x*

where *x* is the name of an externally CONNECTed variable.

Example

```
NETWORK_POINT FAN     {12,0,FAN}

IF OAT >= 75 THEN
    TURNOFF FAN
    OUTPUTTO FAN
ENDIF
```

Interpretation: A connect was made to an external point called FAN of System Element #12 on Bus #0. If the internal variable, OAT, is >= to 75 then the FAN value is set to *0*. The OUTPUTTO causes a network message to be generated to write the value 0 to FAN.

Usage Rules

1. OUTPUTTO is only valid with variables that are externally CONNECTed.

2. OUTPUTTO cannot appear on the left side of an assignment. The following example is invalid:

   OUTPUTTO FAN = 1.

3. The ?VALUE and ?STATUS flags previously used in BEST are no longer required.

PID

This math function provides the **P**roportional, **I**ntegral, **D**erivative control loop. The control loop allows configurable units for the output value and control inputs. The iteration time for the control loop is adjustable.

Syntax

PID *x*
(*enable*, *setpoint*, *sensor_input*, *min_out*, *max_out*, *integrator_clamp*, *reset_integrator*)
{*name*, *"description"*, *units*, *kp*, *ki*, *kd*, *dsblout*, *minout*, *maxout*, *startval*, *blkrate*}

where *x* is the name of the PID control loop.

the following parameters may be constants, variables or mathematical expressions:

enable allows the PID to execute when *enable* = 1.

setpoint defines the setpoint that the PID will calculate to.

sensor_input is the point name that is being controlled.

min_out causes the PID to go to its minimum output whenever it is not equal to zero.

max_out causes the PID to go to its maximum output whenever it is not equal to zero.

integrator_clamp causes the integrator to be clamped whenever it is not equal to zero.

reset_integrator causes the integrator to be reset whenever it is not equal to zero.

The following parameters define the PID initialization:

name is a user-configurable, up to 8 character name for the PID. Default: *PID*

description is a user-configurable, up to 24 character description for the PID. Default: *"PID"*

units is the PID's engineering units. Default: *0*

kp is the PID's proportional gain (-100 to 100). Default: *0*

ki is the PID's integral gain (-100 to 100). Default: *0*

kd is the PID's derivative gain (-100 to 100,0). Default: *0*

dsblout is the PID's disabled output value (-9999.99 to 9999.99). Default: *0*

minout is the PID's minimum output value (-9999.99 to 9999.99). Default: *0*

maxout is the PID's maximum output value (-9999.99 to 9999.99). Default: *0*

startval is the PID's starting value (-9999.99 to 9999.99). Default: *2*.

blkrate is the block iteration rate (10 to 300). The Default: *120*.

Example

```
PID COOL_PID
    (NOT (space_temp.STATUS),        ~ PID enabled if feedback sensor is OK
    sp_setpoint.STPTHI,              ~Setpoint
    space_temp,                      ~ Control to space temperature
    ,,,                              ~ minimum, maximum reference and
                                     ~ clamp integrator
    COOLINGVALUE.FORCE OR            ~ integrator reset
    COOLINGVALUE.STATUS OR
    (NOT (FANST)))
    {,"PID Loop", 1, 10, 1.0, 0, 150, 45, 150, 55, 120}

COOLINGVALUE = COOL_PID
```

Interpretation: A PID defined as COOL_PID is enabled if the STATUS maintenance decision for the point defined as *space_tem*p is 0 (OK). The PID uses the high value of the space setpoint to control *space_temp*. The integrator is reset if COOLINGVALUE is forced or has a bad status or if FANST is not on. The output from the PID is sent to COOLINGVALUE.

Usage Rules

1. You must type a name after PID.
2. If you do not type a description, the BEST^{++} compiler will use the default name of PID, which is "PID".
3. If the description uses spaces or any reserved characters (=, -, etc.), then the description must be enclosed in quotes (“).

POWER

This reserved word is used to raise an indicated value (called the mantissa) by a specified power.

Syntax

POWER (*x*,*y*)

where *x* is the mantissa, and
y is the exponent.

Example

SQRTTEN = POWER (10,.5)

Interpretation: When BEST^{++} encounters this statement, it calculates the value of SQRTTEN as 3.1622777.

See Also

EXP and ^

PROGRAM

This reserved word defines the name of a BEST^{++} custom program. BEST^{++} automatically inserts the PROGRAM statement when you create a new program. It is not necessary to edit the PROGRAM statement.

When tasks are related, you may want to group them into one program, although it is not necessary. Although a program can consist of several tasks, each task must begin with a TASK statement and end with an ENDTASK statement.

Syntax

PROGRAM *x* {*name*, *description*, *units*}

where *x* is the user-defined BEST^{++} custom program name.

name is a user-defined, up to 8 character, uppercase name of a BEST^{++} custom program.

description is a user defined, up to 24 character description for the program.

units is the engineering unit name or number for the program. The default is 0, which is "none."

Example

PROGRAM chiller {CHILLER, “Chiller Program”}

Interpretation: A BEST^{++} program called chiller is defined. Engineering units will not be displayed.

Usage Rules

1. PROGRAM must be the first word in any BEST^{++} program.
2. A PROGRAM statement can be preceded by a REM statement.

RELEASE

This reserved word is used in BEST^{++} statements to remove the force on a variable in another system element. RELEASE is only valid with variables that have been externally CONNECTed using either NETWORK_POINT or one of the three external FID BEST CONNECTs.

Syntax

RELEASE *x*

where *x* is the name of a point connected using either NETWORK_POINT or one of the three external FID BEST CONNECTs.

Example

```
NETWORK_POINT FAN     {12,0,FAN}

IF OAT >= 75 THEN
    TURNOFF FAN
    OUTPUTTO FAN
ELSE
    RELEASE FAN
ENDIF
```

Interpretation: A CONNECT was made to an external point called FAN in System Element #12 on Bus #0. If the internal variable, OAT, is >= to 75 then the FAN value is set to *0*, otherwise, the RELEASE causes a network message to be generated to AUTO the FAN.

Usage Rule

RELEASE can only remove a BEST^{++} force or a force of lower priority on variables residing in a system element other than the one containing the task with the RELEASE statement.

REM

The reserved word REM is used to include remarks or sections of descriptive text within a source file. Remarks can be inserted into any source file to help identify the purpose, actions, flow, and constraint activities of a task. There is no limit to the number of REM statements that can be used and there are no restrictions to the characters that may comprise a remark.

Remarks are best used when they can serve as useful information to others who might use the source file. It is beneficial to explain such things as the purpose of the task and the variables, or to define how the steps are used. Any other explanations should also be included to help others understand the theory behind the task.

The BEST^{++} language allows two forms of usage for remark statements. These two forms are the reserved word REM and the tilde (~). For information on the tilde, refer to the Symbols chapter.

Note: Remarks only appear in the source file. When you compile the BEST^{++} program, the BEST^{++} compiler removes all remarks. Remarks are not downloaded to the Comfort Controller as part of the program.

Syntax

REM *x*

where *x* is the descriptive text (the remark).

Example

```
REM     THIS PROGRAM WAS WRITTEN ON 2/1/96 TO TEST
REM     THE  OPERATION OF FAN1

IF TEMP1 > 72 THEN TURNON FAN1 ENDIF

REM     CHANGE TEST VALUE TO 78 IN SUMMER

IF TEMP1 < 72 THEN TURNOFF FAN1 ENDIF
```

Interpretation: Three lines of descriptive text accompany a display or printout of this program. Remarks are not sent to the controller and will therefore be omitted when the program is uploaded and decompiled.

Usage Rules

1. It is not necessary to separate the remark statements from other statements with a blank line. Doing so is purely cosmetic.

2. There must be at least one space between REM and the descriptive text.

3. A REM statement does not have to be the only statement on a line, but it must be the last statement on a line. For example:

```
x = 72          REM  x should never be less than 65.
```

REMAIN

This reserved word is used to calculate the remainder of *x* divided by *y*.

Syntax

REMAIN (*x*,*y*)

where *x* and y are any constant, variable, or mathematical expressions greater than zero.

Example

REMAIN (10,3)

Interpretation: When BEST^{++} encounters this line, it calculates the remainder of 10 divided by 3 as 1, i.e., 10/3 = 3 remainder 1.

REPEAT

This reserved word is used to define the end of a step for the purpose of repeating the step. It is always used together with the reserved word STEP, which defines the beginning of the step.

All statements entered between the words STEP and REPEAT execute sequentially for as long as the task is directed to remain at the named step.

Syntax

REPEAT

Example

```
STEP ONE
IF PMP1 EQ 1 THEN TURNON PMP2 ENDIF
:TWO
DELAY PUMPS FOR 20
REPEAT
```

Interpretation: Step ONE of a task named PUMPS says that if PMP1 is ON, then PMP2 is turned on. The task then delays for 20 seconds. This step continues to repeat. The task can only exit from the step if a GOTO, EXIT, or HALT statement is included in the step.

Usage Rules

1. REPEAT must always be used together with STEP.

2. REPEAT causes the task to return to the line below the last occurrence of the reserved word STEP and ignores any labels that may be in between.

RESET

This reserved word is used in BEST^{++} statements to assign the value *0* to a specified timer or counter that is located in the same Comfort Controller. When a RESET statement is encountered, the named timer or counter is immediately reset to *0*, whether or not it is active.

Timers are typically created to accumulate runtime and provide delays. Counters are typically created to increment or decrement as directed by the task.

The value of the timer or counter prior to the RESET is not saved, unless the current value is assigned to another variable before the RESET occurs.

Syntax

RESET *x*

where *x* is the name of a single timer or counter.

Example 1

IF FANTIMER > 3600 THEN RESET FANTIMER ENDIF

Interpretation: If the value of a timer called FANTIMER becomes greater than 3600 seconds, it is reset to zero.

Example 2

COUNTER FANCOUNT
IF FANCOUNT > 10 THEN RESET FANCOUNT ENDIF

Interpretation: If the value of a counter called FANCOUNT is greater than 10, then FANCOUNT is reset to *0*.

Usage Rules

1. RESET may appear at the beginning of a line, or after a THEN or a semicolon (;).

2. The name of a single timer or counter must immediately follow the word RESET.

3. The timer or counter specified in the RESET statement must be in the same Comfort Controller as the statement.

RETURN

This reserved word is used in BEST^{++} subroutines to terminate subroutine execution and pass control back to the task that invoked the subroutine.

Syntax

RETURN (*arg*)

where *x* is the constant, variable, or statement RETURNed to the CALLing task. The argument is optional.

Example

```
~  SUBROUTINE to filter a value between a and b
~        x is the value to be filtered
~        if x is less than a then result is 0
~        x increases proportionally from 0 to c between a and b
~        if x is greater than b the result is c
SUBROUTINE FILTER (x,a,b,c)
        IF (x < a) THEN
                result = 0
        ELSE
                IF (x > b) THEN
                        result = c
                ELSE
                        result = (b - a) * (x - a) / c
                ENDIF
        ENDIF
RETURN (result)
TASK subtest
input = 7
filteredinput = CALL FILTER (input,-5,13.7,100)
ENDTASK
```

Interpretation: Refer to the remarks in the example.

See Also

CALL, SUBROUTINE

Usage Rule

If you do not use the argument, you must still type ( ) after the name of the subroutine.

ROUNDDOWN

This function is used to express a numerical value as the nearest whole number (integer) that is less than the stated value.

Syntax

ROUNDDOWN *x*

where *x* is any constant, variable, or mathematical expression.

Example 1

A = ROUNDDOWN 3.25

Interpretation: When BEST^{++} encounters this line, the variable A is set equal to 3 (3.25 down).

Example 2

B = ROUNDDOWN (3.5 * (10-5))

Interpretation: When BEST^{++} encounters this line, the variable B is set equal to 17.5 (the result of the mathematical expression 3.5*(10-5) rounded down).

See Also

ROUNDUP

Usage Rules

1. *x* can be any positive or negative number.

2. A negative number is ROUNDDOWN to the next lowest negative integer value, i.e., ROUNDDOWN (-17.5) = -18.

ROUNDUP

This function is used to express a numerical value as the nearest whole number (integer) that is greater than the stated value.

