Section 1
Introduction

WARNING
Incorrect programming of I/O statements could cause the system to malfunction. Follow the instructions in this manual and check your I/O statements carefully before downloading them to the panel.

1.1 What is I/O Application Programming?

One of the many powerful features of the 4820/21 is its built-in programming language that allows you to create customized applications (or programs) based on users' needs. These programs are called I/O (for input/output) applications. You can think of I/O applications as a way to make use of 4820/21 system data (or events that occur on the 4820/21) to meet specialized needs of your customers. For example, a commercial customer might want an application that would bypass a particular zone if a particular door has been accessed.

Examples of typical residential applications are: Turning on lights at specific times to make a home seem occupied when it is vacant or turning on lights during an entry delay (the timer needed for this application is part of the program, eliminating the need for a timing device).

1.1.1 Controlling X10 Modules

The 4820/21 can interface with X10 modules via the Model 4880 Status Output Module. This allows you to write applications to control lights or appliances based on internal status and key commands. For example, you could write a program that would turn lights on when someone enters a building. Programs that turn lights and appliances on and off are common uses of X10 modules. (The 4820/21 supports up to 16 X10 house codes with 16 devices per house code.)

1.1.2 Controlling 4880 Status Output Module Outputs

I/O statements are commonly used to control the outputs on the 4880. For example, you could write an application that would cause the 4880 to output when the system arms. (See Section 4.4 for an example of this type of application.)

1.2 About this Manual

The purpose of this manual is to teach you how to use the programming language. Much of the instruction is through examples. We have tried to include examples that are applicable to many different installations, but because there is virtually an unlimited number of applications that could be created, we couldn't possibly include them all. Once you learn the basics from analyzing these examples, you should have no trouble creating your own applications.

1.2.1 For More Information

The 4820/21 I/O application programming language is a subset of a programming language called "C", the language that was used to write the 4820/21 control software. C is widely used in technical and business applications. There are literally hundreds of books about C on the market. *The Waite Group's New C Primer Plus* by Mitchell Waite and and Stephen Prata (published by SAMS, a division of Prentice Hall Computer Publishing, (1990), has become a standard introductory book on C and is an excellent resource for beginners.

If you do read more about C (or if you are already familiar), you will see that the 4820/21 I/O application language does not contain all features of the standard C language and uses some features differently than they are typically used in standard C. Follow the syntax as described in this manual when creating I/O applications.

Section 2
The I/O Programming Language

The I/O programming language is made up of a set of keywords and symbols that must be used in the correct "syntax." Syntax is the way in which statements are constructed and organized so they can be understood by the 4820/21 panel.

2.1 Event-Driven Programming

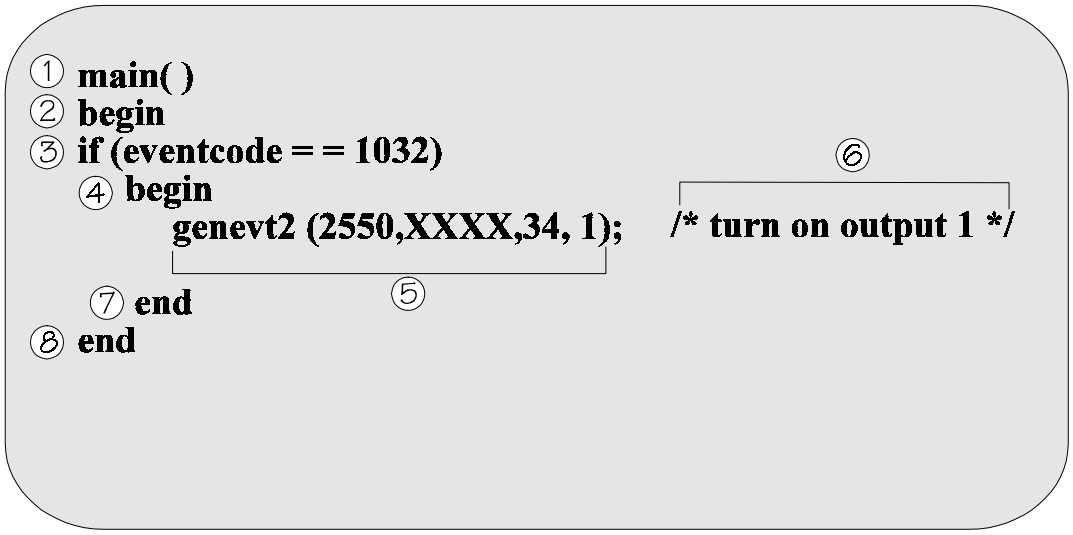
The I/O application programming language uses a technique called event-driven programming. Event-driven programming is unique from other types of programming (including the I/O application language used with the 4724) because it does not read input and make decisions about how to respond to that input. Instead, an event-driven program waits for notification from the 4820/21 operating system that a particular event has occurred, then responds by generating a new event.

The events that I/O applications wait for are "packets" of information consisting of:

eventcode	a number that identifies the event
partition number	a number that identifies where the event occurred
parameters (optional)	additional information about the event

2.2 Simple Script Example

The group of statements that makes up an I/O application is called a script. This section of the manual contains two diagrams annotating a simple script. The terms used in the examples are described in greater detail in Section 2.3.



① Always the first line of an I/O application.

② Begin program.

③ Conditional statement. See Section 2.3.3.

④ Begin to execute the following command if the condition is met.

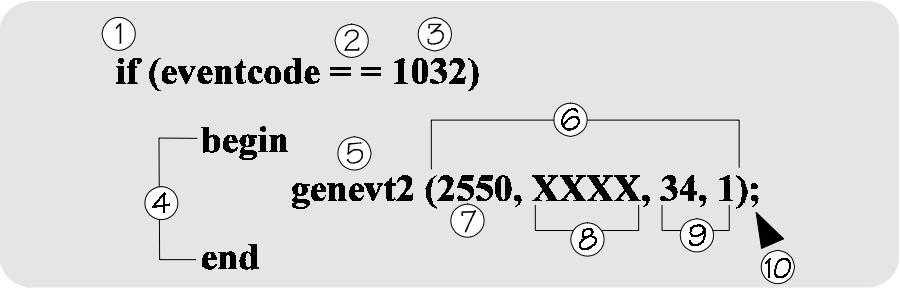
⑤ Command statement. See Section 2.3.4.

⑥ Comment. Describes the command. See Section 2.3.9.

⑦ End of this command.

⑧ End the program.

Figure 2-1 Sample script



① Example of a simple if condition statement.

② Equality operator.

③ *1032 is the eventcode for "area manually armed". The if statement checks to see if event 1032 has occurred.

④ "begin" and "end" are needed to open and close a command statement.

⑤ genevt2 command word is used in this case because this application uses two parameters.

⑥ *Parameters associated with genevt command must be enclosed in parentheses.

⑦ *2550 is the eventcode for the "output on" command (used to turn on a 4880 module).

⑧ A partition number is required for genevt syntax, but is not needed for this sample application, so "XXXX" (the dummy partition number) is used instead of an actual number.

⑨ Parameters associated with eventcode 2550.

⑩ Semicolon required to end statement.

*Eventcodes and parameters are listed in Section 5 of the manual.

Figure 2-2 Annotated fragment of a simple program

2.3 Keywords and Symbols

Table 2-1 is a list of all elements of the I/O application programming language. These items are described in greater detail in the sections that follow.

2.3.1 Quick Reference Chart

Table 2-1: Keywords and Symbols

Keyword / Symbol	Description	Usage Example
main()	This must be the first line of every I/O application.	Used one time only in each script.
begin	First line of all statements or groups of statements.	See Section 2.3.11.
end	Last line of all statements or groups. The last word of an I/O application must be “end.”	See Section 2.3.11.
eventcode	A four-digit number standing for an event that will trigger an action by an I/O application. Section 5.2 lists all eventcodes.	if (eventcode == 1000)
partition	Partition number where an event occurred. Must be included any time the genevt command word is used. Valid range is 1-8, no leading zeros.	if (partition == 4)
param1 - param4	A parameter or additional piece of information associated with an eventcode.	if ((eventcode == 1233) & (param1 == 2))
genevt0	Command word. Used with eventcodes that have 0 parameters. Requires a partition number. (XXXX can be used when your application does not need to know the partition number.)	genevt0(9999,XXXX);
genevt1	Command word. Used with eventcodes that have 1 parameter. Requires a partition number. (XXXX can be used when your application does not need to know the partition number.)	genevt1(9999,XXXX,param1);
genevt2	Command word. Used with eventcodes that have 2 parameters. Requires a partition number. (XXXX can be used when your application does not need to know the partition number.)	genevt2(9999,XXXX,param1,param2);
genevt3	Command word. Used with eventcodes that have 3 parameters. Requires a partition number. (XXXX can be used when your application does not need to know the partition number.)	genevt3(9999,XXXX,param1, param2, param3);
temp1 - temp32	16-bit locations. For temporary storage of data needed by the application while executing.	temp1 = 0;
if	Beginning of a conditional statement.	See Section 2.3.3.
if else	Beginning of second conditional statement.	See Section 2.3.3.
else	Action to be taken if conditional statement criteria *not* met.	See Section 2.3.3.