Syntax

ROUNDUP *x*

where x is any constant, variable, or mathematical expression.

Example 1

A = ROUNDUP 3.25

Interpretation: When BEST^{++} encounters this line, the variable A is set equal to 4 (3.25 rounded up).

Example 2

B = ROUNDUP (3.5 * (10-5))

Interpretation: When BEST^{++} encounters this line, the variable B is set equal to 18 (the result of the mathematical expression 3.5*(10-5) rounded up).

See Also

ROUNDDOWN

Usage Rules

1. *x* can be any positive or negative number.

2. A negative number is ROUNDUP to the next highest negative integer value, i.e., ROUNDUP (-17.5) = -17.

RUN

This reserved word is used in BEST^{++} statements to activate one or all of the BEST^{++} tasks in a Comfort Controller.

If the task commanded to run is inactive, it is immediately activated, and execution begins at the first statement.

If RUN is applied to a currently active task, the task will continue to run normally. The RUN statement will have no affect on the task.

Syntax

RUN *x*

where *x* is a single task name.

Example

RUN CHILLER

Interpretation: A task named CHILLER is activated at its first statement. If the task is already active, execution continues and is not effected. If the task is not active, execution begins from the first statement of the task.

Usage Rule

If the task you wish to run is not in the same program (source file) as the RUN statement, you must CONNECT the task before RUNning it.

SECOND

This reserved word is used in BEST^{++} statements to obtain the numerical value for the second of the minute. Whenever this word is encountered, a value from 0 to 59 is provided. This value may then be compared to a variable or constant, or it may be assigned to a variable.

The value assigned to SECOND is maintained by the internal clock of the Comfort Controller.

Syntax

SECOND relational operator *x*
or
x relational operator SECOND
or
x = SECOND
or
y = SECOND math operator *x*

where *x* is any variable or constant.

Example 1

IF SECOND > 10 THEN TURNON FAN1 ENDIF

Interpretation: If the number of seconds past the minute is 11 or greater when this line is executed, then FAN1 is turned on.

Example 2

IF KW > LIMIT THEN COUNT = SECOND ENDIF

Interpretation: If KW is greater than LIMIT when this line is executed, then the variable COUNT is assigned the value of SECOND.

Example 3

y = SECOND * 2

Interpretation: The variable *y* is assigned twice the value of SECOND when this line is executed.

Usage Rules

1. SECOND cannot be assigned a value.

   For example: SECOND = 3 is improper usage. Apart from this, SECOND may appear anywhere in a line of a program.

   Note: SECOND EQ 3 could be used as a condition to be tested in an IF... THEN... statement. This is because you are not actually assigning a value *to* SECOND. You are only testing to see if SECOND *is equal* to a value.

2. SECOND must be preceded or followed by one of the following operators:

   < < = > > = EQ < >

3. If SECOND precedes one of the above operators, then a variable, constant, or mathematical expression must immediately follow the operator.

4. SECOND cannot be used as a variable. In order to operate on the value of SECOND it must be assigned to a variable. For example: *x* = SECOND.

SIN

When used in a BEST^{++} statement, the SIN (sine) function provides the sine of an angle expressed in radians.

Syntax

SIN *x*

where *x* is a constant, variable, or mathematical expression.

Example

RATIO = SIN (X - .3)

Interpretation: When BEST^{++} encounters this line in the program, it subtracts .3 from *x* and calculates the sine of the angle as a number between -1 and 1.

SINH

When used in a BEST^{++} statement, the SINH (hyperbolic sine) function provides the hyperbolic sine of a number. The formula for calculating SINH is:

SINH (*x*) = (EXP (*x*) - EXP (-*x*)) / 2

Syntax

SINH *x*

where *x* is a constant, variable, or mathematical expression.

Example

HYPERSINE = SINH (.3)

Interpretation: When BEST^{++} encounters this line, it calculates the hyperbolic sine of .3 as 0.3045.

SQRT

This function is used in BEST^{++} statements to calculate the positive square root of the constant, variable, or mathematical expression immediately following it. The square root of positive real or integer values can be calculated.

Syntax

SQRT *x*

where *x* is any positive constant, variable, or mathematical expression.

Example 1

SQRT 16

Interpretation: The square root of 16 (4) is calculated.

Example 2

SQRT VP

Interpretation: The square root of a variable named VP is calculated.

Example 3

TVAL = SQRT (TEMP1 - TEMP2)

Interpretation: TVAL is assigned the value of the square root of the difference between TEMP1 and TEMP2.

Usage Rules

1. SQRT may appear anywhere in a line of program.
2. If SQRT is followed by a mathematical expression, the use of parentheses ensures proper order of evaluation.
3. Attempting to calculate the square root of a negative number results in a value of 0.

START

This reserved word is used in BEST^{++} statements to activate a specified timer that is located in the same Comfort Controller. The named timer immediately starts when the command is issued.

Timers are used to accumulate a length of time in seconds. Typically, they are created to accumulate runtime.

The timer starts counting from its last value. The value of a stopped timer is retained in the timer until it is set to zero with the RESET command or until it is started again.

Note: This command is only used to start timers created using the reserved word TIMER. It does not turn on output devices.

Syntax

START *x*

where *x* is any timer.

Example

START TIMER1

Interpretation: A timer named TIMER1 will commence counting, beginning from the last value in the timer.

Usage Rules

1. START may appear anywhere in a line of program.
2. START can only be applied to a single timer at a time.
3. The timer that is started must be in the same Comfort Controller as the BEST^{++} program that starts it.
4. If *x* is an active timer, then START *x* has no effect.

STEP

This reserved word is used in BEST^{++} statements to define the beginning of a portion of a program. It is good practice to break up long programs into smaller parts, or steps.

STEP is typically used with REPEAT. All statements entered between the word STEP and the word REPEAT are executed sequentially for as long as the program is directed to remain at the named step.

The only way to have the program execute statements outside of a step that includes the word REPEAT is to include a GOTO in the step.

A step name can also be used as the target of a GOTO statement.

Note: You can also use labels (designated by the symbol *:*) to define sections of a program and as a target of a GOTO statement. For information on labels, refer to the Symbols chapter.

Syntax

STEP *x*

where *x* is the name of the step.

Example 1

```
TASK RUNCOUNT
STEP ONE
COUNTER = COUNTER + 1
DELAY RUNCOUNT FOR 60
REPEAT
```

Interpretation: In RUNCOUNT the variable COUNTER is incremented by one and the program is delayed for 60 seconds. Then, the REPEAT statement returns the execution sequence to the previous step name, which is STEP ONE.

Example 2

```
TASK CNTR
STEP SIX
COUNTER = COUNTER + 1
DELAY CNTR FOR 10
IF COUNTER < 60 THEN GOTO SIX
```

Interpretation: In CNTR the counter is incremented by one, and the program is delayed for 10 seconds. If the value of COUNTER is less than 60, then the program execution is directed back to STEP SIX by the GOTO statement. When the value of COUNTER is 60 or greater, the program execution continues to the next statement in the program sequence.

Usage Rules

1. STEP must be the only reserved word in a line of program.

2. Step names must conform to the rules for variable names.

STOP

This reserved word is used in BEST^{++} statements to deactivate a specified TIMER located in the same Comfort Controller. The named TIMER immediately stops when the command is issued. STOP is used in conjunction with START.

Timers are used to accumulate time in seconds. Typically, they are created to accumulate runtime.

The value of a stopped timer is retained in the timer until it is started again or reset.

Note: This command is used only to stop timers created using the reserved word TIMER. It does not turn off output devices.

Syntax

STOP *x*

where *x* is the name of a timer.

Example

STOP TIMER1

Interpretation: A timer named TIMER1 ceases counting, remaining at the last value accumulated unless it is subsequently reset to zero.

Usage Rules

1. STOP may appear anywhere in a line of program.
2. STOP can only be applied to a single timer at a time.
3. The timer must be in the same Comfort Controller as the program that stops it.

SUBROUTINE

This reserved word defines the beginning of a subroutine. A subroutine is a subprogram that consists of a series of statements that may be required in several places in a task or tasks. Instead of typing the commands several times, a subroutine can be written and invoked whenever needed.

Syntax

SUBROUTINE *x* (*arg1,arg2,arg3,arg4*)

where *x* is the name of the SUBROUTINE.
arg1, arg2, arg3, and *arg4* are values passed to the subroutine by the CALL statement.

Example

```
~  SUBROUTINE to filter a value between a and b
~        x is the value to be filtered
~        if x < a then result equals b
~        if x > = a then result equals c

SUBROUTINE FILTER (x,a,b,c)
        IF x < a THEN
                result = b
        ELSE
                result = c
                ENDIF
RETURN (result)
TASK subtest
input = 7
filteredinput = CALL FILTER (input,-5,13.7,100)
ENDTASK
```

Interpretation: Refer to the remarks in the example.

See Also

CALL, RETURN

Usage Rules

1. The SUBROUTINE must be either in the same program as the CALLing task (before the calling task) or in the global dictionary.
2. SUBROUTINE must be followed by a subroutine name.
3. A SUBROUTINE must end with a RETURN statement.
4. A subroutine executes as if it were a single statement. No other programs in the controller will execute until the subroutine returns to the calling task.

SWITCH

This function defines a named variable based on the value of a boolean operation or the result of a true or false condition. The variable (*n*) is set to the first variable or constant (*x*) if the boolean operation or true/false condition (*z*) is false. If *z* is true, *n* is set to *y*.

Syntax

SWITCH *n (x, y, z)*

where *n* is the name of the switch.
x and *y* are any constants or variables.
z is a boolean operation or the result of a true or false condition.

Example

SWITCH OFFSET_TEMP (-3, -10, LIMIT_TEMP > 60)
x = OFFSET_TEMP

Interpretation: When BEST^{++} encounters this line, *x* will be set to -10 if LIMIT_TEMP is greater than 60. If LIMIT_TEMP is less than or equal to 60, *x* will be set to -3.

TAN

When used in a BEST^{++} statement, the TAN (tangent) function provides the tangent of an angle.

Syntax

TAN *x*

where *x* is a constant, variable, or mathematical expression.

Example

RATIO = TAN (π/4) = TAN (.7853981)=1

Interpretation: When BEST^{++} encounters this line, it calculates the tangent of -π/4 as 1.

TANH

When used in a BEST^{++} statement, the TANH (hyperbolic tangent) function provides the hyperbolic tangent of a number.

Syntax

TANH *x*

where *x* is a constant, variable, or mathematical expression.

Example

HYPERTAN = TANH (.3)

Interpretation: When BEST^{++} encounters this line, it calculates the hyperbolic tangent of 0.3 as 0.29131.

TASK

This reserved word defines the beginning of a BEST^{++} task. A task is a sequence of statements used to perform particular actions or calculations. A statement is a combination of BEST^{++} reserved words, mathematical functions, operators, symbols, variables, and constants. Several tasks may be entered in one program.

Syntax

TASK *x*

where *x* is the name of the task.

Alternate Syntax

TASK *x*
{*TASKNAME,"description",priority,forcepri,reschrat,reschpor*}

where *TASKNAME* is the user-configurable, up to 8 character task name.

"description" is the user-configurable, up to 24 character task description. The default is "PROGRAM".

priority is a numeric value between 1 and 5, indicating the order in which tasks will execute. The highest priority is *1*. Tasks and functions with a priority of 1 will execute first. Tasks and algorithms with a priority of 5 will execute last.

You can assign different priorities to tasks within the same program. For example, you can write five tasks in one program and assign each task a different priority ranging from 1 to 5. If you assign the same priority to more than one task, BEST^{++} will execute Line 1 of each of the tasks, followed by Line 2 and so on. Once BEST^{++} executes all tasks of the same priority, it then executes tasks of the next highest priority.

If all the tasks have the same priority, a shorter task will execute more often.

forcepri is the level of the Comfort Controller force priority. For a list of force priorities and their descriptions, refer to Appendix A in this manual. The levels range from 1 to 10, with 1 being the highest level. The default is 8, which is a BEST^{++} force. For information on removing forces, refer to AUTO and RELEASE in this chapter.

When forcing variables over the CCN network, all writes will cause BEST forces (*forcepri* = 8) regardless of what *forcepri* is set to.

reschrat is the rate (in seconds) at which the task automatically runs.

reschpor is the amount of time that must elapse after a download or Power On Reset (POR) occurs before the task will repeat.

Example

```
CONNECT OAT AS A TEMP_INPUT {OAT}
CONNECT FAN AS A DISC_OUTPUT {FAN}

TASK SAMPLE

REM     THIS PROGRAM CONTROLS THE COOLING FAN

STEP ONE
IF OAT > 70 AND FAN EQ 0 THEN TURNON FAN; COUNT =
COUNT + 1 ENDIF
IF OAT < 68 AND FAN EQ 1 THEN TURNOFF FAN ENDIF
REPEAT

ENDTASK
```

Interpretation: Because SAMPLE follows TASK, the program's name is SAMPLE. Note that TASK is the first word of the first line of the task.

Usage Rules

1. TASK must be followed by a task name.
2. A task must end with an ENDTASK statement. (ENDTASK must be the last statement).

TIMER

This reserved word is used to create and name a clock (a timer). When the named timer is encountered, its current value can be read and then compared to a variable or constant, or assigned to a variable.

A timer can be used to accumulate runtimes or provide a time delay. Once the timer has been named, it can be started, stopped, or reset to 0 using the START, STOP, and RESET statements.

A timer can count up to 4,294,967,295 seconds and then automatically resets to 0 and continues incrementing.

Syntax

TIMER *x* {0}
where *x* is the timer name
{0} = seconds
{1} = minutes
{2} = hours
default = {0}

Example 1

```
TIMER PUMPRUN
IF PUMP1 EQ 1 THEN START PUMPRUN ENDIF
```

Interpretation: A timer named PUMPRUN is created. If PUMP1 is equal to 1 (ON), then the timer named PUMPRUN begins incrementing seconds from *0*.

Example 2

```
TIMER RTIM {1}
START RTIM
```

Interpretation: A timer named RTIM is created. RTIM is started and begins incrementing minutes from 0.

Usage Rules

1. A timer can be created anywhere in a program.
2. A timer is not incremented until the elapsed seconds, minutes, or hours has expired.
3. The word TIMER must be followed by the timer name.
4. Each timer name must be unique within a given program.
5. A timer must be started in order for it to begin incrementing.
6. A timer can be started repeatedly without affecting its count.

TURNON/TURNOFF

These two reserved words are used in BEST^{++} statements to command a specified discrete output device ON or OFF.

Note: The specified device can only be a properly connected internal discrete output point or network point.

BEST^{++} interprets the logical value of 1 to be equal to the word TURNON. It interprets the logical value of 0 to be equal to the word TURNOFF. (See Usage Rule 4 below.)

Caution: Do not confuse these statements with the START and STOP statements. START and STOP can only be applied to a timer.

Syntax

TURNON *x* or TURNOFF *x*

where *x* is a connected discrete output point in the same controller as the task. If the device is in another controller, the TURNON/TURNOFF command must be followed by an OUTPUTTO statement (refer to OUTPUTTO in this section).

Example

IF TEMP < 72 THEN TURNON PUMP1 ENDIF

Interpretation: PUMP1 is turned on if the temperature is less than 72 degrees.

Usage Rules

1. TURNON and TURNOFF may appear at the beginning of a line, or after THEN or a semicolon (;).

2. TURNON and TURNOFF must be immediately followed by a specified internally connected discrete output point or network point.

3. TURNON and TURNOFF can only be applied to one device at a time.

4. The following is an alternative to using TURNON and TURNOFF:

   FAN = 1 (turns FAN on)
   FAN = 0 (turns FAN off)

5. You cannot TURNON or TURNOFF a discrete input point.

UPDATE

This reserved word is used in BEST^{++} statements to cause a variable CONNECTed to a NETWORK_INPUT to read data or a variable CONNECTed to a NETWORK_OUTPUT to write data.

Syntax

UPDATE *x*

operation on *x*, where *x* is the name of an externally connected variable.

Example

```
CONNECT OAT as a NETWORK_INPUT {REMOTEOAT}

UPDATE OAT
IF OAT.STATUS EQ 0 AND OAT >75 THEN
        TURNOFF FAN
ENDIF
```

Interpretation: A CONNECT was made to point REMOTEOAT, which is a NETWORK_INPUT. The UPDATE causes the NETWORK_INPUT to read the data from its specified CCN point if the CCN rate has expired. If the CCN rate has not expired, the last read value is reused.

Usage Rules

1. UPDATE is valid only with NETWORK_INPUT and NETWORK_OUTPUT.

2. UPDATE cannot appear on the right side of an argument. The following is invalid:

   x = UPDATE OAT.

VARIABLE

This reserved word is a symbolic name that you specify to represent a value to be used in the program you are currently writing or editing. If you want to use the variable in more than one program and maintain its definition, you must define the variable in the global dictionary. This type of variable is a global variable.

Note: If you use the same variable name in multiple programs without defining it in the global dictionary, the variable can have a different meaning in each of the programs. This type of variable is a local variable.

Any name that you use in a BEST^{++} program that is not CONNECTed is assumed to be a variable.

A BEST^{++} variable can represent a value written with or without a decimal point. A value may be in customary US or metric engineering units. For information on how to specify engineering units, refer to Appendix B in this manual.

Syntax

VARIABLE *x {y}*

where *x* is the variable name, and
y is the initial value (starting value) of *x*.

Note: Including the initial value is optional. If you do not specify an initial value, BEST^{++} will use the default value of 0.

Example 1

VARIABLE NUM_CHILLERS {10}

Interpretation: When BEST^{++} encounters this line, it creates a variable named NUM_CHILLERS with an initial value of ten.

Example 2

VARIABLE NUM_CHILLERS

Interpretation: When BEST^{++} encounters this line, it creates a variable named NUM_CHILLERS with an initial value of zero.

WHEN

This reserved word is similar to the IF... THEN... statement, in that it tests for a specific condition. However, this statement prohibits further execution of the program until the expected condition exists.

When the condition is true, the program is allowed to proceed.

Caution: Careful consideration should be given to the use and placement of the WHEN statement. WHEN causes the program to wait until the condition is TRUE.

Syntax

WHEN *x* relational operator *y*

where *x* and *y* are variables or constants.

Example 1

```
TASK FIRECTRL

WHEN FIRE1 EQ 1
TURNON HALON1
HALT FANS
RUN ALARMF1

IF SPT1 > 240 THEN TURNON SPKLR1

ENDTASK
```

Interpretation: The program FIRECTRL waits at the line WHEN FIRE1=1 until a sensor indicates a fire. The indication of a fire allows the program to proceed.

Example 2

```
TASK COOL

WHEN ((FAN1 EQ 1 AND FAN2 EQ 1) OR FAN4 EQ 1
TURNON FAN3)

ENDTASK
```

Interpretation: The program COOL waits at the line WHEN (FAN1 = 1 AND FAN2 = 1) OR FAN4 = 1 until both FAN1 and FAN2 are ON or FAN4 is ON. When either condition tests true, the program proceeds to the next line, which is to turn on FAN3.

Usage Rules

1. The reserved word WHEN must be immediately followed by a logical expression.

2. If multiple conditions are to be tested, the word WHEN should only be stated once. Each additional condition is included in the test by the reserved words AND, OR, or NOT.

XOR

This logical operator calculates the exclusive OR of two operands. If the operands are logically the same, the result is *false* (0). If the operands are not logically the same, the result is *true* (*1*).

Syntax

x XOR y

where x and y are any constant, variable, or mathematical expression.

Truth Table

Inputs		Outputs
x	*y*	*z*
0	0	0
1	0	1
0	1	1
1	1	0

Example

Y = (X - 3) XOR 1

Interpretation: When BEST^{++} encounters this line, it calculates Y as 0 when X = 4. When X equals any other number, Y = 1.

Mathematical and Relational Operators

Mathematical and Relational Operators

About this Chapter

This chapter discusses each mathematical and relational operator used in BEST^{++}.

The third type of operator, the logical operator, is discussed in the Reserved Words, Logical Operators, and Functions chapter.

Order of Execution

Within a statement, mathematical operators are executed first, then relational operators, and logical operators are done last.

Mathematical Operators

The following operators are used in the BEST^{++} language to perform mathematical processes. They are listed in order of precedence of execution in a statement.

^ (exponentiation)
/ (division)
* (multiplication)
- (subtraction or negation)
+ (addition)

Relational Operators

BEST^{++} uses relational operators to perform comparisons. All relational operators have the same precedence and, therefore, are executed from left to right in any single statement.

The single-character relational operators are:

< (less than)
> (greater than)

In some cases, it is necessary to type two characters together in order to represent a single relational operator. These characters must be typed as shown to work correctly. The two-character operators are:

EQ (equal to)
< = (less than or equal to)
> = (greater than or equal to)
< > (not equal to)

^ (Exponentiation)

The exponentiation sign (^) is a mathematical operator that raises a number to a power.

Syntax

x ^ *n*

where *x* is the base, and may be any variable, constant, or mathematical expression.

n is the exponent and may be any variable or constant with a positive integer value.

Note: If *n* is a real value, the fractional portion is truncated before the computation takes place.

Example

AREA = SIDE ^ 2

Interpretation: AREA is assigned the value of SIDE raised to the second power (SIDE ^ 2 = SIDE * SIDE).

Usage Rule

The exponentiation sign must be preceded by the variable, constant, or expression that is to be raised to a power and followed by the integer, variable, or constant that specifies the power.

/ (Division)

The division sign (/) is a mathematical operator used for dividing one value by another.

Syntax

x / *y*

where *x* is the dividend and may be any variable, constant, or mathematical expression.

y is the divisor and may be any variable, constant, or mathematical expression with a nonzero value.

Example 1

RUNHOUR = RUNMINUT/ 60

Interpretation: RUNHOUR is assigned a value 1/60th the value of RUNMINUT.

Example 2

DIVAL = (X - 1) / (Y + 2)

Interpretation: The value of X - 1 is divided by the value Y + 2 and the resulting value is assigned to DIVAL.

Usage Rules

1. The division sign must be placed between the divisor and the dividend. The value that precedes the division sign is divided by the value that follows it.

2. Spaces are permitted before and after the division sign.

3. Any number divided by zero will return zero (0).

* (Multiplication)

The multiplication sign (*) is a mathematical operator used for multiplying two variables, constants, or mathematical expressions.

Syntax

*x * y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example 1

PUMP1RUN = PUMP2RUN * 2

Interpretation: PUMP1RUN is assigned twice the value of PUMP2RUN.

Example 2

MULTVAL = (X + 1) * (Y - Z)

Interpretation: The value of X + 1 is multiplied by the value Y - Z and the resulting value is assigned to MULTVAL.

Usage Rules

1. The multiplication sign must be placed between the variables, constants, or expressions that are to be multiplied.

2. Spaces are permitted before and after the multiplication sign.

- (Subtraction)

The subtraction sign (-) is a mathematical operator used for subtracting one value from another.

Syntax

x - *y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example 1

ERROR = SETPOINT - TEMP

Interpretation: ERROR is assigned a value equal to the value of SETPOINT minus the value of TEMP.

Example 2

SUBVAL = (X * 12) - (Y ^ 2)

Interpretation: The value Y ^ 2 is subtracted from the value X * 12, and the resulting value is assigned to SUBVAL.

Usage Rules

1. The subtraction sign must be placed between the variables, constants, or expressions that are to be subtracted.

   The value that follows the subtraction sign is subtracted from the value that precedes it.

2. Spaces are permitted before and after the subtraction sign.

- (Negation)

The negative sign (-) is a mathematical operator used to indicate a negative quantity. A negative sign can be used with a constant, variable, or mathematical expression.

Syntax

-*x*

where *x* is any variable, constant, or mathematical expression.

Example 1

y = -10

Interpretation: BEST^{++} assigns *y* the value of negative 10 (*y* = 0 - 10).

Example 2

z = N * (-L)

Interpretation: z is assigned the value of N multiplied by negative L.

Note: If the value of L is less than zero, then negative L will be greater than zero.

Example 3

z = -(x + y)

Interpretation: z is assigned the value of negative the sum of x and y.