Table 2-1: Keywords and Symbols

Keyword / Symbol	Description	Usage Example

BIT1 - BIT16	Decimal value of a 16 bit "word" when one of the bits is on. Must be entered as shown here (all capital letters). Use when you want to differentiate between particular bits. (See Section 2.4.1 for more information about words.)
	Note: If an account is uploaded, these keywords will be replaced with their numeric values. See *Appendix A* to this manual for more information.
		EXAMPLE 1:
		To access Bit 4:
		if ((param1 & BIT4) == (BIT4))
		EXAMPLE 2:
		To access Bit 4 or Bit 5:
		if ((param1 & (BIT4 | BIT5)
		== (BIT4 | BIT5)))

XXXX	A dummy partition number. Used with genevt command when you do not care which partition the event occurred in. Must be entered as shown here (all capital letters). Note: If an account is uploaded, this keyword will be replaced with its numeric value. See *Appendix A* to this manual for more information.	genevt0(9999,XXXX);
HOUSEA - HOUSEP	Decimal value of an X10 module house number. (There are 16 possible with names HOUSE*A* through HOUSE*P*.) Must be entered as shown here (all capital letters). Use with UNIT1 - UNIT16 to access X10 devices. See below. Note: If an account is uploaded, these keywords will be replaced with their numeric values. See *Appendix A* to this manual for more information.	(HOUSEA + UNIT1)
UNIT1 - UNIT16	Decimal value of an X10 device number. Must be entered as shown here (all capital letters). Use with HOUSEA - HOUSEP (see above) to access X10 devices. Each HOUSE_ can have up to 16 units assigned to it. Note: If an account is uploaded, these keywords will be replaced with their numeric values. See *Appendix A* to this manual for more information.	See above.
;	A semicolon must follow all command statements using genevt and assignment operators (equal symbols).	genevt0(9999,XXXX); temp1 = 0;
,	A comma separates multiple parameters.	genevt3(9999,XXXX,param1,param2, param3);
( )	Parentheses are for grouping or separating expressions.	if ((eventcode == 1233) & (param1 == 2))
/*	Begin comment. A comment is an explanation of a statement. Words surrounded by comment markers are not read as part of the application.	/* This is a short comment. */ /* This is a long comment. It wraps over several lines. Make sure no statements are embedded in your comment. */
*/	End comment.	See above.

2.3.2 Relational and Mathematical Operators

In addition to the keywords and symbols in the *Quick Reference Chart*, the relational and mathematical operators shown in the table below can also be used.

The compiler evaluates relational and mathematical expressions from left to right with anything in parentheses having priority. See Section 2.3.15 for an example.

Table 2-2: Operators

Operator	Use
&	AND operator. Use when you have two conditions that must both be true. (Also used for ANDing two bits. See Section 2.4.2 for more information.)
|	OR operator. Use when you have two conditions and *only one* of them must be true. (Also used for ORing two bits. See Section 2.4.2 for more information.)
^	XOR operator. Use when have two conditions that must be different from each other. (This is the carat character, located above the "6" on the top row of the keyboard.) (Also used for XORing two bits. See Section 2.4.2 for more information.)
=	Assignment operator. Use when you need to assign a variable (allocate space). Do not confuse with = = (two equal symbols, see below). Statements using this operator must be ended with a semicolon.
= =	Equality operator (two equal symbols, no space between them). Compares two statements to see if they are equal.
!=	Not equal operator. Compares two statements to see if they are *not* equal.
>	Greater than operator.
<	Less than operator.
>=	Greater than or equal to operator.
<=	Less than or equal to operator.
>>	Shift right operator. Use when you need to access the low byte of an event-code. See Section 2.4.3 for more information.
<<	Shift left operator. Use when you need to access the high byte of an event-code. See Section 2.4.3 for more information.
+	Addition operator.
-	Subtraction operator.
*	Multiplication operator.
/	Division operator.

2.3.3 Conditional Statement

Conditional statements always begin with the word(s) “if”, “if else”, or “else”. They tell the 4820/21 what condition to look for in order to execute the command.

Example 1: One condition statement

```
if (eventcode == 1000)
   begin
      the system will do something;
   end
```

Example 2: Two condition statement

```
if (eventcode == 1000)
   begin
      the system will do something;
   end
else if (eventcode == 1001)
   begin
      the system will do something else;
   end
```

Example 3: Two condition statement with an else statement

```
if (eventcode == 1000)
   begin
      the system will do something;
   end
else if (eventcode == 1001)
   begin
      the system will do something else;
   end
else
   begin
      neither condition was true, the system will do this instead;
   end
```

2.3.4 Command Statement

Command statements tell the system what to do if the condition statement is true. There are two types of command statements: genevt commands and assignment commands. See Sections 2.3.5 and 2.3.6 for more information.

2.3.5 Genevt commands

A genevt command tells the system to *generate* an event. There are four commands, depending on the number of parameters that w

genevt0	generate an event with zero parameters
genevt1	generate an event with one parameter
genevt2	generate an event with two parameters
genevt3	generate an event with three parameters

Genevt commands require information that follows—eventcode, partition number (or dummy partition number), any parameters—to be enclosed in parentheses. Figure 2-3 and Figure 2-4 show the form that genevt commands take.

The number of parameters used with genevt depends on how many parameters are available for the eventcode (from Section 5 of this manual) and whether or not you want to use the parameter.

Not all eventcodes can generate commands. Some eventcodes are for use in conditional statements only. See Section 5 for more information.

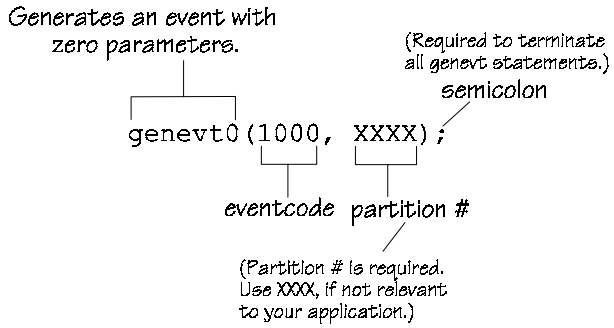


Figure 2-3 Example genevt0 command

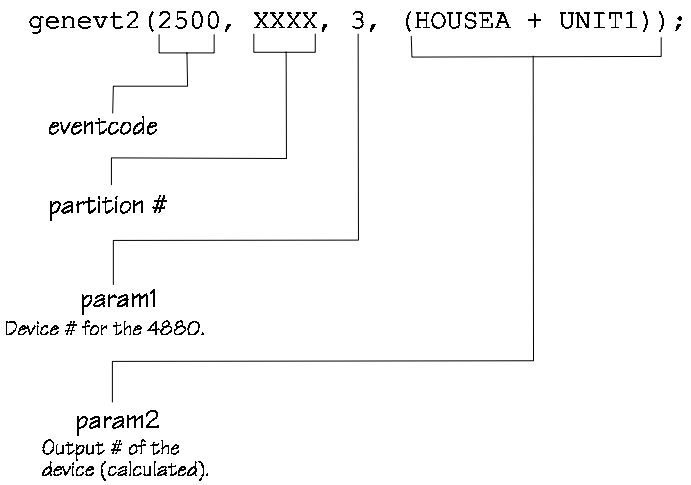


Figure 2-4 Example genevt2 command structure

2.3.6 Assignment statement

Temporarily assigns a value to a variable. In I/O application scripts, assignment statements are often used to set the initial value of a variable (see below) but they could have many other uses.

```
if (eventcode == 1000)
   begin
      temp1 = 60;    /* assignment statements are command statements
                           so they must end with a semicolon (;)       */
   end
```

2.3.7 Eventcode

An eventcode is a number that represents an event that occurred or will occur on the system. These events drive your I/O applications. Some eventcodes are used only to provide status information for conditional statements. Some eventcodes can generate commands in conjunction with the genevt command. See Section 5 for more information.

2.3.8 Parameter

A parameter is an additional piece of information that provides more specific information about an eventcode. For example, eventcode 2500 indicates that an X10 device is on. There are two parameters associated with this eventcode, param1 indicates the device number of the 4880 where the X10 module is attached; param2 indicates the House Code and Unit Number. An example of the form you would use is shown below.

```
genevt2 (2500, XXXX, 3, (HOUSEA + UNIT1));
```

In this example, param1 is “3” and param2 is the calculation “HOUSEA + UNIT1”.

Parameters are optional and not all eventcodes have parameters assigned to them. See Section 5 for a list of all eventcodes and associated parameters.

2.3.9 Comments

Commenting is an important part of an I/O application script. It tells you and anyone else who may need to use your script how the application works. It is a good idea to write a comment for each statement in your program. The text of your comment must begin with the special characters "`/*`" and end with the special characters "`*/`".

Comment Example

```
genevt2 (2550,1,34, param1);
/* Above statement turns on 4880 outputs */
```

Examples of comments used in scripts appear in Section 4.

2.3.10 Case Sensitivity

The I/O programming language is case-sensitive. This means that if you use a capital letter when a lowercase is required (or vice versa), you will receive a syntax error. Most of the time you will want to use all lowercase for your scripts. There are a few exceptions using all upper-case. The *Quick Reference Chart* (Section 2.3.1) shows all keywords in the correct case.

2.3.11 Begin / End

All command statements (and groups of statements) must begin and end. Also, the script itself must open with begin and close with end. Syntax errors often occur from too few or too many begin/end statements. An easy way to check is to count the number of begins and ends in a script to make sure they match as shown in Figure 2-5.

```
main()
  begin
    if ((eventcode == 501) & (param1 == 20))
      begin
        temp1 = 1;
      end
    if ((eventcode == 552) & (param1 == 20))
      begin
        temp1 = 0;
      end
    if (((eventcode == 1233) & (param1 == 1)) & (temp1 == 1))
      begin
        genevt2(2500,XXXX,3, (HOUSEA + UNIT1));
      end
  end
```

Figure 2-5 Begin / End Syntax

2.3.12 Script

Script is the name for the complete set of commands that runs an application. Each sample application in Section 4 is a script. Some scripts have more than one function; there is only one script per account.

2.3.13 Variables (temporary storage locations)

A variable is a temporary storage location for data that will change or needs to be checked during execution of an application.

The I/O application programming language has several variables that can be used only for particular types of data.

There are also 32 variables, named temp1 - temp32 that can store any data. You might use a temp variable to, for example, keep track of how much time has elapsed in a timer application.

Name	Use For
HOUSEA - HOUSEP	House Number for X10 modules
UNIT1 - UNIT16	X10 module device numbers
temp1 - temp32	Any 16-bit data

2.3.14 Constants

Constants are values that have been given pre-assigned names to make them easier to work with. The data is always the same. For example, BIT8 is the predefined name of the eighth bit of a byte, which has the value 128. XXXX, the predefined name for a dummy partition number, which has the value 15. (A dummy partition number is used with the genevt command, which requires a partition number, when your application does not need to know the partition number.)

2.3.15 Parentheses

Parentheses (parens) are necessary to separate groups of statements.

Not enough or too many parens is a common syntax error. Make sure that every open paren has a close. One easy way to check is to count the number of open parens and the number of close parens. You should have the same number of both, as shown in Figure 2-6.

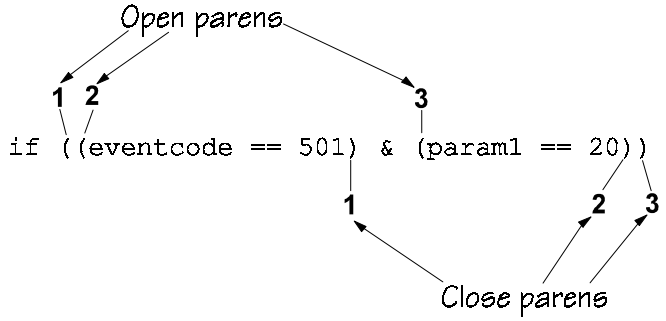


Figure 2-6 Use of parentheses

Parentheses have the highest priority for order of operation. This means that the compiler evaluates statements within parentheses first, starting with the innermost parentheses and moving from left to right. Figure 2-7 provides a simple example.

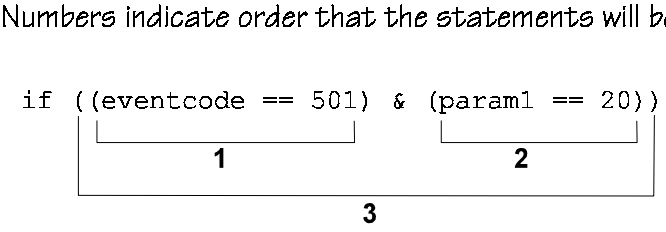


Figure 2-7 Order of operations using parentheses

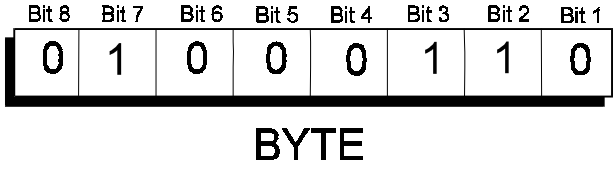
2.4 Special Data Handling

Most eventcodes and parameters used in I/O applications are simple. You just enter the eventcode number and associated parameters in the correct syntax. In a few cases, accessing the data you need is complicated by how that data is stored internally. Besides the eventcode, you may need to know additional information (such as the specific *bits* or whether the data is stored in the *high* or *low byte*).

This section of the manual briefly explains the simple computer math concepts you need to know to handle special data. Section 5 of this manual tells you which eventcodes and parameters require special handling.

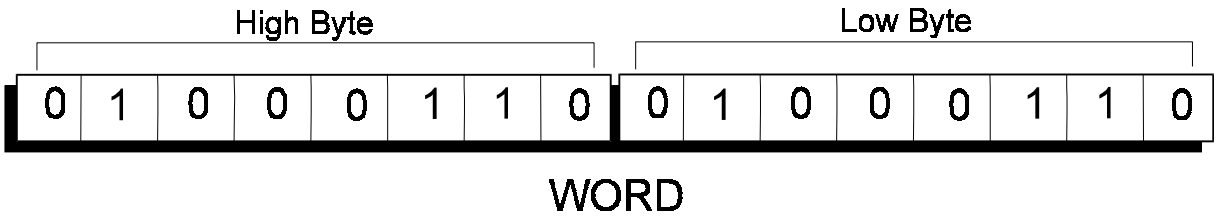
2.4.1 Bits, bytes, words

Internally, the 4820/21 stores data in the form of bytes and, in some cases, words.



A byte is made up of 8 bits. A bit is simply a 0 or a 1. The pattern of bits determine what data is stored in the byte. For some applications you will need to identify which bit is “on” (set to 1) or “off” (set to 0). See Section 2.4.2 for more information.

Note: If you are familiar with other programming languages, you may be used to seeing bits identified as 0-7, instead of 1-8.

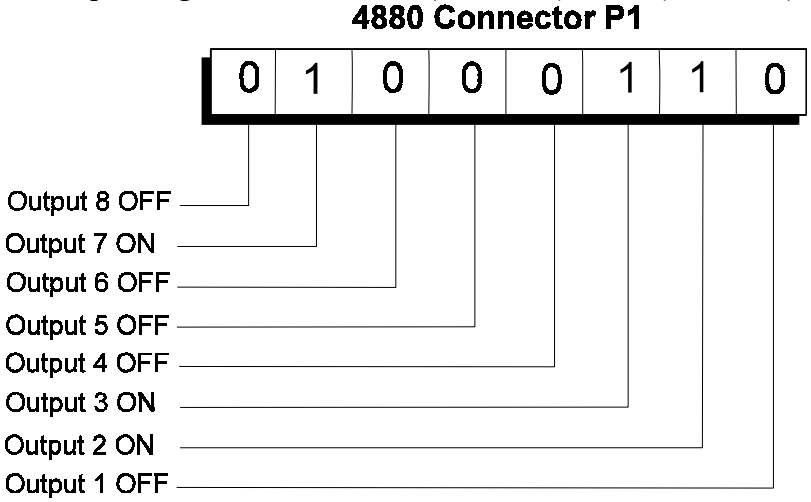


The 4820/21 uses what is a called a *word* to store long data. A word is two bytes, with one byte designated as the *high byte* and the other as the *low byte*. Some I/O applications need to know the location within the word (high byte or low byte) where the data is located. See Section 2.4.3.

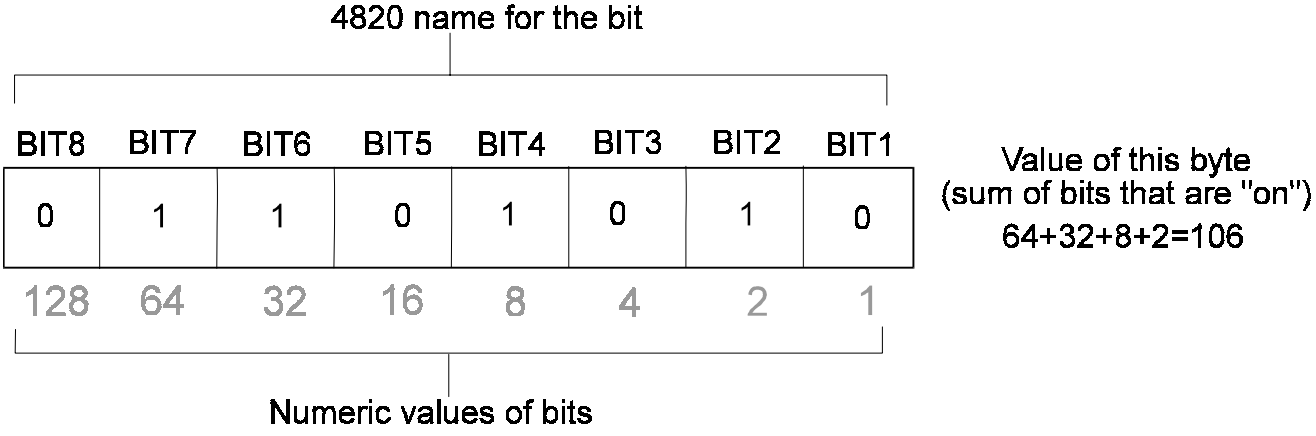
2.4.2 Bit operations

In some cases, I/O applications control a hardware device by sending it a byte of data. Each bit of the byte has a specific purpose. An I/O script can make use of the individual bits to activate commands. In these cases, your applications need to tell which bits are relevant.

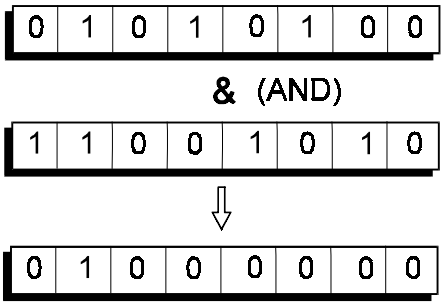
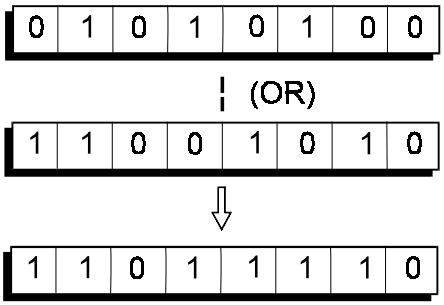
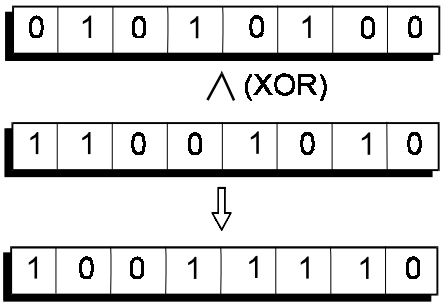
An example is the 4880. As the diagram below shows, each pin of Connector P2 on the 4880 has a specific use depending on if the bit is on (set to “1”) or off (set to “0”).



As the diagram below shows, bits have numeric values. The value of a byte is the total of the bits that are “on” (set to 1). You can use mathematical calculations and comparisons to determine if bits are on or off.



Some typical uses of bit operations are described here. Section 5 of this manual tells you if the parameter you want work with requires bit operations.

BIT Operator	Usage
& (AND)	In addition to its use as a logical operator as described in Section 2.3.2, AND is often used to turn all bits off that are currently on. 
| (OR)	In addition to its use as a logical operator as described in Section 2.3.2, OR is often used to turn a particular bit on without changing the other bits. 
^ (XOR)	In addition to its use as a logical operator as described in Section 2.3.2, XOR is often used to *toggle* bits (turn on if off, or turn off if on). 
+	Adding the value of bits is often used to turn them on (similar to ORing).
<<	Shift left (<<) operator. Moves bits to the high byte. See Section 2.4.3 for more information.
>>	Shift right (>>) operator. Moves bits to the low byte. See Section 2.4.3 for more information.