Usage Rule

A space is permitted between the negative sign and the constant, variable, or expression it is negating.

+ (Addition)

The addition sign (+) is a mathematical operator used for adding two values together.

Syntax

x + *y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example 1

TOTALRUN = RUNTIM1 + RUNTIM2

Interpretation: TOTALRUN is assigned the value of RUNTIM1 plus RUNTIM2.

Example 2

ADDVAL = (X * 4) + (Y - 2)

Interpretation: The value X * 4 is added to the value Y - 2, and the resulting value is assigned to ADDVAL.

Usage Rules

1. The addition sign must be between the variables, constants, or expressions that are to be added.

2. Spaces are permitted before and after the addition sign.

< (Less Than)

The less than sign (<) is a relational operator used to compare two values. The comparison yields a value of TRUE if the value to the left of the sign is less than the value to the right of the sign.

Syntax

x < *y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example

IF TEMP1 < TEMP2 THEN RUN HEATER ENDIF

Interpretation: A task named HEATER runs if TEMP1 is less than TEMP2 when this statement is executed.

Usage Rules

1. The less than sign must be between the variables, constants, or expressions that are to be compared.

2. Spaces are permitted before and after the less than sign.

< = (Less Than or Equal To)

The less than or equal to sign (< =) is a relational operator used to compare two values. The comparison yields a value of TRUE if the value to the left of the sign is either less than or equal to the value to the right of the sign.

Syntax

x < = *y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example

IF TEMP1 - 4 < = TEMP2 THEN RUN HEATER ENDIF

Interpretation: If the value of TEMP1 - 4 is less than or equal to TEMP2, then a task named HEATER runs when this statement is executed.

Usage Rules

1. The less than or equal to sign must be between the variables, constants, or expressions that are to be compared.
2. Spaces are permitted before and after <=.
3. Spaces are not permitted between < and =.

< > (Not Equal To)

The not equal to sign (< >) is a relational operator used to compare two values. The comparison yields a value of TRUE if the two values are not equal.

Syntax

x < > *y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example

IF STATUS < > 1 THEN HALT RUNTIME ENDIF

Interpretation: If the value of STATUS is not equal to 1 when this statement is executed, then a task named RUNTIME is halted.

Usage Rules

1. The not equal to sign must be between the variables, constants, or expressions that are to be compared.
2. Spaces are permitted before and after < >.
3. Spaces are not permitted between < >.

> (Greater Than)

The greater than sign (>) is a relational operator used to compare two values. The comparison yields a value of TRUE if the value to the left of the sign is greater than the value to the right of the sign.

Syntax

x > *y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example

IF MONTH > 9 THEN TURNON PUMP1 ENDIF

Interpretation: If the value of MONTH is greater than 9 (that is, 10, 11, or 12) when this statement is executed, then PUMP1 is turned on.

Usage Rules

1. The greater than sign must be between the variables, constants, or expressions that are to be compared.

2. Spaces are permitted before and after the > sign.

> = (Greater Than or Equal To)

The greater than or equal to sign (> =) is a relational operator used to compare two values. The comparison yields a value of TRUE if the value to the left of the sign is greater than or equal to the value on the right of the sign.

Syntax

x >= *y*

where *x* and *y* are any variables, constants, or mathematical expressions.

Example

IF MONTH >= 9 THEN TURNON PUMP1 ENDIF

Interpretation: If MONTH is greater than or equal to 9 (that is, 9, 10, 11, or 12) when this statement is executed, then PUMP1 is turned on.

Usage Rules

1. The greater than or equal to sign must be between the variables, constants, or expressions that are to be compared.

2. Spaces are permitted before and after >=.

3. Spaces are not permitted between > and =.

EQ (Equal To)

The equal to symbol (EQ) is a relational operator used to compare two values. The comparison yields a value of TRUE if the two values are mathematically equal.

Note: Do not confuse this relational operator with the mathematical operator =, which is used to assign a value to a variable.

Syntax

x EQ *y*

where *x* and *y* are variables, constants, or mathematical expressions.

Example

IF TMR1 EQ 60 THEN TURNOFF FAN1 ENDIF

Interpretation: If the value of TMR1 is equal to 60 when this statement is executed, then FAN1 is turned off.

Usage Rules

1. The equal to symbol must be between the variables, constants, or expressions that are to be compared.

2. Spaces are required before and after the equal to symbol.

Symbols

Symbols

About this Chapter

This chapter familiarizes you with BEST^{++} symbols. Symbols are listed according to their priority of execution. A description of and syntax for each symbol is presented along with examples and usage rules.

= (Mathematical Assignment)

This symbol is used to assign a value to a variable.

Syntax

x = *y*

where: *x* is a variable name, and

y is any variable, constant, or mathematical expression.

Example

AVTEMP = (TEMP1 + TEMP2 + TEMP3) / 3

Interpretation: When this statement is executed the mathematical expression AVTEMP = (TEMP1 + TEMP2 + TEMP3) / 3 is evaluated and the resulting value is assigned to the variable AVTEMP.

Usage Rules

1. A single variable name must be placed before the mathematical assignment symbol.
2. A valid variable, constant, or mathematical expression must be placed after the mathematical assignment symbol.
3. Spaces are permitted before and after the mathematical assignment symbol.

Note: Do not confuse this mathematical operator with the relational operator EQ. EQ cannot be used to assign a value to a variable. EQ is used to compare two values.

, (Comma)

This symbol is used to separate values or serve as a placeholder for a value in statements.

For the syntax, usage rules, and examples of how the comma is used, refer to Use of Commas in the Statements chapter.

( ) (Parentheses)

Parentheses are used as a grouping symbol — what is between them is to be treated as a quantity.

Parentheses are also used to create or alter the order in which BEST^{++} mathematical and logical operators are evaluated. Operators inside parentheses are evaluated before operators outside parentheses. When parentheses are nested, the inner parentheses are evaluated first, and outer parentheses are evaluated last.

Syntax

(*x*)

where *x* is a mathematical or logical expression.

Example 1

z = (x + 2) * z

Interpretation: The operator +, which appears inside the parentheses, is evaluated before the operator * , which is outside the parentheses.

Note: This changes BEST^{++}'s normal sequence of evaluation, which puts multiplication before addition.

Example 2

IF X AND (Y OR Z) THEN W = 0 ENDIF

Interpretation: The operator OR, which appears inside the parentheses, is evaluated before the operator AND, which is outside the parentheses.

Note: This changes BEST^{++}'s normal sequence of evaluation, which evaluates AND before OR.

Example 3

N = ((X + 2) * (Y-1)) / M

Interpretation: The quantities in the inner parentheses (X + 2) and (Y - 1) are evaluated first, then the operation in the outer parentheses (multiplication) is performed.

Note: This changes BEST^{++}'s normal sequence of evaluation, which is multiplication — division — subtraction — addition.

Usage Rules

1. Parentheses must be used in pairs. Each left parenthesis must have one matching right parenthesis and vice versa.
2. Parentheses may be nested, i.e., pairs of parentheses may appear within other pairs of parentheses.

[ ] Array Parentheses

Array parentheses are used to define the arguments of an array. You define arrays of numbers using the DIM keyword. You define arrays of points, functions, schedules or alarms using the ARRAY keyword.

Syntax

[*x*]

where *x* is a number, variable, or mathematical expression.

Example 1

z = A[3+i]

Interpretation: The (3+i)th element of the array A is assigned to the variable *z*.

Example 2

A[9]=25

Interpretation: The 9th element of the array A is set to 25.

Usage Rules

1. Array parentheses must be used in pairs.
2. Array parentheses may be nested, i.e., pairs of parentheses may appear within other pairs.

: (Colon)

This symbol is used to designate a label name. Labels can be used to define sections of a task. They can also be the target of a GOTO statement. Unlike step names, however, labels cannot be used in conjunction with REPEAT.

Syntax

: *x*

where *x* is the label name.

Example

```
TASK CTRL

STEP S1
IF (TEMP > 78) THEN GOTO LBL1
TURNON FAN; TURNON PUMP
GOTO LBL2

:LBL1
TURNOFF FAN; TURNOFF PUMP

:LBL2
DELAY CTRL FOR 60

REPEAT
```

Interpretation: In the task CTRL, FAN and PUMP are turned off if TEMP1 is over 78. Otherwise, they are turned on. In either case, the task is delayed for 60 seconds. Execution then continues at STEP S1.

Note: REPEAT causes the task to go back to the *step*, not to the labels.

Usage Rules

1. A label must be the only statement on its line.
2. Label names must conform to the rules for variable names.

; (Semicolon)

The semicolon may be used to separate statements that have been entered on one line to increase the readability of the program. Spaces may be used in place of a semicolon.

Syntax

x; x

where each *x* is a statement.

Example 1

IF TEMP > 72 THEN TURNON FAN; COUNT = COUNT + 1; GOTO TWO ENDIF

Interpretation: If the condition is true, then the three linked statements will be executed.

Example 2

SETPT1 = 72; SETPT2 = 72

Interpretation: The semicolon is used here for organizational purposes, to group two similar statements together.

Usage Rule

It is not necessary to put spaces or blanks before or after the semicolon.

~ (Tilde)

This symbol can be used in BEST^{++} to indicate a remark statement.

To use the tilde to indicate a remark statement:

1. Type an executable statement (optional)
2. Type ~.
3. Type the remark (descriptive text).

A remark statement preceded by a tilde is not downloaded to the Comfort Controller when the program is sent, nor is it uploaded.

The reserved word REM can also be used to indicate a remark statement. For more information regarding this alternate use, refer to REM in the Reserved Words, Logical Operators, and Functions chapter.

Syntax

~*x*

where *x* is the remark.

Example 1

```
~THIS PROGRAM WAS WRITTEN ON 2/1/96 TO TEST THE
~OPERATION OF FAN1

IF TEMP1 > 72 THEN TURNOFF FAN1 ENDIF

~CHANGE TEST VALUE TO 78 IN SUMMER

IF TEMP1 > 78 THEN TURNOFF FAN1 ENDIF
```

Interpretation: Three lines of descriptive text accompany a display or printout of the program. None of them will be sent to the Comfort Controller when the program is sent.

Example 2

```
IF (TEMP1 > 78)          ~ CHECK THE TEMPERATURE
THEN TURNON FAN1;        ~ START BOTH FANS IF TOO HOT
TURNON FAN2
ENDIF
```

Interpretation: Not only are the tildes in this example used as line continuation characters, any text appearing after them is recognized as remarks. The remarks will not be sent to the Comfort Controller when the program is sent.

Example 3

```
~REM     THIS PROGRAM WAS REVISED ON 3/31/96 TO
~REM     INCLUDE CHLR8 AND CHLR9 IN THE
~REM     SEQUENCE.
```

Interpretation: Sometimes the two forms of remark statements are combined as follows: ~REM. Although redundant, the combination is sometimes used for easier identification of the remarks in the body of the program while preventing the remarks from being sent to the Comfort Controller.

? (Question Mark)

This symbol is used to assign the address of a CONNECTed hardware or software point to an ARRAY element. This type of assignment is called an indirect assignment.

Syntax

arrayname[index].ADDRESS = ? *connectname*

where *arrayname* is a BEST^{++} array that has been previously defined using the ARRAY statement.

index is a constant, variable, or mathematical expression that is equal to an integer value.

connectname is the name of a CONNECTed hardware or software point or the name of a BEST^{++} program.

Example

FANARRAY[1].ADDRESS = ?FANSTATUS1

Interpretation: The address of the CONNECTed point FANSTATUS is placed into the first element of the ARRAY FANARRAY.

Usage Rules

1. Addresses may only be assigned to ARRAYs using the .ADDRESS array function.
2. Addresses may only be CONNECTed variables.
3. The ? symbol may only appear to the right of the = sign in an indirect assignment.

Task Execution

Task Execution

Single Task Execution

When you download a BEST^{++} task, it executes after the user-configurable time delay. For information on the time delay, refer to TASK in the Reserved Words, Logical Operators, and Math Functions chapter in this manual.

Task execution begins at the TASK statement. BEST^{++} continues to execute each line sequentially, from top to bottom until it reaches an ENDTASK statement, unless the task itself instructs otherwise (for example, GOTO another line, CALL a subroutine, EXIT the task, REPEAT).

Multiple Task Execution

All BEST^{++} tasks run concurrently in the Comfort Controller. If multiple programs are running, BEST^{++} executes the tasks according to priority as specified in the TASK statement. Each priority gets a time allocation to execute in and can execute as many lines as possible in that amount of time. For more information on priority, refer to TASK in the Reserved Words, Logical Operators, and Math Functions chapter in this manual.

Halting a Task

If a task is HALTed and then continued, the task will resume at the next statement after the HALT.

Subroutine Execution

When a task CALLs a SUBROUTINE, BEST^{++} executes that subroutine from beginning to end without executing any other tasks or subroutines.

Interaction with Algorithms

BEST^{++} tasks and Comfort Controller algorithms run concurrently. If a BEST^{++} task has control of a point, any algorithms that control the point are locked out until the BEST^{++} task encounters an AUTO command. AUTOing the point removes the force. For more information on AUTO, refer to AUTO in the Reserved Words, Logical Operators, and Math Functions chapter in this manual.

This exchange of control is useful when standard control algorithms are appropriate, but a special control routine is occasionally required.

When the Comfort Controller's Power Is Cycled

If a task is running when the Comfort Controller's power is turned off and then on again, the task will restart from the beginning after the POR delay, as specified in the TASK statement, expires. If more than one task was executing, all the tasks will begin according to their priority, as specified in the TASK statement. They will all begin executing Line 1 after their respective POR delays expire.

Any BEST^{++} variables that were changed before the power was turned off will retain their values.

Debugging System

Debugging System

The debugging system consists of two functions: BEST^{++} Debug and System Debug. BEST^{++} Debug is a subset of System Debug intended for those who require only access to their programs. This chapter describes both debuggers.

BEST^{++} Debug allows you to access the currently downloaded programs and the global dictionary in the target Comfort Controller.

System Debug allows you to view the current maintenance and configuration status of all hardware and software points, schedules, setpoints, standard algorithms, global algorithms, and BEST^{++} programs in the target Comfort Controller.

You can use these debuggers to troubleshoot a program that compiled and downloaded to the controller successfully but is not operating as intended. They provide you with the following capabilities to control individual tasks in a program:

- RUNning, HALTing, or DELAYing a task
- Viewing or modifying the configuration of an algorithm or point in the target Comfort Controller
- Viewing, forcing, or autoing points to simulate control operation

For more information, refer to Monitoring and Controlling Tasks later in this chapter.

Using BEST^{++} Debug

This section describes how to use BEST^{++} Debug, associated menu items, and the Debug dialog box when you have a copy of the source program. If you do not have a copy of the source program, use the BEST Sources function in System Debug.

To access BEST^{++} Debug, select *BEST^{++} Debug* from the BEST menu in the Programmer's Environment main menu or press key F8.

Follow the steps below to debug your program:

1. Select the target controller using *Set Controller Address* in the BEST menu.

2. Open the folder containing the source programs. Display the folder program list by selecting the small down arrow (¬) on the right of the status bar. Select the program to be debugged from the list. The program will be displayed in the PE window.

   Note: If you changed the program since the last compile, you must compile it before activating the debugger.

3. Select BEST^{++} Debug from the BEST menu in the PE main menu or press F8.

You are now ready to debug your program. Refer to How To Debug A Program later in this chapter.

Debug Dialog Box

The Debug dialog box contains the menu items and information described below. The names displayed are the user names defined in your program, including variables, tasks, connects, etc.

The left box displays a list of all user names in the program. Use the scroll bars to view names that do not fit in the box.

The right box displays data for the selected user name based on the selected menu item. For example, the default menu item selection is Functions. If you select the user name for the task, the functions supported by the keyword TASK are displayed.

Debug Dialog Box Menu Items

Menu Item	Use this menu item to
Functions	display the current value of any output functions and a list of input functions supported by the selected username.[1]
Configuration	display current configuration decisions and their values for the selected username.[2]
Maintenance	display maintenance decisions and their values for the selected username.[2]

Debug Dialog Box Menu Items *(continued)*

Menu Item	Use this menu item to
Show/Hide Global	when Show Global is selected, display the list of user names, including those from the program currently being debugged (Local Variables) and those defined in the global dictionary (Global Variables). when Hide Global is selected, display only the list of user names from the program currently being debugged (Local Variables).
Name/Desc	when Name is selected, display the name used to connect to the maintenance or configuration decision. This feature is available only when you are viewing maintenance or configuration decisions. when Desc is selected, display the standard 24 character description that identifies each of the maintenance or configuration decisions.
Exit	return to the Programmer's Environment.

[1]When the task name is selected, the Function menu item supports a pull down menu. Refer to Monitoring and Controlling Tasks later in this chapter.

[2]If the task name is selected, the Function pull down menu will be active when this items is selected.

How To Debug a Program

This section summarizes how to troubleshoot a program using BEST^{++} Debug.

1. Open the folder and select the program to analyze, If you do not have the source program, refer to System Debug, Debugging a Program When You Do Not Have the Source and go to step 4.

2. Compile the program. Make a hard copy list file, which adds line numbers to the program, using the Make List File menu item. The line numbers are used with Debug to monitor and control program flow.

3. Choose *BEST^{++} Debug* from the BEST menu. The Debug dialog box is displayed.

4. Select a task from the list in the Debug dialog box. Then select the *Maintenance* menu item, which allows you to see the current state of the task. Each time you select the task user name, the maintenance parameters will be updated. Otherwise, the maintenance parameters will be updated at the rate of once every 5 seconds.

5. To step through a task:

   a. Use the *Breakpoint* command from the Functions menu to set a breakpoint at the line number where you want to start evaluating the task program flow. The status bar on the main window will indicate that the breakpoint is set on the line number entered.

   Note: Since Step and Label statements are not actually executed, you cannot use them as breakpoints. Select only lines that have logic keywords.

   b. When the breakpoint is reached, use Debug to view and modify the values of any constants and variables. Use the *Show Global* menu item to view global variables and points.

c. To single step through the program, select the task name, then select the *Single Step* command from the Functions menu. BEST^{++} executes the current line and advances to the next one based upon the results. The status bar will identify the next line to be executed and the line will be displayed in the edit box.

Note: Before you can Single Step, you must set a breakpoint.

d. Continue to single step through the program or execute another task section by setting a new breakpoint and continuing program execution.

6. When debugging is completed, clear the breakpoint and select Continue to run the program.

Monitoring and Controlling Tasks

This section describes how to determine the state, reschedule rate, and execution time of a BEST^{++} task. It also describes how to control the task in order to troubleshoot your programs.

1. Select the program in the target device that you would like to work with as described in How To Debug A Program.

2. Select the user name that identifies the task in the selected program. A program can contain more than one task.

To monitor the task:

3. Select *Maintenance* from the Debug dialog box. The current task state, task timer, and execution time will be listed in the status window. The data will be updated dynamically every 5 seconds.

To control the task:

4. Perform step 3 above, then select *Functions* from the Debug dialog box. The following list of task control commands will be displayed.

Task Control Commands

Command	Use this command to
Show Functions	display the functions associated with the task in the status list of the Debug dialog box. To return, select the Maintenance function.
RUN	execute the currently selected task from the beginning.
HALT	stop a task in the Comfort Controller. If the task is already halted, this command will have no effect. However, the task will not execute again until it is RUN. Any devices controlled by a task that is halted remain in the last state commanded by the program before it was halted. Any actions not yet taken do not occur. HALT does not restore any devices controlled by the task to a predetermined state. If any of the BEST^{++} forces that were completed by the program are to be changed, either you or another program must change them. If a task is already stopped when this command is issued, the normal operation of the program is not affected. The task will interpret the redundant HALT as confirmation that the program should not be running.

Task Control Commands *(continued)*

Command	Use this command to
DELAY	suspend the execution of a selected task. A dialog box is displayed where you can enter the delay time in seconds. After you confirm, the task delays for the specified time then continues executing.
BREAKPOINT	halt a task at a specific program line. A dialog box will be displayed for you to enter the line number. (You can obtain a copy of the source program with line numbers by compiling the program and selecting *Make List File* from the BEST menu.) Breakpoints can only be inserted on lines that contain logic keywords (IF, WHEN, =, DELAY, etc.). To clear a breakpoint, select this command and enter a line number of zero. **Note:** Version 1.3 and earlier of the Comfort Controller will not accept a breakpoint on the first logical statement after TASK. Put the breakpoint on the second or higher logic statement.
CONTINUE	cause a task delayed as a result of BREAKPOINT or SINGLE STEP to resume executing. The task will begin executing from the breakpoint or last single step until the program is halted or the breakpoint is reached again. The shortcut key for this command is F5.

Task Control Commands *(continued)*

Command	Use this command to
SINGLE STEP	cause a task at a breakpoint to execute the current line number and then stop again. The short-cut key is F8. When the program stops after SINGLE STEP, the line is displayed in the edit box at the bottom of the debug window.

System Debug

This section describes System Debug and associated menu items and the Upload/Debug dialog box. The Upload/Debug dialog box is a window into the target Comfort Controller, allowing you to:

- view and override values and modify configuration.
- access functions and BEST^{++} programs.

To access the System debugger, select *System Debug* from the BEST menu in the Programmer's Environment. BEST^{++} displays the Upload/Debug dialog box, which is similar to the Debug dialog box discussed previously with the Select menu item added.

Debugging When You Do Not Have the Source Program

This section describes how to debug when you do not have the source program. Follow the steps below to debug your program in a Comfort Controller.

1. Select the target controller using *Set Controller Address* in the BEST menu. Select System Debug.
2. Select *BEST^{++} Source* from the Select menu.
3. After the programs are listed, select the one to debug.
4. Select *Upload*. The selected program is uploaded, and the decompiled version is displayed in the PE window.

   Note: If more than one program exists, save each of the uploaded programs to a temporary folder so that the

programs do not have to be uploaded each time you select a new program.

You are now ready to debug your program. Refer to How To Debug A Program.

Upload/Debug Dialog Box Menu Items

The Upload/Debug dialog box contains the following menu items. Use the Tab key to access the menu.

Menu Item	Use this menu item to
Select	display a list of the different object type categories in the target device. Refer to Select Menu below for more details. When you select a category, all of the objects of that type in the device are uploaded, and you can view or modify them.
Create	This function is not currently supported.
Delete	active only when the BEST^{++} Sources category is chosen under Select. Refer to BEST^{++} Sources Command.
Upload	active only when the BEST^{++} Sources category is chosen under Select. Refer to BEST^{++} Sources Command.
Debug	access the Debug dialog box for the selected item in the category chosen. Refer to BEST^{++} Sources Command.
Exit	exit the debugger and return to the Programmer's Environment main menu.

Select Menu Commands

This section describes the Select menu commands displayed in the Upload/Debug dialog box.

Command	Use this command to
I/O Channels	access all of the hardware and software points in the target controller. The system will read each of the possible hardware channels (64 in a 6400 and 16 in a 1600) and software channels (32 in a 6400 and 16 in a 1600). The status bar will indicate when the read is completed and the Upload I/O Channels dialog box will be displayed. The Upload I/O Channels dialog box will contain each hardware channel's name and description, starting with Channel #1, followed by the software channels. If you are using V1.1 or V1.2 Comfort Controller software, then any point not used is identified by the name *Deconfig* and the descriptor *Deconfigured I/O Channel*. V1.3 software and above does not display deconfigured I/O channels. Once you select a point, you can select *Debug* to activate the Debug dialog box. Refer to Debug Dialog Box for a description of its operation.
Schedules	access all time and holiday schedules in the target device,

Select Menu Commands *(continued)*

Command	Use this command to
	including Occupancy, Linkage, AOSS, and network time schedules. The schedules are displayed with the 8 character name and the 24 character description. Once you select a schedule, you can select *Debug* to activate the Debug dialog box. Refer to Debug Dialog Box for a description of its operation.
Setpoints	access all setpoint schedules, including AOSS setpoint schedules, in the target device. The system reads each setpoint schedule and displays its name and description in the Upload Setpoints dialog box. If you are using V1.1 or V1.2 Comfort Controller software, any setpoint schedule not used will be identified with the name *Deconfig* and the description *Deconfigured Setpoint*. V1.3 software and above does not display deconfigured setpoints. Once a setpoint is selected, you can select Debug to activate the Debug dialog box. Refer to Debug Dialog Box for a description of its operation.
Data Collection	access runtime and consumables in the target device. The system reads each data collection table and displays its name and description in the Upload Data Collection dialog box.