The chart below is what is known as a *truth table.* It shows all possible results of bit calculations using XOR, OR, AND.

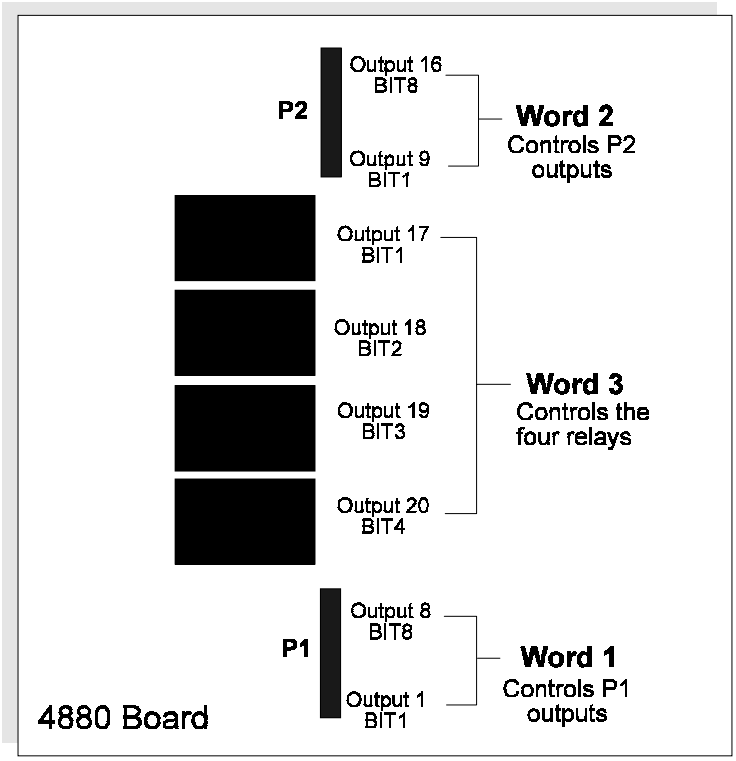
Input		Result		
"A"	**"B"**	**AND**	**OR**	**XOR**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

EXAMPLE:

When Input A and Input B are both 0, ANDing them results in 0.
When Input A is 0 and Input B is 1, ANDing them results in 0.
When Input A is 1 and Input B is 0, ANDing them results in 0.
When Input A and Input B are both 0, ANDing them results in 1.

2.4.3 Accessing high or low bytes

The 4880 provides an example of why you may need to access the high or low byte of a word. Eventcodes 2555 and 2556 are for controlling more than one output at a time. When you use these codes, you must identify a specific byte to show which group of outputs the ones you want to control belong to.



For Each Word

High Byte: Location of the group of outputs

Low Byte: Bits that control the outputs

EXAMPLE:
Turn on outputs 17 and 18.

```
genevt2(2555, XXXX, 1, (3 << 8) + (BIT1 + BIT2));
```

HIGH BYTE

LOW BYTE

Event to turn on more than one output.

Partition # (Not relevant in this case.)

Device # (Assume Device 1 for this example.)

Move the bits into the high byte so they can control the outputs. ("3" represents Word 3, the location of the bits for Output 17 and 18.)

Add the bits that represent the outputs you want to turn on.

2.5 Creating Aux Menu Items

Through I/O programming you can your create own menu items that will activate when the user presses a numeric key on the touchpad. When the key is pressed, the touchpad will display the message you created and the system will perform the action associated with the key.

You can create up to 10 Aux Menu items, corresponding to keys - . Each of these menu items is associated with text that displays on the LCD when the key is pressed. The messages can be up to 16 characters in length (any character is acceptable).

The special eventcodes are shown below. (See also example on next page.)

9000	User pressed touchpad key 0.
9001	User pressed touchpad key 1.
9002	User pressed touchpad key 2.
9003	User pressed touchpad key 3.
9004	User pressed touchpad key 4.
9005	User pressed touchpad key 5.
9006	User pressed touchpad key 6.
9007	User pressed touchpad key 7.
9008	User pressed touchpad key 8.
9009	User pressed touchpad key 9.

EXAMPLE APPLICATION

Suppose you want to create an I/O script that turns on a light attached to an X10 module when the user presses . You also want the touchpad to display “LIGHT ON” when is pressed. Here are the steps.

1. Create a I/O script similar to the one shown in the partial script below.

```
...
if (eventcode == 9001)                    /* if touchpad key 1 is pressed
   */
   begin
      genevt2(2500,XXXX,3,  (HOUSEA + UNIT1)) /*turn on X10 device
   HOUSE1,UNIT1
                                                 located on 4880 Device ID
   3 */
   end
...
```

2. Create the Aux Menu text that will display on the touchpad LCD when the menu item is activated. Use the Aux Menu option to enter the text “LIGHT ON”. Aux Menu text displays can be up to 16-characters long; any characters can be used. The Aux Menu option is available from the General System Options screen of the System SubMenu. (See the 5580 manual if you need more information.)

2.5.1 Activating Aux Menu Items

Users activate Aux Menu items by entering [2] [ENTR] [*Code] to activate the Aux Menus, then pressing [0] - [9] as needed for their specific applications.

*Note: *Normal User and higher*

Section 3
Using the Script Editor

I/O application scripts are created through the 5580 Upload-Download Software and are downloaded to the panel.

3.1 Entering the Programmable I/O Menu

To access the text editor that allows you to create application scripts, follow these steps.

1. From the 5580 Main Menu, select Accounts.
2. Choose an account to Edit or Create a new account.
3. At the System SubMenu, select Programmable IO. A screen for creating a script displays. Working with this screen is similar to using a text editor or simple word processor. Section 3.2 describes the editing keys that are available. The default script that displays depends on the template that was used to create the account. If the “System Template” was used, a script that controls bell outputs according to UL requirements will be the default. See Section 3.1.1 for more information.

3.1.1 UL Required Script for Controlling Bells

If you are installing a UL system, the script for controlling bell ouputs must be part of the account. This script is automatically included for any accounts that are created using the System Template. If you are installing a UL system and you need additional statements, you can add them to the end of the UL script as shown in Figure 3-1.

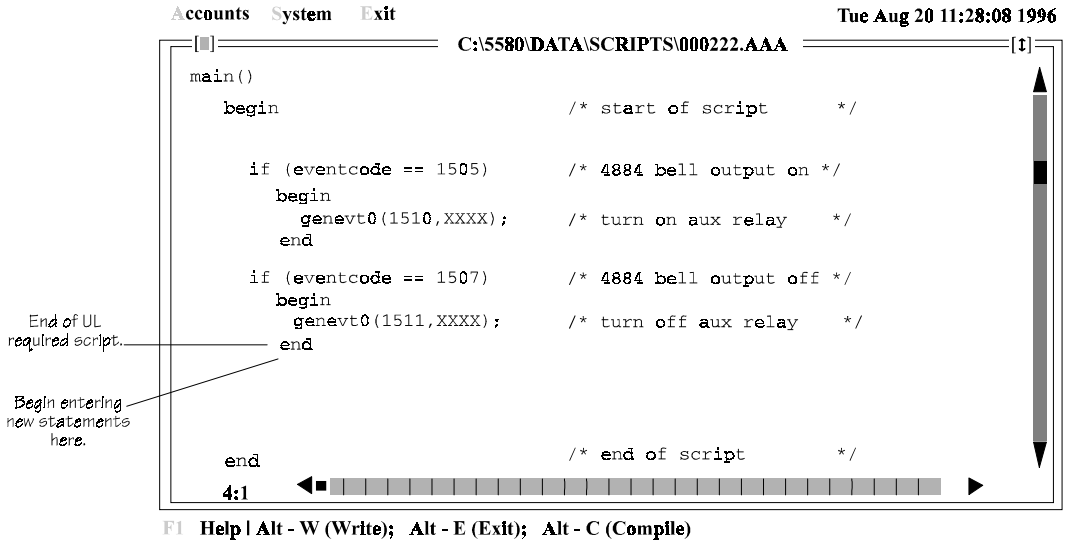


Figure 3-1 Modifying the UL required script

3.2 Objects on the Script Editing Screen

Figure 3-2 shows the objects that appear on the script editing screen. Editing keys are described in Section 3.3.

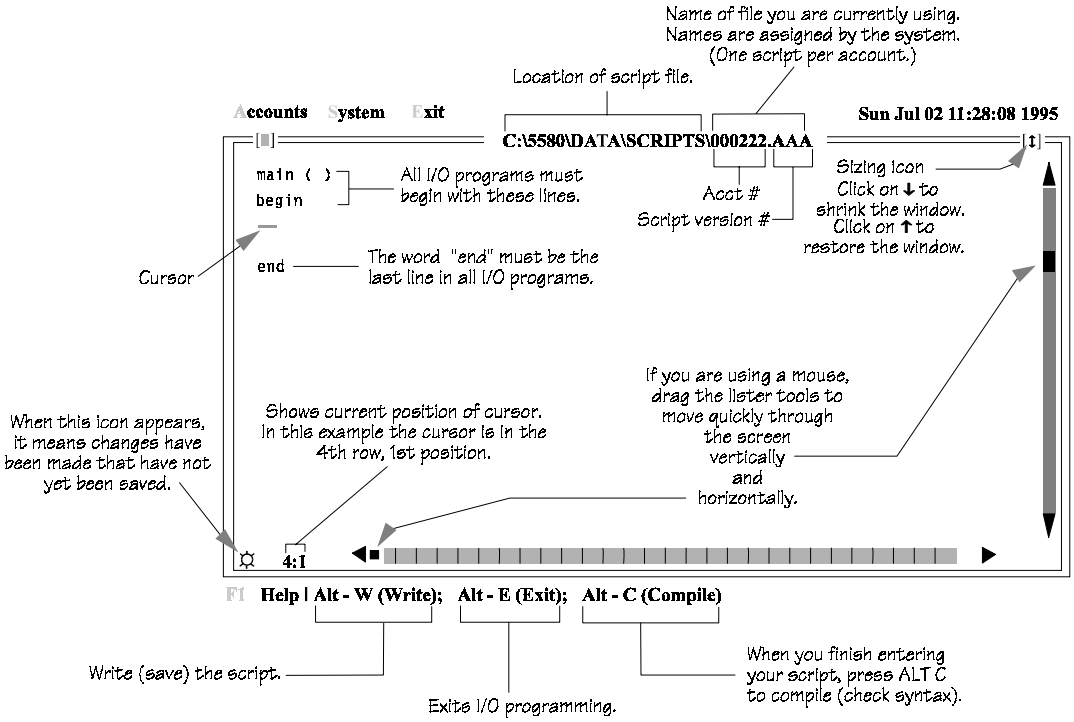


Figure 3-2 Objects on a Script Editing Screen

3.3 Editing Keys

The keys shown in Table 3-1 are available on the script editing screen.