Select Menu Commands *(continued)*

Command	Use this command to
Standard Algorithms	access all of the standard algorithms in the target device. The system reads each algorithm and displays its name and description in the Upload Algorithms dialog box. The status bar indicates when the read is completed. If you are using V1.1 or V1.2 Comfort Controller software, any algorithm not used is identified by the name *Deconfig* and the description *Deconfigured Function*. V1.3 software and above does not display deconfigured algorithms. Once you select an algorithm, you can select *Debug* to activate the debug dialog box. Refer to Debug Dialog Box for a description of its operation.
BEST^{++} Programs	access any configuration or maintenance decisions in a BEST^{++} program in the target device. This command gives you the same functionality as BEST^{++} Debug.
BEST^{++} Sources	This command provides you with capabilities not supported in BEST^{++} Debug. It allows you to delete a program or initialize to delete all programs and the dictionary. Additionally, you can upload and decompile a program for which you do not have the source.

Select Menu Commands *(continued)*

Command	Use this command to
Select	has the same functionality as in the Upload/Debug dialog box.
Create	not supported.
Delete	remove the selected BEST^{++} program from the connected controller.
Upload	upload the selected program from the connected controller and decompile it.
Debug	access the BEST^{++} Debug window.
Initialize	delete all the controller's BEST^{++} programs. You will be prompted to confirm the initialization. Select *Yes* or *No*. If a program is running when you initialize the Comfort Controller, points forced by the BEST^{++} program remain forced. When BEST^{++} Debug is selected, BEST will access the target device to determine what programs are currently loaded. The status bar indicates if the target device read was successful. If *No Response* is displayed, BEST was unable to communicate with the target Comfort Controller. Verify that the address is correct and retry the command.

Select Menu Commands *(continued)*

Command	Use this command to
	After the target Comfort Controller is successfully read, the list of programs loaded will be displayed in the Upload BEST Programs dialog box. The status bar will indicate *Read Complete*.
Exit	exit the debugger and return to the BEST^{++} Programmer's Environment.
System	access all Service-Config Tables and other functions. The most commonly used functions are described below.

System Functions

Name (Description)	Functions
CC6400 (Comfort Controller)*	Configuration - Controller identification table Maintenance - Available real time
CCNCNTRL (CCN control)	Configuration - Communication configuration Maintenance - Communication statistics
RTC (Real-time clock)	Configuration - Broadcast configuration Maintenance - Real-time clock
SETCLOCK (Set clock)	Configuration - Real-time clock configuration Maintenance - None

*Name and description found in CNTRL-ID.

System Functions *(continued)*

Name (Description)	Functions
UPDATEDB (Update database)	Configuration - Updates database from Service Configuration tables Maintenance - Identifies memory allocation and error status

How To Delete BESTPROGRAMS

This procedure describes how to use System Debug to delete a single program or all programs in a target Comfort Controller.

Note: The program is not deleted from the disk.

1. Select the target Comfort Controller using *Set Controller Address* in the BEST menu.
2. Select *System Debug* to display the Upload/Debug dialog box.
3. Select *BEST*$^{++}$ Programs to display a list of all the programs currently loaded in the target device.
4. To delete a single program, select the program and select *Delete* from the menu. A confirmation dialog box is displayed. Selecting *Yes* replaces the listed program name with *Delete*. Selecting *No* causes no change to the program list.

   Note: It may take some time to delete a large program.

5. To delete all programs, select *Initialize* from the menu. A confirmation dialog box is displayed. Selecting *Yes* deletes all program names from the list. Selecting *No* causes no change to the program list.

Syntax Error Messages and Report Warnings

Syntax Error Messages and Report Warnings

What Is Syntax?

Syntax is the proper arrangement of words and symbols in a program. An incorrect arrangement of words and symbols results in a syntax error. To determine the syntax for a reserved word, math function, operator, or symbol, refer to the appropriate chapter.

Detection of Syntax Errors and Report Warnings

When you select the *Compile Program*, *Compile All*, or *Make List File* commands, the BEST^{++} compiler checks your program(s) for syntax errors and report warnings.

Note: Selecting Make List File does not compile your program. You must compile the program before selecting Make List File.

Syntax Errors

If a syntax error is detected, the compiler generates a syntax error message and displays it below the line in error.

The statement below contains a syntax error followed by the applicable syntax error message.

```
CONNECT SPT AS A TEMP_INPUT {SPT}
TURNON SPT
        ***LINE 40, Error 19 (TURNON), invalid parameter name
```

Note: You must correct all syntax errors in the source file and compile the program correctly before the program can be downloaded (sent) to a controller. You do not need to correct warnings before downloading the program.

Do not edit the list file because changes to it will not be saved to the source file.

Report Warnings

If a missing argument, unreferenced object, or misuse of = (equal sign) is detected, the compiler corrects the error, and generates and displays a report warning.

The statement below contains an error followed by the applicable report warning.

```
IF STEPTIMER = 60 THEN TURNOFF FAN1 ENDIF
```

Line 7, Error 33 (VARIABLE STEPTIMER), warning, = used as a conditional. Changed to EQ.

List of Syntax Error Messages and Report Warnings

The following BEST++ syntax error messages and report warnings are listed according to their error code numbers. A description for each syntax error message and report warning is provided.

Note: Report warnings will not display unless you select *Report Warnings* from the Option menu.

1. "IN element missing"

   The external CONNECT statement is missing the element number of the controller to which you wish to CONNECT.

2. "THEN missing"

   The THEN portion of the IF ... THEN ... [ELSE] ... ENDIF statement is missing.

3. "warning, undeclared name. Assumed type VARIABLE"

   There was no definition for the name encountered in the Task or the Global Dictionary and, therefore, it is assumed to be a variable.

4. "expected a user name"

   The compiler expected a unique, user-defined name to follow the reserved word.

5. "warning, missing arguments"

   There are fewer than the expected number of arguments for this keyword. The compiler will add null arguments.

6. "duplicate user name"

   This variable has been defined already. Use a different name.

7. "missing arguments in ( )"

   There are fewer than the expected number of arguments for this keyword. The compiler will add null arguments.

8. “GOTO label not defined”

   The label name following GOTO has not been defined with a “STEP labelname” or “:labelname”.

9. “cannot find a previous operand”

   An operator (such as +, -, *, or /) is missing before a variable and/or constant.

10. “post operand missing”

    An operator (such as +, -, *, or /) is missing after a variable and/or constant.

11. “object does not have a default function”

    The user-defined name was used without defining or defined incorrectly.
    vartype.decname or *functiontype,subfunction.decnam*e.

12. “missing) “

    The number of open parentheses exceeds the number of closed parentheses.

13. “too many constants defined”

    Too many initialization parameters have been defined for this reserved word. Initialization parameters are defined between { }.

14. “end of file before } “

    An end of initialization symbol “ } “ is missing.

15. “expected a KEYWORD”

    In the CONNECT statement, *var* was used without specifying a valid *vartype* (such as TEMP_INPUT or DISC_OUTPUT). For a list of default *vartype* names, use the Paste Function command to display the syntax for CONNECT.

16. “name is not a label”

    The name used in the GOTO statement is defined, but it is not a label name.

17. “warning, unreferenced object”

    A name was defined but not used in the program.

18. “nothing requiring initialization”

    The reserved word entered before { } cannot be initialized.

19. “invalid parameter name”

    The decision name associated with the function is not valid for the *vartype*.

20. “variable is not an array”

    The variable is not an array and does not support array indexing.

21. “PROGRAM definition missing”

    The PROGRAM definition statement is missing.

22. “array without a subscript”

    The variable is an array and requires an array index.

23. “constant outside limits”

    The constant that is used in the initialization string is outside its valid range.

24. “name too long”

    The name used in the initialization string has too many characters. The limit is eight characters.

25. “description text too long”

The description in the initialization string consists of more than 24 characters.

26. “time field invalid”

The time field in the initialization string is invalid. The valid format is HH:MM, where HH is 0 to 24 and MM is 0 to 59.

27. “date field invalid”

The date field in the initialization string is invalid. The valid format is MM/DD/YY, where MM is 0 to 12, DD is 0 to 31, and YY is 0 to 99.

28. “invalid flag field”

The value entered in the { } is invalid. You can only enter 1's and 0's in the flag field.

29. “ENDIF missing”

The ENDIF required to terminate an IF statement is missing.

30. “ENDLOOP missing”

The ENDLOOP required to terminate a LOOP statement is missing.

31. “field must be a number”

The data entered in the initialization string is not a number.

32. “TASK missing”

The program is missing the TASK statement which defines the task name.

33. “warning, = used as a conditional. Changed to EQ”

An = sign was used in a conditional statement (following an IF or WHEN). The BEST^{++} compiler changed the equal sign to EQ, which must be used to show conditional equality.

34. “operand missing”

An operand such as +, -, *, or / is missing in a mathematical expression.

35. “REPEAT without STEP”

A REPEAT statement was used without a previous STEP statement.

36. “ELSE without ENDIF”

The ENDIF required to terminate an ELSE in an IF statement is missing.

37. “ENDTASK missing”

The ENDTASK statement required to terminate a program is missing.

38. Not used.

39. “PROGRAM NAME cannot be same as Dictionary NAME”

The global dictionary must have the same name as the folder in which it resides. The global dictionary cannot have the same name as another program. The global dictionary must be the first program in a folder.

40. “Can only use this keyword with the advanced version”

This reserved word is not supported by the BEST^{++} compiler.

41. “compilation terminated - too many logic words”

The number of logic reserved words (TASK, IF, THEN, = DELAY, etc.) has exceeded the maximum number of 600.

42. “compilation terminated - dictionary full”

The number of reserved words has exceeded the maximum number of 300.

43. “compilation terminated - too many errors”

The number of errors has exceeded the maximum number of 120.

44. “compilation terminated - object functions list full”

The number of functions (hardware and software points, configuration and maintenance decisions, HVAC functions, system functions, schedules, and alarms) has exceeded the maximum number of 240. The program must be split to compile and execute successfully.

45. “compilation terminated - too many lines of code”

The number of source lines has exceeded the maximum number of 600. The program must be split to compile and execute successfully.

46. “compilation terminated - link list full”

The number of links used by reserved words, operators, mathematical functions, etc. has exceeded the maximum number of 1500. The program must be split to compile and execute successfully.

47. “compilation terminated - object list full”

The number of objects has exceeded the maximum number of 600. The program must be split to compile and execute successfully.

48. “compilation terminated - initialization list full”

The number of initialization parameters has exceeded the maximum number of 900.

49. “compilation terminated - ran out of heap space”

The internal program compilation array has exceeded the maximum size of 1500.

50. “compilation terminated -group priority stack overflow”

The nesting stack has exceeded the maximum level of 20. You have nested too many IF, THEN, or LOOP statements.

51. “incomplete syntax”

The syntax is incomplete and the BEST^{++} compiler cannot properly evaluate it. For example, the = sign is missing from this statement: $x \quad y + 3$

52. “duplicate label name”

The label name has been used in a previous STEP labelname or :labelname.

Appendixes

Appendix A

Comfort Controller Force Priorities

The following table lists, in decreasing order, the force priorities within the Comfort Controller. You specify the force priority (*forcepri*) in a TASK statement. For the syntax, refer to TASK in the Reserved Words, Logical Operators, and Math Functions chapter.

When BEST^{++} writes a value to an internal hardware or software point, it will write the value as dictated by the *forcepri*. When forcing variables over the CCN network, all writes will cause BEST forces, regardless of the *forcepri* setting.

forcepri	Description	Network Service Tool Display
0	Standard control algorithm	(blank; no display)
1	Fire	Fire
2	Safety	Safety
3	Network Service Tool/LID	Service
4	Building Supervisor	SUPVSR
5	Offsite monitoring Building Supervisor	Monitor
6	Minimum on or off time	Min Off
7	Controlling POC	Control
8	BEST^{++} command (*8* is the default value. If you enter a comma as a placeholder for *forcepri* in the TASK statement, BEST^{++} will compile *forcepri* as an *8*.)	BEST
9	Temperature override	Temp
10	Loadshed	Load

Appendix B

Metric Units

BEST++ programs can read and write values in either customary US or metric engineering units. The default is customary US. All system elements that share data in BEST++ must communicate using the same engineering units (customary US or metric).

Changing to Metric Units

To change from customary US to metric engineering units, select *Metric Units* from the Option menu in the Programmer's Environment. Then, when necessary, define constants with metric engineering unit names or numbers. Refer to the next topic to determine when you must define a constant's engineering unit.

Defining the Engineering Units of a Constant

In BEST++ you must define the engineering unit of a constant used in a calculation or comparison if:

- the constant's value is not the same in both customary US and metric engineering units. For example, 20·F is different from 20·C.

- the program is to be converted from customary US to metric engineering units or from metric to customary US.

To define the engineering unit of a constant, type the standard BEST++ engineering unit name or number within { } as an initialization string. To view a list of engineering unit names and numbers, select *Unit Names* from the Help menu. BEST++ displays the list of engineering units and numbers. The units displayed (customary US or metric) will depend on whether you selected *Metric Units* from the Options menu.

Example 1

Example 1 below indicates how to define the units of a constant by entering the unit name.

```
IF TEMP1 > 20 {deg_C} THEN TURNON FAN1
```

Example 2

Example 2 below indicates how to define the units of a constant by entering the unit number.

```
IF TEMP1 > 20 {1} THEN TURNON FAN1
```

Converting Engineering Units

It is possible to write a program using customary US units, compile it, then convert to Metric Units by selecting the Metric Units option. When such a program is decompiled (or uploaded and decompiled), the constant's value will be converted to metric and its customary US units name will be changed to the corresponding metric name. Similarly, programs written in metric may be converted to customary US units.

Example 3

Example 3 below illustrates how the BEST^{++} compiler will convert program lines written with metric engineering units to customary US units.

Metric option selected

```
x = spacetemp + 3 {rel_C}          ~ add 3 relative degrees C to
                                   ~ spacetemp
IF hcvflow > 30 {Ltr_min} THEN     ~ compare hcvflow to 30 ltr_min
hcv = hcv - 1     ENDIF
```

Customary US option selected and program decompiled

```
x = spacetemp + 5.4 {rel_F}
IF hcvflow > 7.926023 {GPM} THEN
hcv = hcv - 1     ENDIF
```

Appendix C

Comparison of FID BEST and BEST++ PE Commands

The table below compares the FID BEST Programmer's Environment (PE) commands to the BEST++ PE commands.

Note: Not all BEST PE commands have comparable BEST++ PE commands.

FID BEST PE Command	Comparable BEST++ PE Command	BEST++ PE Command Location
ACTIVATE	None. Once you download a BEST++ program to a controller and the user-configurable time delay expires, the program executes automatically.	Not applicable
BYE	Exit	File menu
CHECK	Compile Program or Compile All	BEST menu
COPY	Save Program As. . .	File menu
DELETE	Delete	System Debugger's BEST++ Sources dialog box
EDIT	Open Folder and/or Open Program	File menu
GET	Upload	System Debugger's BEST++ Sources dialog box when task name is highlighted
HALT	Halt	Functions menu in the BEST++ Debugger's Debug dialog box when task name is highlighted
INITIALIZE	Initialize	System Debugger's BEST++ Sources dialog box
KEEP	Save Program As. . . (save as a different name)	File menu
LIST	Open Folder	File menu
MOVE	Set Address	BEST menu
PRINT	Print Program or Print All	File menu
RECOVER	None	Not applicable
SEND	Download Program or Download All	BEST menu

Appendix D

ComfortWORKS Programmer's Environment

The information that follows provides instructions on how to use ComfortWORKS to access the BEST++ Programmer's Environment, and a general explanation of each ComfortWORKS Programmer's Environment menu item and command.

Accessing the Programmer's Environment

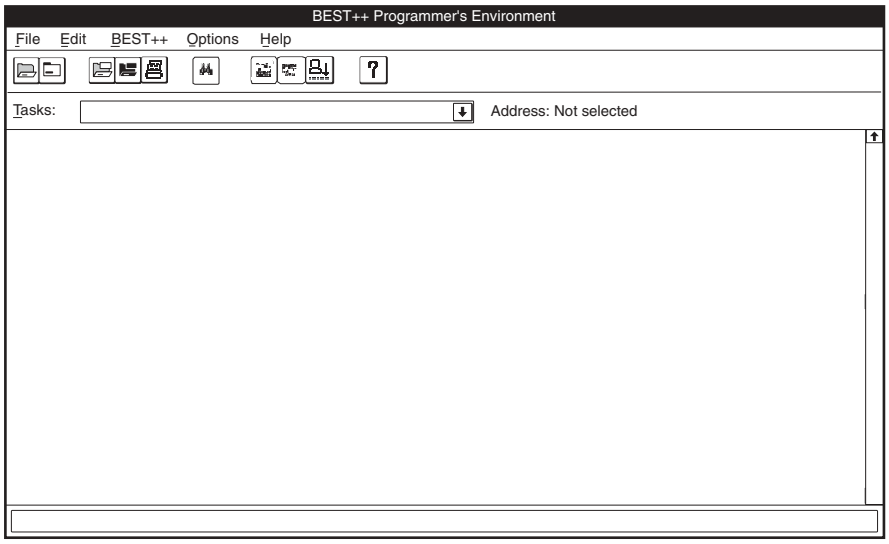
Follow the instructions below to launch the ComfortWORKS BEST++ application and display the Programmer's Environment window.

1. Double click on the BEST++ icon, which is displayed in the ComfortWORKS program group window.

   ComfortWORKS displays a Programmer's Environment window similar to the one shown in the figure below. The top line of the window contains the menu bar. Refer to the BEST++ Menu Command Summary, which appears at the end of this appendix, for a description of each menu item and command. The second line of the window contains the toolbar. Refer below to BEST++ Toolbar for information on using each toolbar button.

   The third line of the window contains a drop-down list, which displays the names of all programs in the folder that is currently open. The section on the far right displays the target Comfort Controller address. If you have enabled the Options menu's *Show units names* command, a drop down list that lists the names of the Comfort Controller's units will display on the far right of this line.

Figure D-1
Programmer's Environment Window



BEST++ Toolbar

The following table describes the buttons that appear on the Programmer's Environment toolbar.

Table D-1
Programmer's Environment Toolbar Buttons

Click on	To
	open an existing folder.
	save the current folder.
	open an existing BEST++ program.
	save the existing BEST++ program.
	print the current BEST++ program.
	search for text. Click on this button with the Ctrl key depressed to Replace text.
	compile the current BEST++ program. Click on this button with the Ctrl key depressed to compile all programs.
	de-compile the current BEST++ program.
	download the current BEST++ program to the Comfort Controller. Click on this button with the Ctrl key depressed to download all programs.
	display help.

Programmer's Environment Menu Command Summary

This section describes each of the commands that appear in the Programmer's Environment window menus.

File Menu

The following table describes each of the commands that appear in the File Menu.

Command	Use this command to
New Folder	create a new folder. When selected, this command displays the New Folder dialog box, which contains a list of the existing folders in the current directory. Enter the new folder file name and click on *OK*. BEST++ will create a new folder, and create and display a global dictionary program for the folder. BEST++ will automatically insert the PROGRAM statement, and assign it the same file name that you assigned to this new folder. The new program name will also display in the Programmer's Environment window's title bar. New Folder Dialog Box Options: File Name: Enter a unique name (up to 8 alphanumeric characters, with first 7 characters unique) of the new folder. You must maintain the file name extension provided by BEST++ (.FLD). Drives: Select the drive on which you wish to store the new folder. The Directories folder: Select the name of the directory to which you wish to save the new folder.

(*continued*)

Command	Use this command to
	New Folder Dialog Box Options (*continued*): Network: Selects a shared network drive to which to save your folder. OK: Exits the dialog box and saves the new folder. Cancel: Exits the dialog box without saving the folder. Help: Displays help information.
Open Folder	open an existing folder. When selected, this command displays the Open Folder dialog box, which contains a list of existing folders in the current directory. Select an existing folder's file location and file name and click on *OK*. BEST++ will open the existing folder. When you open a folder, BEST++ will display the first program in the folder, which is always the global dictionary. The folder name will display in the title bar of the Programmer's Environment window. To display other programs in the folder, click on the Tasks drop down list.
Save Folder	save the open folder. If the folder was not already saved, this command, when selected, displays the Save As dialog box. Save As dialog box options are the same as the New Folder dialog box options. For further information on these dialog box options, refer above to the New Folder Command.

(*continued*)

Command	Use this command to
Save Folder As	save the open folder under a new name. When selected, this command displays the Save As dialog box. When you save a folder using this command, all programs in the folder will be copied to the desired path. Save Folder As dialog box options are the same as the New Folder dialog box options. For further information on these dialog box options, refer above to the New Folder Command.
Delete Folder	delete a selected folder. This command deletes the folder only; it does not delete any programs listed in the folder. When selected, this command displays the Delete Folder dialog box. Delete Folder dialog box options are the same as the New Folder dialog box options. For further information on these dialog box options, refer above to the New Folder Command.
New Program	create a new BEST++ program. When selected, this command displays the New Program dialog box. This dialog box lists the existing programs and prompts you to name the new program. BEST++ opens a new program and automatically inserts the PROGRAM statement. The new program name will display in the title bar of the Programmer's Environment window.
Open Program	open an existing program to view or edit. If the program is not part of the open folder, you will be prompted to add it to the open folder with the message *Add program to task list?*.

(*continued*)

Command	Use this command to
Save Program	save the open program. If the program was not already saved, this command, when selected, displays the Save As dialog box. Save as dialog box options are the same as the New Folder dialog box options. For further information on these dialog box options, refer above to the New Folder Command.
Save Program As	save the open program under a new name. When selected, this command displays the Save As dialog box. Save Program As dialog box options are the same as the New Folder dialog box options. For further information on these dialog box options, refer above to the New Folder Command.
Remove Program	remove the open program from the currently open folder. A confirmation dialog box will appear. Click on *Yes* to remove the program from the open folder. This process only removes a program name from the open folder's program list. It does not delete the program.
Delete Program	delete a selected program. When selected, this command displays the Delete File dialog box. Delete File dialog box options are the same as the New Folder dialog box options. For further information on these dialog box options, refer above to the New Folder Command.
Print Program	print the current program.
Print Setup	select printer options.
Exit	exit the BEST++ application.

Edit Menu

The following table describes each of the commands that appear in the Edit Menu.

Command	Use this command to
Copy	copy selected text to the clipboard.
Cut	remove the highlighted text. The removed text is saved to the clipboard for future use with the Paste command. The clipboard contains only the most recently cut or copied text.
Paste	insert previously cut or copied text from the clipboard into your program at the current cursor location.
Delete	permanently delete the highlighted text. Deleted text is not stored on the clipboard, and therefore cannot be retrieved using the Copy or Paste commands.
Find	locate a designated text string. Find Dialog Box Options: Find What: Enter the text string you wish to find. Find Next: Start the search and locate the designated text string in the program you are editing. Match Whole Word Only: The search finds only entire words. For example, if you are searching for the word *main*, BEST++ will find *main* in the word re*main*der, unless you have enabled this option. Match Case: Match the text string's capitalization exactly.

(*continued*)

Command	Use this command to
	Find Dialog Box Options (*continued*): Up or Down: Searches from the current cursor location back up to the beginning of the program, or to search from the current cursor location to the end of the program. Cancel: Exits the Find dialog box without executing the Find command
Replace	search and replace one text string with another. Replace Dialog Box Options: Find What: Enter the text string you wish to find and replace. Replace With: Enter the text that will replace the Find What text string. Match Whole Word Only: Have the search find only entire words. For example, if you are searching for the word *main*, BEST++ will find *main* in the word re*main*der, unless you have enabled this option. Match Case: Match the text string's capitalization exactly. Find Next: Begin the search from the current cursor location and locate the next instance of the Find What text string.