Table 3-1: Editing Keys

Menu Keys	
ALT W	Write (save) the script.
ALT C	Compile (syntax check).
ALT E	Exit the editing session.
ALT U	Undo last change.
Highlighting and Editing Keys	
SHIFT ← or →	Selects previous or next character.
SHIFT ↑ or ↓	Selects previous or next line.
SHIFT PgUp	Select the previous screen.
SHIFT PgDn	Select the next screen.
SHIFT Home	Select from cursor position to the beginning of the line.
CTRL Y	Delete the current line.
SHIFT Delete	Cut the selected text
CTRL Ins	Copy to the clipboard (see Section 3.3.1 for more information).
SHIFT Ins	Paste from the clipboard (see Section 3.3.1 for more information).
SHIFT End	Select from cursor position to the end of the line.
Cursor Movement Keys	
← → ↑ ↓	Move through the screen in the direction indicated.
Tab	Move to the left by 8 (tab positions cannot be changed).
CTRL PgUp	Move to the beginning of the script.
CTRL PgDn	Move to the end of the script.
CTRL ← or →	Move to the previous or next word.

3.3.1 Copying to the Clipboard

The script editing screen has a “clipboard” (memory available for you to store text temporarily, similar to most Windows applications). You can use it to copy a script from one account or template into another.

1. When you are in the script that you want to copy *from*, highlight the text you want to copy.
2. Press CTRL Ins. The text is now on the clipboard.
3. Open the script that you want to copy *to.*
4. Press SHIFT Ins. A copy of the text should now appear in the script editing screen you are currently working with.

(You can also copy text within the same script. The steps are the same except that at Step 3 just move to the location you want to copy *to*.)

3.4 Writing (Saving) the Script

Press ALT W to save the script at any time. (No message displays on the screen, but your script will be saved.)

3.5 Compiling the Script

When you have entered and saved the script, press ALT C to compile (run the syntax check). If you have not written (or saved) the script before you select Compile, you will be prompted to do so. You cannot compile a script that has not been saved.

If your script does not contain any syntax errors, the message “Syntax checks OK!” displays. Your application should be able to run.

3.5.1 Syntax Errors

The program will not be able to compile if a syntax error exists. A Syntax Error message box displays, showing you the line number where the error occurred and a brief description of the error. Syntax messages are not always meaningful (because the compiler is attempting to understand your program, including any typing errors you may have made). The vast majority of syntax errors are typos, spelling errors, missing punctuation, and so on. Figure 3-3 shows examples of syntax errors and what caused them.

The compiler stops immediately and displays the syntax error message as soon as it encounters an error. If your script has more than one syntax error, you may need to compile several times.

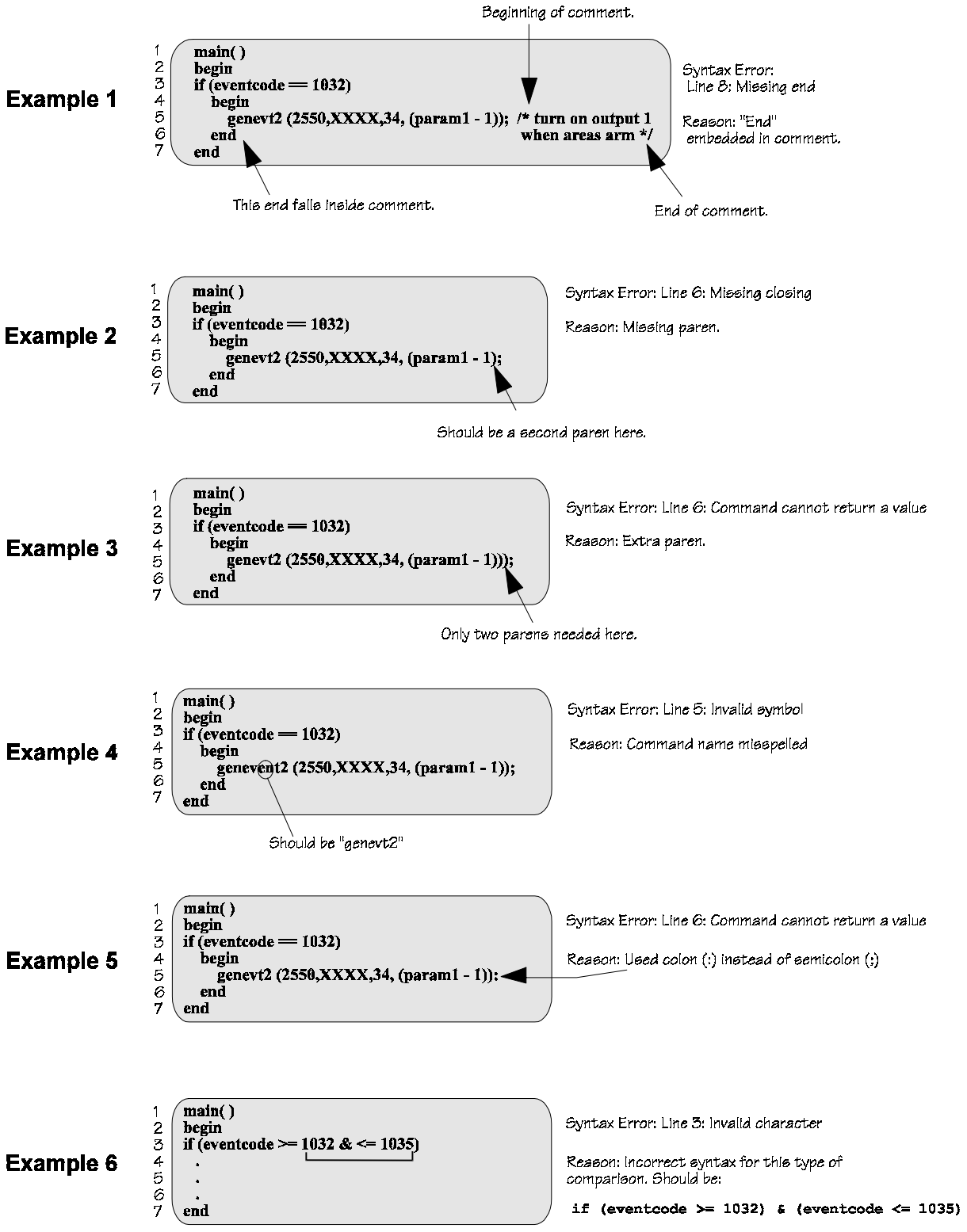


Figure 3-3 Sample Syntax Errors

4.6 Timer Routine

The script below is a timer routine that can be customized for many different applications.

```
main()
begin
   if (eventcode == ?????)                      /* select event that
                                                   begins timer     */
      begin
         temp1 = 60;                            /* begin timer, duration
                                                   is 60 seconds in this
                                                   example   */
         genevt2(2550,XXXX,3,1);                /* turn on first output
                                                    of device #3    */
      end
   if ((eventcode == 1200) & (temp1 != 0))     /* check timer once per
                                                  second until zero    */
      begin
         temp1 = (temp1 - 1);                  /* count down timer by 1 */
            if (temp1 == 0)                     /* if timer is equal to
                                                    zero  */
               begin
                  genevt2(2551,XXXX,3,1);       /* turn off first output
                                                  of device #3     */
               end
      end
end                                             /* end of script     */
```

4.5.2 Time Schedule Triggers Vacation Start

In this sample script, Time Schedule 2 is the schedule for the vacation. Time Schedule 1 is the "night time" schedule, the hours that the user wants house lighting on. This application is somewhat simpler than the one described in Section 4.5.1 with the possible disadvantage that the user does not trigger the vacation.

```
main()
begin
   if ((eventcode == 1233) & (param1 == 2))     /* checks if vacation
   started */
      begin
         temp1 = 1;                            /* if yes, turn flag on */
      end
   if ((eventcode == 1234) & (param1 == 2))     /* checks if vacation
   ended */
      begin
         temp1 = 0;                             /* if yes, turn flag off
   */
         genevt2(2501,XXXX,3, (HOUSEA + UNIT1)); /* turn X10 #1  off
      end
   if (((eventcode == 1233) & (param1 == 1)) & (temp1 == 1))
                                              /* if vacation is active
                                                 and it's night time,
                                                 turn X10 #1  on      */
      begin
         genevt2(2500,XXXX,3, (HOUSEA + UNIT1));
      end
   if (((eventcode == 1234) & (param1 == 1) & temp1 == 0))/* if vacation
                                                          is inactive
                                                          or it's day
                                                          time, turn
                                                          X10 #1 off*/
      begin
         genevt2(2501,XXXX,3, (HOUSEA + UNIT1));
      end
end                                                  /* end of script */
```

4.5 Vacation Lights Programs

The two scripts in this section are for making a home seem occupied during a vacation.

4.5.1 Phantom Zone Bypass Triggers Vacation Start

In this sample script, Zone 20 is configured as a “phantom zone” (no wires are connected to the hardware point assigned to Zone 20). Through programming, the zone is configured so that it will always be ready whether the system is armed or disarmed. The bypass state of the zone is used as a trigger. When the user bypasses Zone 20, this application will execute. Time Schedule 1 is the “night time” schedule, the hours that the user wants house lighting on. When the user returns from vacation, unbypassing Zone 20 triggers the end of the vacation. The lights will no longer turn on and off automatically. The hardware device number for the 4880 is 3.

```
main()
   begin                                          /* start of script */
      if ((eventcode >= 501) &
      (eventcode <= 518) & (param1 == 20))   /* checks to see if
                                               zone 20 is bypassed */
      begin
         temp1 = 1;                           /* set “vacation mode”
                                                 flag              */
      end
      if ((eventcode >= 552) &
      (eventcode <= 569) & (param1 == 20))    /* if zone 20 is
                                                 unbypassed        */
      begin
         temp1 = 0;                            /* clear vacation mode
                                                  flag             */
         genevt2(2501,XXXX,3,(HOUSEA + UNIT1)); /* turn X10 #1 off */
      end
      if (((eventcode == 1233) & (param1 == 1)) & (temp1 == 1))
                                                 /* if time schedule
                                                 active and vacation
                                                 mode flag on turn on
                                                 X10 #1 */
   begin
      genevt2(2500,XXXX,3, (HOUSEA + UNIT1));
   end
      if (((eventcode == 1234) & (param1 == 1)) | (temp1 == 0))
                                                 /* if either the time
                                                 schedule is not valid
                                                 or vacation mode is
                                                 turned off, turn off
                                                 X10 #1         */
   begin
      genevt2(2501,XXXX,3,(HOUSEA + UNIT1));
   end
end                                             /* end of script */
```

4.4 Turn X10 Module On and Off

Note: X10 modules are not UL listed.

This script turns an X10 device on and off per a time schedule. This sample script assumes the following:

HOUSEA is defined as the house number of the X10 device

UNIT1 is defined as location of the X10 device

Time schedule that makes the device become active is defined as number 10

```
main()
   begin                                          /* start of script    */
      if ((eventcode == 1233) & (param1 == 10)) /* time schedule 10 is
                                                    active             */
         begin
            genevt2(2500,XXXX,3, (HOUSEA + UNIT1)); /* command to turn
                                                       device on       */
         end
      if ((eventcode == 1234) & (param1 == 10))
         begin
            genevt2(2501,XXXX,3, (HOUSEA + UNIT1));/* command to turn
                                                      device off      */
         end
   end                                             /* end of script */
```

4.3 Arm/Disarm by Area

To use the example script below, you would need to have a 4880 Status Output module attached to the system. The script turns 4880 outputs on as individual areas are armed and disarmed (for example, 4880, Output 1 corresponds to Area 1 being armed).

```
main ()
begin                                        /* beginning of script */
   if (eventcode == 1032)                    /* if area arm event
                                                    occurs*/
      begin                                  /* beginning of
                                                    statement(s) */
         genevt2 (2550,XXXX,34, param1);     /* turn on the appropriate
                                                 output when areas arm */
   end                                          /* end of statement */
   if (eventcode == 1038)                 /* if area disarm event occurs */
      begin                                 /* beginning of statement(s) */
         genevt2 (2551,XXXX,34, param1);/* turn off the appropriate
                                             output when areas disarm */
      end                                    /* end of statement */
end                                             /* end of script */
```

4.2 Separate Sounding Device Outputs Between Partitions

In the sample script below, a burglary alarm condition in Partition 2 outputs on P1-8 which activates a sounding device attached to Output 20. The sounding device is attached to a Model 4880 Status Output Module.

This sample script assumes the following that Device 3 is the device ID for the 4880.

```
main ()
begin     /* beginning of script       */
   if ((eventcode == 24) & (partition == 2))/* 24 is code for burg
                                              alarm */
      begin
         genevt2(2550,XXXX,3,20);      /* Turn on Output 20 on
                                          Device 3 */
      end
   if ((eventcode == 224) & (partition == 2))    /* 224 is code for burg
                                                  alarm restored*/
      begin
         genevt2(2551,XXXX,3,20);                /* Turn off Output 20
                                                       on Device 3 */
      end
end                                               /* end of script   */
```

Section 4
Example Programs

This section of the manual contains annotated example programs. You may be able to use some of the example programs exactly as they are. Others you can use as models for your own applications.

4.1 Controlling Bell Output Via the Auxiliary Relay

This application causes bell sounds to occur in alarm conditions. The bell will follow the bell cadence option for fire and be on steady for any other type of alarm. Shutdown will occur, if programmed.

This application must be used in UL installations. When you create a new account using the System Template, this script will automatically be included in the account. See the *Model 5580 Download Software Installation and Operation Manual* (P/N 150925) for more information.

```
main()
/* This script causes the auxiliary relay to follow the 4884 bell output
so that an unsupervised bell can be connected to the panel.          */
begin                                    /* start of script         */
   if (eventcode == 1505)                   /* 4884 bell outputon   */
      begin
         genevt0(1510,XXXX);                  /* turn on aux relay */
      end
   if (eventcode == 1507)                   /* 4884 bell output off */
      begin
         genevt0(1511,XXXX);                  /* turn off aux relay*/
      end
end
```

Section 5
Event Tables

The following sections are tables of eventcodes and parameters used in I/O application statements. Eventcodes are numbers that represent a system event; parameters are optional pieces of additional information associated with the event.

- Column 1 is a description of the event.
- Column 2 is the eventcode.
- Column 3 lists parameters associated with the event.
- Column 4 tells if the event can receive and/or generate events. Events listed as “receive only” can be used in conditional statements but cannot be used with the genevt command to generate new events. Events listed as “receive and generate” can be used in conditional statements and with the genevt command to generate new events. Events listed as “generate only” are not used in conditional statements; they are used only with the genevt command to generate new events.

5.1 Accessing Area Bit Masks

Parameters that store information about area where an event occurred, require use of bit masking to determine the area number. The bit mask is a mathematical operation that can be used to calculate whether bits are on or off.

Table 5-1: Bit Mask Values

Area #	Value
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128

5.2 Zone Events

Most zone events require a zone type code. Zone type codes are listed in Table 5-3 (next page).

Table 5-2: Zone Events

Event Description	Eventcode	Parameters	Receive/Generate
Zone alarm	20 + zone type code	param1 = Zone number param4 = Area bit mask	Receive only
Zone trouble	70 + zone type code	param1 = Zone number param4 = Area bit mask	Receive only
Zone not ready	120 + zone type code	param1 = Zone number param4 - Area bit mask	Receive only
Zone ready	170 + zone type code	param1 = Zone number param4 - Area bit mask	Receive only
Zone alarm restored	220 + zone type code	param1 = Zone number param4 - Area bit mask	Receive only
Zone trouble restored	270 + zone type code	param1 = Zone number param4 - Area bit mask	Receive only
Zone cross alarm alert	320 + zone type code	param1 = Zone number param4 - Area bit mask	Receive only
Bypass zone	500	param1 = Zone number param2 = User ID	Receive and Generate
Zone bypassed	501 + zone type code	param1 = Zone number param2 = User ID param3 = Area bit mask	Receive only
Unbypass zone	551	param1 = Zone number param2 = User ID	Receive and Generate
Zone Unbypassed	552 + zone type code	param1 = Zone number param2 = User ID param3 = Area bit mask	Receive only

5.2.1 Zone Event Type Codes

Section 5.2 lists eventcodes for zone events. In some cases, a code indicating type of alarm is used with the eventcode. Table 5-3 lists these codes.

EXAMPLE:

Eventcode 24 (actually eventcode 20 + zone type code 4) means that a burglary alarm has occurred. This is because:

20 = Eventcode for a zone alarm (from Section 5.2).
4 = Zone type code for burglary (from Table 5-3).

Table 5-3: Zone Types and Their Code

Zone Type	Code
Fire	0
Holdup	1
Emergency	2
Panic	3
Burglary	4
Tamper	5
Gas	6
Undefined	7
Water	8
Heat	9
Cold	10
Sprinkler	11
Doorbell 1	12
Doorbell 2	13
Key Arming Input	14
[Reserved, do not use.]	15
Door Monitor	16
Fire Contact	17
Egress Input	18

5.3 Chime or Doorbell State Change Events

Parameter: param1 = Zone number

Event Description	Eventcode	Parameters	Receive/Generate
Chime zone not ready	653	param1 = Zone number	Receive only
Chime zone ready	654	param1 = Zone number	Receive only
Doorbell1 zone changed	655	param1 = Zone number	Receive only
Doorbell2 zone changed	656	param1 = Zone number	Receive only

5.4 Duress Event

Parameter: param1 = Zone number assigned to duress code

Event Description	Eventcode	Parameters	Receive/Generate
Duress trigger event has occurred	657	param1 = zone number assigned to duress code	Receive only

5.5 Arm Request Events

Table 5-4: Arm Request Events

Event Description	Eventcode	Parameters	Receive/Generate
Reset alarms in areas	1000	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Auto arm areas	1001	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Manual arm areas	1002	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Exception arm areas	1003	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Auto force arm areas	1004	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Manual force arm areas	1005	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Exception force arm areas	1006	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Auto disarm areas	1007	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Manual disarm areas	1008	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Exception disarm areas	1009	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Manual disarm areas and reset alarms	1011	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Exception disarm areas and reset alarms	1012	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Activate interior 1 in areas	1013	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Activate interior 2 in areas	1014	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Deactivate interior 1 in areas	1015	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Deactivate interior 2 in areas	1016	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate

5.6 Arm Acknowledge Events

These events are generated *after* the request has been completed.