(*continued*)

Command	Use this command to
	Replace Dialog Box Options (*continued*): Replace: Replaces the highlighted text with the Replace With text string. Replace All: Changes all instances of the Find What text string. Cancel: Exits the Replace dialog box without executing the Replace command.
Paste Function	access the Paste Function sub-menu (All, Operators, Math functions, Program control). Paste Function, which is designed as a time-saving feature, displays the Paste Function dialog box. This dialog box gives you the ability to view the syntax, identify configuration and maintenance parameters, and display on-line help for each BEST++ keyword. You can then select a keyword and BEST++ will automatically insert the syntax of the selected word into your BEST++ program. If you select *Paste Function* followed by: *All*, the Paste Function dialog box will display all logical operators, and program control functions. *Operators*, the Paste Function dialog box will display all supported mathematical, relational, and logical operators. *Math functions*, the Paste Function dialog box will display mathematical functions (ABS, ACOS, ASIN, etc.). *Program control*, the Paste Function dialog box will display reserved words used to control flow of program logic (AUTO, CALL, DELAY, etc.).

(*continued*)

Command	Use this command to
	Paste Function Dialog Box Options: Select item: Select an item to paste in your BEST++ program. Syntax: Enable to display the required syntax for the selected reserved word. The syntax will display in the lower half of this dialog box. Functions: Enable to display the functions list for the selected reserved word. The functions will display in the lower half of this dialog box. Help: Enable to display help information for the selected reserved word. Help will display in the lower half of this dialog box. Show references: Updates the Select item list to include extended syntax for the selected reserved word. Don't show references: Exits the extended syntax list and returns the Select item list to its previous state. Paste: Pastes the displayed syntax into the currently-displayed BEST++ program. The syntax will be inserted into the program at the current cursor location. Close: Exits the Paste Function dialog box without executing the Paste function command. Help: Displays help information on the Paste Function dialog box.

(*continued*)

Command	Use this command to
	Example: The following example shows how to Use Paste Function to insert CONNECT syntax into an open program automatically. The Connect will be to the setpoint schedule. 1. Select *Edit*, *Paste function*, *All*. 2. In the Paste Function dialog box's Select item list, select *Connect*. 3. Click on the *Functions* radio button and then click on *Show references*. 4. In the Select item list, select *Setpoint*. 5. Then, click on *Paste*. BEST++ will paste the corresponding Connect syntax into the open program at the current cursor location. 6. Click on *Close* to close the dialog box.

BEST++ Menu

The following table describes each of the commands that appear in the BEST++ menu.

Command	Use this command to
Select Comfort Controller	select the target controller. When you select this command, the Select Comfort Controller dialog box displays. After you enter and confirm the address, it displays in the third line of the Programmer's Environment window.
Compile Program	compile the currently-displayed BEST++ program. When you select this command, the compilation process begins. The Compiling window lists the status and results of the compilation process, including any syntax errors detected and the compiled program size.

(*continued*)

Command	Use this command to
	Compiling Window Options: Abort: Stops the compilation. Print: Prints the information in the Compiling window. Close: Closes this window.
Compile All	compile all of the BEST++ programs contained in the open folder. When you select this command and select Yes in the confirmation dialog box, the Compiling window displays the results of each compile process. Programs are compiled in the order in which they are listed in the Tasks drop down list. Compiling Window Options: Abort: Stops the compilation. Print: Prints the information in the Compiling window. Close: Closes this window.
Decompile Program	decompile the currently-displayed BEST++ program. Decompiling can be used during debugging and is especially helpful if you wish to determine how the compiler interpreted your logic.
Make List File	create a decompiled version of the currently open BEST++ program that includes line numbers. You can use this command when debugging your program. When you select this command, the results of the decompile are displayed in the Listing window. Listing Window Options: Abort: Not available for Make List File command. Print: Prints the information in the Listing window. Close: Closes this window.

(*continued*)

Command	Use this command to
Show Compiler Window	This command is currently not supported.
Download Program	send the currently open program from ComfortWORKS to the target Comfort Controller. BEST++ will download the last compiled version of the program. The program will automatically begin to execute after the time specified in the TASK statement's *reschpor* (power on reset) parameter.
Download All	send all of the programs in the currently open folder from ComfortWORKS to the target Comfort Controller. BEST++ will download the last compiled version of each program. The programs will download in the order listed in the folder and will automatically begin to execute after the time specified in the TASK statement's *reschpor* (power on reset) parameter.
BEST++ Debug	access and troubleshoot a program that has been downloaded to the target Comfort Controller. You can list a program's user names (variables, task connects, etc.) and then display a selected name's input functions, or display the current value of supported output functions, maintenance, or configuration decisions. BEST++ Debug also gives you the ability to monitor and evaluate program flow and logic by allowing you to halt, resume, insert breakpoints, and to view and modify program and configuration values. On selecting this command, BEST++ will display the Debug dialog box. Debug Dialog Box Options: The Debug List: Select the user name (variable, task connect, etc.) that you wish to examine. Then click on *Functions*, *Configuration*, or *Maintenance*.

(*continued*)

Command	Use this command to
	Debug Dialog Box Options (*continued*): Functions: Displays the current value of the associated output functions and a list of input functions supported by the selected name. Configuration: Displays the current value of configuration decisions for the selected name. Maintenance: Displays the current value of maintenance decisions for the selected name. Show Description: Enables or disables display of the standard 24-character description in place of the maintenance or configuration decision name. Show Global: Enables or disables display of global dictionary names in the Debug list. When this option is enabled, the Debug list will include local variables (names from program currently being debugged) as well as all global dictionary variables. Edit: Modifies the value of the currently-selected Debug data list data. Note that these edits change data and values of variables as they exist in the Comfort Controller — they do not change values in source BEST++ programs or data in your ComfortWORKS database. Run: Executes the currently-selected task from the beginning. Halt: Stops execution of the currently-selected task. The task will not resume execution until you start it by clicking on *Run* or *Continue*. Any devices controlled by the halted task remain in the last state commanded. (*continued*)

Command	Use this command to
	Debug Dialog Box Options (*continued*): Delay: Delays task execution for the specified number of seconds. The task will delay for the specified time and then continue execution at the point where it was when you selected Delay. Breakpoint: Halts the currently-selected task at the specified line number. To determine line numbers, display a copy of the source program with line numbers by selecting the *Make List File* command from the BEST++ menu. When a task halts at a specified breakpoint, you would typically begin single-stepping line-by-line through the code and examining task flow. You could also select an item from the Debug list and examine its current associated function, configuration, or maintenance data. You can place breakpoints only on lines that contain logic keywords (IF, WHEN, =, DELAY, etc.). For example, you cannot place breakpoints on non-executable lines such as those that begin with STEP or label names. To resume execution, click on the dialog box's *Continue* or *Run* button. The task will resume execution, but will halt again when it reaches the specified breakpoint. You can also resume execution by clicking on Single step, which will cause BEST++ to execute the next program line and stop. If there is a breakpoint set, the breakpoint line number will display in the Debug dialog box. You clear a breakpoint by entering *0* in the Breakpoint dialog box. Single step: Evalutes task flow on a line-by-line basis beginning at the line number that you specify by clicking on Breakpoint.

(*continued*)

Command	Use this command to
	Debug Dialog Box Options (*continued*): BEST++ will execute one line and then stop again. The task will remain stoppped until you resume execution by clicking on the Debug dialog box's *Continue* or *Run* button. The current line number and the associated line of code will display in the Debug dialog box.
Continue:	Causes a task that was delayed as a result of selecting Breakpoint or Halt to resume execution. The task will resume from the breakpoint or the line it was at when halted, and will continue until the program is halted or the breakpoint is reached again. Close: Exits the Debug dialog box. Help: Displays help information on the Debug dialog box.
System Debug	obtain source file copies of and troubleshoot BEST++ programs, as well as to access the following objects in a selected Comfort Controller: I/O channels (hardware and software points), time and holiday schedules, setpoint schedules, runtime and consumable tables, standard algorithms, and system tables. The System Debug command provides you with a window into a Comfort Controller, and gives you the ability to directly access and modify a Comfort Controller's BEST++ programs and objects. On selecting this command, BEST++ will display the System Debug dialog box.

(*continued*)

Command	Use this command to
	System Debug Dialog Box Options: Categories: I/O Channels, Schedules, Setpoints, Data Collection, Standard algorithms, BEST++ programs, BEST++ sources, System. Accesses the selected item in the target Comfort Controller. The Select list will display each item's name and description. You can select an item and examine it using the BEST++ debugger. If you select BEST++ programs, you can also delete all programs by clicking on *Initialize*. If you select BEST++ sources, you can obtain a copy of a selected program's source file by clicking on *Upload*. You can also delete a selected or all programs by clicking on *Delete* or *Initialize.* Refer above to the explanation of the BEST++ Debug command for a description of debugger operation. Delete: Removes the selected BEST++ program source from the Comfort Controller. Upload: Obtains a copy of the selected BEST++ program source from the Comfort Controller and displays it at ComfortWORKS. BEST++ automatically decompiles the program as it is uploaded, and displays an editable copy of the program source in the Programmer's Environment window. Note that if you have the Options menu's Delete Template After Download Command enabled, a copy of the program source will be unavailable for upload. Debug: Accesses the BEST++ debugger and allows you to display configuration or maintenance data or the value of the selected

(*continued*)

Command	Use this command to
	System Debug Dialog Box Options: (*continued*)
	item's associated functions. The Debug dialog box is displayed. Refer above to the explanation of the Debug command for instructions on using the Debug dialog box.
	Initialize: Deletes all of BEST++ programs in the Comfort Controller. If a program is executing when you select Initialize, points forced by the program remain forced.
	Close: Closes the System Debug dialog box.
	Help: Displays help information on the System Debug dialog box.

Options Menu

The table below describes the commands that appear in the Options menu.

Command	Use this command to
Report Warnings	display warnings along with the syntax error messages, which will always be reported, during compilation. If you use the BEST++ menu's Make List File command to request a list file of errors, warnings will also be included there. Note that a program containing warnings will compile and download correctly. However, the program could operate improperly. An explanation of errors and warnings can be found in this manual's Syntax Error Messages chapter.
Metric	indicate whether to display data in metric engineering units instead of customary US engineering units. Note that the Debugger's Function data is not affected by this setting.
Delete Template After Download	delete the BEST++ program template from the Comfort Controller after downloading.

(*continued*)

Command	Use this command to
	This command is intended for use primarily by Carrier Corporation representatives as a security measure to prevent access to proprietary BEST++ programs. **Note:** You are strongly cautioned against using the Delete Template After Download command, as deleting a program template erases the BEST++ program image and permanently prevents program access. You will not be permitted to upload or decompile the program.
Show Units Names	display the allowable entries for defining the engineering units of constants. Refer to Appendix B of this manual for additional information on this topic.
Save Settings On Exit	indicate whether to automatically save any changes you make by using commands on the Options menu. These saved options will be in place the next time you launch the BEST++ application. When an option is in effect, a check mark appears next to the command on the Options menu.

Help Menu

The table below describes the commands that appear in the Help menu.

Command	Use this command to
Contents	display the help table of contents.
Search	search for a help topic by typing a keyword.
About	display program information, version information, and copyright.

Index

Index

D

E

F

G

H

I

K

Q

R

S

T

Reader's Comments

Your comments regarding this manual will help us improve future editions. Please comment on the usefulness and readability of this manual, suggest additions and deletions, and list specific errors and omissions.

Document Name: _____ **Publication Date:** _____

Usefulness and Readability:

_____

_____

_____

Suggested Additions and Deletions:

_____

_____

_____

Errors and Omissions (Please give page numbers):

_____

_____

_____

Date: _____

Name: _____

Title or Position: _____

Organization: _____

Address: _____

Fold so that the mailing address is visible, staple closed, and mail.

Carrier Corporation
Carrier World Headquarters Building
One Carrier Place
Farmington, CT 06034-4015

Attn: CCN Documentation