Table 5-5: Arm Acknowledge Events

Event Description	Eventcode	Parameters	Receive/Generate
Alarms have been reset	1030	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area auto armed	1031	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area manually armed	1032	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area exception armed	1033	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area auto force armed	1034	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area manual force armed	1035	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area exception force armed	1036	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area auto disarmed	1037	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area manually disarmed	1038	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area exception disarmed	1039	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area auto disarmed and alarms reset	1040	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area manually disarmed and alarms reset	1041	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Area exception disarmed and alarms reset	1042	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Interior 1 activated	1043	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Interior 2 activated	1045	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Interior 1 deactivated	1047	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Interior 2 deactivated	1049	param1 = Area number (1-8) param2 = User ID (if applicable)	Receive only
Armed report acknowledged (This event occurs only if the “Bell After Closing Report Acked” option has been selected and the system successfully reports a closing event.)	1050	param1 = Area number (1-8) param2 = User ID (if applicable)	Receive only

5.7 Area Status Events

Table 5-6: Area Status Events

Event Description	Eventcode	Parameters	Receive/Generate
Make areas instant	1060	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Area made instant	1061	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Make areas delayed (not instant)	1062	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Area made delayed (not instant)	1063	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Activate chime in areas	1064	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Chime activated	1065	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Deactivate chime in areas	1066	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Chime deactivated	1067	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Clear alarm memory in areas	1068	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate
Alarm memory cleared	1069	param1 = Area bit mask param2 = User ID (if applicable)	Receive only
Silence areas	1070	param1 = Area bit mask param2 = User ID (if applicable)	Receive and Generate

5.8 Timer Events

Table 5-7: Timer Events

Event Description	Eventcode	Parameters	Receive/Generate
Entry delay timer expired	1080	param1 = Area bit mask	Receive only
Exit delay timer expired	1081	param1 = Area bit mask	Receive only
Swinger bypass timer expired	1083	No parameters	Receive only
Report delay timer expired	1084	param1 = Area bit mask	Receive only
Alarm restore timer expired	1085	param1 = Area bit mask	Receive only
Auto Arm Timer Expired Event	1086	param1 = Area bit mask param2 = Timer mode 0 = auto arm delay time request 1 = power-up arm timer request	Receive only
Entry and/or exit delay timer changed event	1093	param1 = Entry/exit delay timer changed BIT1 on = Entry delay timer changed BIT2 on = Exit delay timer changed	Receive only
Auto arm ten second tic	1094	No parameters	Receive only
Set auto arm delay time	1095	param1 = Area number in bits (See Section 2.4.2 for more information.) param2 = timer value (0-255 minutes) param3 = timer mode 0 = auto arm delay time requested 1 = power-up arm timer requested	Receive only

5.9 Late to Close/Open Events

Table 5-8: Late to Close/Open Events

Event Description	Eventcode	Parameters	Receive/Generate
Late to close	1091	param1 = Area number (1-8)	Receive only
Late to open	1092	param1 = Area number (1-8)	Receive only

5.10 System Events

System-wide events that are not related to a specific partition, zone, or area.

Table 5-9: System Events

Event Description	Eventcode	Parameters	Receive/ Generate
One second tic	1200	param1 = Hour*256 + minute	Receive only
System power up	1202	No parameters	Receive only
Expander trouble	1203	param1 = Device ID (low byte). See Note below.	Receive only
Expander trouble restore	1204	param1 = Device ID (low byte) See Note below.	Receive only
Note: For eventcodes 1203 and 1204, you need to access only the low byte of the word (high byte contains data that is not used in an application). The expression (param1 & 255) masks off the high byte. **EXAMPLE:** if ((eventcode == 1203) & ((param1 & 255) == 3)			
Event memory cleared	1205	No parameters	Receive only
AC trouble	1206	No parameters	Receive only
AC restoral	1207	No parameters	Receive only
Power supply trouble	1208	param1 = Power type 0 = smoke power 1 = fire bell (4884), currently not available 2 = auxiliary power	Receive only
Power supply restored	1209	param1 = Power type 0 = smoke power 1 = fire bell (4884), currently not available 2 = auxiliary power	Receive only
System battery restoral	1210	No parameters	Receive only
System battery trouble	1211	No parameters	Receive only
Log threshold	1212	No parameters	Receive only
Log overflow	1213	No parameters	Receive only
Local program begin	1214	No parameters	Receive only
Local program success	1215	No parameters	Receive only
Local program fail	1216	No parameters	Receive only
Remote program begin	1217	No parameters	Receive only
Remote program success	1218	No parameters	Receive only
Remote program fail	1219	No parameters	Receive only
Listen in ended	1220	No parameters	Receive only

Table 5-9: System Events

Event Description	Eventcode	Parameters	Receive/ Generate
Listen in begin	1221	No parameters	Receive only
Phone line trouble	1222	param1 = line# (1 or 2)	Receive only
Phone line restoral	1223	param1 = line# (1 or 2)	Receive only
Automatic test	1225	param1 = account priority level (1-255)	Receive and Generate
Manual test	1226	param1 = user ID	Receive and Generate
Test end	1227	No parameters	Receive only
Test start	1228	No parameters	Receive only
Service required	1232	No parameters	Receive and Generate
User code tamper	1280	No parameters	Receive only
Date changed	1281	param1 = User ID	Receive only
Time changed	1283	param1 = User ID	Receive only
Date tic	1400	param1 = Year*256+month Year = Current year-1900 Month = 1 -12	Receive only
User code changed	1284	param1 = User ID	Receive only
User code deleted	1285	param1 = User ID	Receive only
User code added	1286	param1 = User ID	Receive only
Printer out of paper trouble	1300	param1 = Device number	Receive only
Printer out of paper trouble restore	1301	param1 = Device number	Receive only
Printer offline trouble	1302	param1 = Device number	Receive only
Printer offline trouble restore	1303	param1 = Device number	Receive only

5.11 Time Schedule Events

Table 5-10: Time Schedule Events

Event Description	Eventcode	Parameters	Receive/Generate
Schedule start	1233	param1 = Number of time schedule that changed	Receive only
Schedule end	1234	param1 = Number of time schedule that changed	Receive only

5.12 Door Access Events

Table 5-11: Door Access Events

Event Description	Eventcode	Parameters	Receive/Generate
Door access inhibited	1260	param1 = Door number param2 = User ID (if applicable)	Receive only
Door access enabled	1261	param1 = Door number param2 = User ID (if applicable)	Receive only
Door held open	1263	param1 = Door number	Receive only
Door restoral	1264	param1 = Door number	Receive only
Door access granted	1265	param1 = Door number param2 = User ID (if applicable)	Receive only
Door access denied	1266	param1 = Door number param2 = User ID (if applicable)	Receive only

5.13 Audible Alert Events

These events indicate if audible alarm signals have been turned on or off.

Table 5-12: Audible Alert Events

Event Description	Eventcode	Parameters	Receive/Generate
Internal and external speaker sound turned on	1309	param1 = Eventcode from the alarm event, indicates the type of alarm that occurred.	Receive only
Internal and external speaker sound turned off	1310	param1 = Eventcode from the alarm event, indicates the type of alarm that occurred.	Receive only
Internal speaker alarm sound turned on	1311	param1 = Eventcode from the alarm event, indicates the type of alarm that occurred.	Receive only
Internal speaker alarm sound turned off	1312	param1 = Eventcode from the alarm event, indicates the type of alarm that occurred.	Receive only
Speaker trouble sound turned on	1313	No parameters	Receive only
Speaker trouble sound turned off	1314	No parameters	Receive only
Cross alarm alert tone turned on	1315	No parameters	Receive only
Cross alarm alert tone turned off	1316	No parameters	Receive only

5.14 Dialer Report Events

These events are generated by the dialer after it receives acknowledgment for reported events from the central station.

Table 5-13: Dialer Reports Events

Event Description	Eventcode	Parameters	Receive/Generate
Report acknowledged	1330	param1 = Eventcode that was reported.	Receive only
Report failed	1331	param1 = Eventcode that was reported.	Receive only
Dialer reset requested	1332	No parameters	Receive only
Dialer has been reset	1333	No parameters	Receive only
Date tic	1400	param1 = Year*256+month Year = Current year-1900 Month = 1 -12	Receive only
Account trouble, dialer failed to report to an account.	1410	param1 = Account priority level (1-255)	Receive only
Account trouble restore	1411	param1 = Account priority level (1-255)	Receive only

5.15 System Command Events

These events are used to control system resources. **No parameters associated with these events.**

Table 5-14: System Command Events

Description	Eventcode	Receive/Generate
Request to turn off smoke power	1500	Receive and Generate
Smoke power turned off (acknowledgment)	1501	Receive only
Request to turn on smoke power	1502	Receive and Generate
Smoke power turned on (acknowledgment)	1503	Receive only
4884 bell output on (acknowledgment)	1505	Receive only
4884 bell output off (acknowledgment)	1507	Receive only
Request to enable reporting	1508	Receive and Generate
Request to disable reporting	1509	Receive and Generate
Request to turn on the auxiliary relay	1510	Receive and Generate
Request to turn off the auxiliary relay	1511	Receive and Generate
Auxiliary relay turned on (acknowledgment)	1512	Receive only
Auxiliary relay turn off (acknowledgment)	1513	Receive only

5.16 X10 Control Events

Codes for controlling the X10 devices that are attached to the system.

Table 5-15: X10 Control Events

Description	Eventcode	Parameters	Receive/Generate
X 10 on	2500	param1 = Device ID number of 4880 where X10 module is located. param2 = House Code and Unit Number (See Note below.)	Generate only
X 10 off	2501	param1 = Device ID number of 4880 where X10 module is located. param2 = House Code and Unit Number (See Note below.)	Generate only
X 10 all lights on	2502	param1 = Device ID number of 4880 where X10 module is located.	Generate only
X 10 all lights off	2503	param1 = Device ID number of 4880 where X10 module is located.	Generate only
X 10 all off	2504	param1 = Device ID number of 4880 where X10 module is located.	Generate only
X 10 dim	2505	param1 = Device ID number of 4880 where X10 module is located. param2 = House Code and Unit Number (See Note below.)	Generate only
X 10 bright	2506	param1 = Device ID number of 4880 where X10 module is located. param2 = House Code and Unit Number (See Note below.)	Generate only

Note: param2 = House Code and Unit Number
(HOUSEA through HOUSEP and UNIT1 through UNIT16)

If you need to access both House Number and Unit Number, add them together; param2 will be the calculation, for example, (`HOUSEA + UNIT1`).

5.17 Output Control Events

Eventcodes for controlling system voltage outputs and relays on 4880 expanders and 4860C touchpad devices.

Table 5-16: Output Control Events

Event Description	Eventcode	Parameters	Receive/Generate
Turn output on	2550	param1 = Device ID number (required for all events) param2 = Output number, 1-20 for the 4880, not needed for the 4860C	Generate only
Turn output off	2551	param1 = Device ID number (required for all events) param2 = Output number, 1-20 for the 4880, not needed for the 4860C	Generate only
Turn all outputs off (This command turns off all outputs. For example, if the device is a 4880 and outputs 1-20 are all used, they will all turn off.)	2552	param1 = Device ID	Generate only
Turn all outputs on (This command turns on all outputs. For example, if the device is a 4880 and outputs 1-20 are all used, they will all turn on.)	2553	param1 = Device ID	Generate only
Turn more than one output on (This command turns on specified outputs from a group of outputs.)	2555	param1 = Device ID param2 = Location of the output See Section 2.4.3 for more information about this eventcode.	Generate only
Turn more than one output off (This command turns off specified outputs from a group of outputs.)	2556	param1 = Device ID param2 = Location of the output See Section 2.4.3 for more information about this eventcode.	Generate only

5.18 Touchpad Events

The following events indicate that user has pressed keys 0-9 on the touchpad while using the Aux Menu. See Section 2.5 for more information.

Table 5-17: Touchpad Events

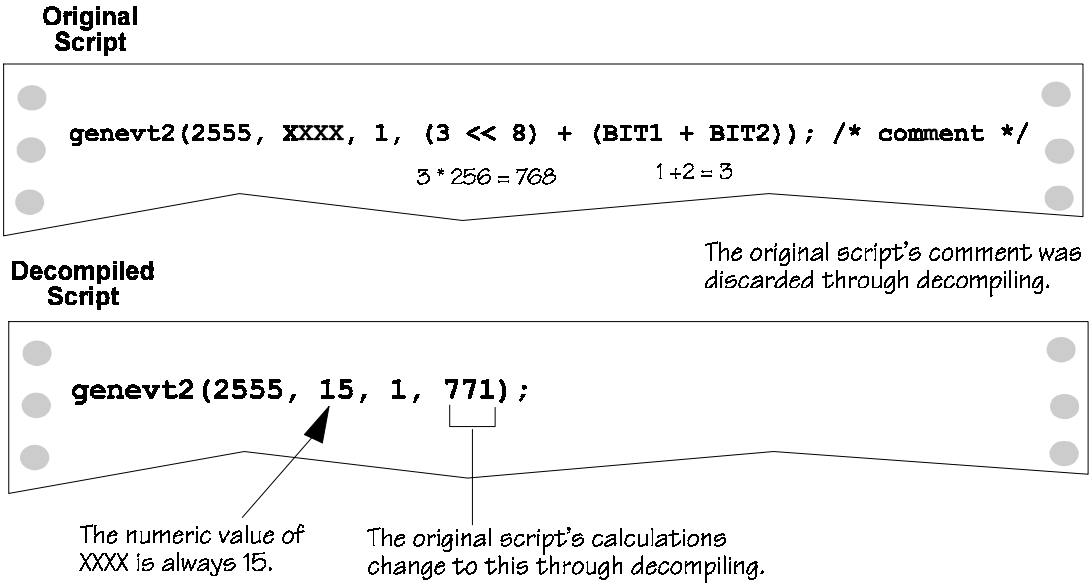
Description	Eventcode	Receive/Generate
User pressed touchpad key [0]	9000	Receive only
User pressed touchpad key [1]	9001	Receive only
User pressed touchpad key [2]	9002	Receive only
User pressed touchpad key [3]	9003	Receive only
User pressed touchpad key [4]	9004	Receive only
User pressed touchpad key [5]	9005	Receive only
User pressed touchpad key [6]	9006	Receive only
User pressed touchpad key [7]	9007	Receive only
User pressed touchpad key [8]	9008	Receive only
User pressed touchpad key [9]	9009	Receive only

Appendix A
Decompiled Data

When an account that contains an I/O script is uploaded, *decompiling* takes place. This means that name constants (numbers that have been assigned meaningful names, for example, BIT_, HOUSE_, and UNIT_) will be converted to numbers. Table A-1 on the next page lists the numeric values of constants that will be affected by decompiling.

Besides the conversion of some constants to numeric values, the decompiling process also discards any comments that were part of a script.

The diagram below shows "before and after" versions of a decompiled script.



IMPORTANT:

Silent Knight recommends that you always maintain current backups of all I/O scripts. If an account containing a script is uploaded, you can use the backup to replace the decompiled version. Decompiling *only* occurs to accounts that have been uploaded. If you never upload an account, you will never need to replace the original version of a script.

Decompiling does not affect operation of an I/O application. Replacing a decompiled script with its original version is never *necessary.* You simply may find it easier to use the script in its original form.

Table A-1: Numeric Values of named Constants

Variable Name	Numeric Value
BIT1	1
BIT2	2
BIT3	4
BIT4	8
BIT5	16
BIT6	32
BIT7	64
BIT8	128
HOUSEA	96
HOUSEB	224
HOUSEC	32
HOUSED	160
HOUSEE	16
HOUSEF	144
HOUSEG	80
HOUSEH	208
HOUSEI	112
HOUSEJ	240
HOUSEK	48
HOUSEL	176
HOUSEM	0
HOUSEN	128
HOUSEO	64
HOUSEP	192
XXXX	15

Variable Name	Numeric Value
BIT9	256
BIT10	512
BIT11	1024
BIT12	2048
BIT13	4096
BIT14	8192
BIT15	16384
BIT16	32768
UNIT1	6
UNIT2	14
UNIT3	2
UNIT4	10
UNIT5	1
UNIT6	9
UNIT7	5
UNIT8	13
UNIT9	7
UNIT10	15
UNIT11	3
UNIT12	11
UNIT13	0
UNIT14	8
UNIT15	4
UNIT16	12

Appendix B
Application Size

I/O applications are limited to 500 bytes. If you create an application that is too large, it will not compile. The error message “Application too large” will display.

When you are creating an application, compile frequently. If you receive the “Application too large” error message and the previous version compiled successfully, you will know that the new items you added are what caused the application to become too large.

If you have created a too-large application that you cannot shorten, contact Silent Knight Technical Support. Technical Support staff may be able to help you determine how to make your application more concise.

The following chart shows the numer of bytes each element of an I/O statement us

Table B-1: Bytes Per Statement Type

I/O Statement Element	Uses
genevt command (genevt0 - genevt3)	2 bytes
temp variables (temp1 - temp16)	2 bytes
constant (a number that does not change or a name that represents a number, such as BIT1)	3 bytes
parameter (param1 - param4)	2 bytes
All relational and mathematical operators	1 byte

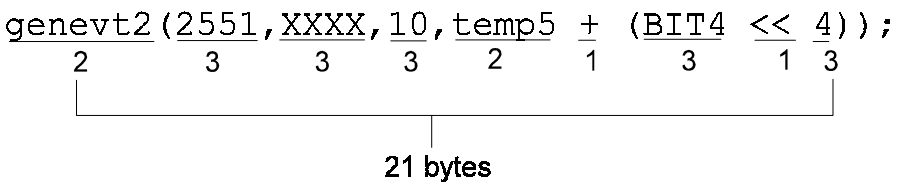


Figure B-1 Sample command statement size

Appendix C
Terms Used in This Manual

application	The programs you create for your customers are referred to as “applications” (or application programs) in this manual.
bit, byte	A byte is made up of 8 bits. A bit is a 0 or a 1. The pattern of bits determines what data is stored in the byte. I/O applications control a hardware device by sending it a byte of data. Each bit of the byte has a specific purpose. See Section 2.4.1 for more information.
case-sensitive	The 4820 I/O application programming language is case-sensitive. This means that if you use a capital letter when a lowercase is required (or vice versa), you will receive a syntax error. See Section 2.3.10 for more information.
command statement	Tells the system what to do. The I/O application programming language has two types of commands: assignment commands (temp1 = 1) and genevt commands (genevt0 (9999XXXX)).
comments	Any text surrounded by /* */ is a comment. A comment is an explanation of a statement. Words surrounded by comment markers have no effect on the application. See Section 2.3.9 for more information.
compile, compiler	When you compile your application (from the script editor screen), you are running a syntax check (that is, verifying that your script has no errors). The compiler is a part of the I/O application programming language.
conditional statement	A statement beginning with “if” or “if else” that tells the application what events to look for.
constant	A number that will not change. For example, in the statement *temp1 + 1*, “1” is a constant. “BIT1” is a *named* constant. It represents a number that will not change.
decompile	Occurs when an application is uploaded. Variables and named constants are returned to their numeric values. Comments are lost. See Appendix A for more information.
else	Tells the application what to do if a condition is not met.
eventcode	Number that represents an event that occurs on the system. See Section 5.2.
generate only	Eventcodes identified as “generate only” are used only with the genevt command. See Section 5.
genevt command	Tells the system to generate an event.
if	Opens a conditional statement.
logical operators	The operators OR, AND, XOR can be used in I/O applications. See Section 2.3.2 for more information.
parameter	An additional piece of data that is associated with an event. See Section 5.
receive and generate	Eventcodes identified as “receive and generate” can be used to generate events with the genevt command and in conditional statements. See Section 5.
receive only	Eventcodes identified as “receive and generate” can be used only in conditional statements. See Section 5.

script	The complete set of commands that runs an application.
script editor	The 5580 contains a simple word processor for creating applications. See Section 3.
syntax	The way in which statements are constructed in order to be understood by the compiler.
syntax errors	If you create a statement that is not understood by the compiler, you will receive a syntax error. See Section 3.5.1 for examples of syntax errors.
templates	When you create an application using the System Template, the UL required script for controlling bells will be the default. If your installation is not UL, you can delete the default script. See Section 3.1.1. You can also save an account as a template so that a script you have created will be available to other accounts. If you need more information about how to save an account as a template, refer to the *Model 5580 Upload-Download Installation and Operation Manual* (P/N 150925).
variable	Temporary storage location.
word	Two bytes