WEStation Graphics Language Reference Manual

Section **Title** **Page**

Summary of Changes

Section 1. Introduction

Section 2. Process Points and Conditionals

Section 3. Source Commands Reference

Table of Contents, Cont'd

Section **Title** **Page**

Section 3. Source Commands Reference (Cont'd)

Table of Contents, Cont'd

Section **Title** **Page**

Section 3. Source Commands Reference (Cont'd)

Section 4. Application Programs

Table of Contents, Cont'd

Section **Title** **Page**

Section 4. Application Programs (Cont'd)

Table of Contents, Cont'd

Table of Contents, Cont'd
List of Tables

Summary of Changes

This revision of "WEStation Graphics Language Reference Manual" (U0-8211) has been updated to include information related to software release **8.5** and later. The changes include the following:

Section 2. Process Points and Conditionals increases the maximum number of $G points allowed in a graphic to 250 (previously 99).

Section 3. Source Commands Reference describes three new commands:

DEF_FKEY_GROUP
FKEY_STATE
LOAD_FKEY_GROUP

This section also describes the following changes to existing commands:

- State (on, off) parameter added to the FUNC_KEY command.
- Fraction of a second update rates (0.1 - 0.9) added to the DIAGRAM command.
- Bitmap text support added to the OL_BUTTON, OL_CHOICE, and OL_EVENT_MENU commands.
- Support for $SET, $CONST, and $STATUS arguments added to the macro capability (see MACRO, POKE_FLD, FUNC_KEY, SETVAL, RUN_PROGRAMS, TEXT, MULTI_TEXT, set expression, and simple expression).
- Support for segments 213, 214 added to the Pointer command.
- The diag_num and prog_num arguments to the POKE_FLD (type 7 and 23) can now be specified by a $P pointer.
- National Language Support (NLS) rules for using multi-byte text in the TEXT, MULTI_TEXT, OL_EVENT_MENU, OL_BUTTON, and OL_CHOICE commands.

Section 1. Introduction

1-1. Overview

This document provides a reference to the graphics source language rules and commands. Each process diagram is created in source language form as well as binary object form. The object (.diag) file is used for display of the process diagram. The source code (.src) file can be used to create and edit the process diagram. In particular, “logic” graphics commands (such as if/endif), which do not have associated display items, are created using source code.

1-2. Contents of this Document

Section 1. Introduction describes the organization of the Graphics Language Reference Manual.

Section 2. Process Points and Conditionals provides information on point records, pointers, and conditional statements.

Section 3. Source Commands Reference lists the section labels and main commands used in the graphics language along with a description of each item. It also discusses the rules associated with the graphics language.

Section 4. Application Programs discusses the application programs that can be used in process diagrams.

Appendix A. Reserved Words lists the words that are reserved for use by the compiler.

Appendix B. Status Words lists the graphics language status words.

1-3. Additional Reference Documentation

Table 1-1 lists additional reference documentation which may be helpful while using this document.

Table 1-1. Reference Documentation

Document Number	Title	Description
U0-8115	Alarm Management System User's Guide	Describes the Alarm Management System (provides information on Alert Lists and alarm cues).
U0-0106	Control Algorithms	Describes the WDPF control algorithms.
M0-8005	Drop Installation Manual	Provides information on required hardware and software.
U0-8500	Historian WEStation User's Guide	Describes the Event Logger function.
U0-8100	Operator WEStation User's Guide	Describes the Group Builder function.
U0-0131	Record Types User's Guide	Describes the WDPF record types.
M0-0003	Self-Test Diagnostics	Provides information on the System Status Display diagrams.
U0-8205	System Point Directory and WEStation Database Compiler User's Guide	Provides information on the System Point Directory.
U0-8210	WEStation Graphics Builder User's Guide	Provides detailed instructions on using the Graphics Builder menus and windows.

In addition to the documents listed in Table 1-1 , this manual may refer to "applicable vendor documentation", which is defined on a project basis. These documents may include:

- Operating system (UNIX/Solaris) manuals.
- Window manager (OpenWindows) documentation.
- Workstation hardware (Sun) documentation.
- Peripheral device documentation.

Man pages (standard UNIX on-line documentation) are also available for operating system functions and certain WDPF functions.

Section 2. Process Points and Conditionals

2-1. Section Overview

This section describes the point record information used in graphics. It also provides an overview of the conditionals that are used in the graphics language. The following topics are included:

- Assumptions (Section 2-2).
- Process point record information (Section 2-3).
- Conditionals (Section 2-4).

2-2. Assumptions

This document assumes that the user will create and edit graphics language programs via the Graphics Builder's source editor (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on using the source editor). The user may, however, create programs using a text editor supplied with the operating system or the window manager. See the applicable vendor documentation for information on using these system text editors.

Note

It is recommended that the user be familiar with the source editor functions before creating and editing source code.

2-3. Process Point Record Information

Point record information is required in many contexts when programming the WDPF system. This section defines the valid point names and record fields that can be used in the graphics language.

Valid point names in the graphics language consist of the following:

- Dummy points (Section 2-3.1).
- $P, $G, $H, $W, and $D pointers (Section 2-3.2).
- Process point names defined in the System Point Directory (see "System Point Directory User's Guide" (U0-8205)).

The System Point Directory process points, dummy points, and "$" pointers can be used wherever point names (or point name/record field pairs) are used in graphics commands.

A backslash (\) character may be used to delimit any process point, dummy point, or "$" pointer name in graphics commands. The backslash character acts as a flag, indicating that a process point name is following. The process point name is terminated with a backslash also (both backslashes are required, for example, \A100\). The backslash character is not required for "$" pointers, standard process point names, or standard dummy point names. It is required for non-standard process point names and dummy point names.

The maximum number (N) of characters allowed in process point names is determined by the system point directory. The system point directory is specified by the WDPF_PDIR environment variable. If WDPF_PDIR references an 8-character point directory, point names may be 8 characters; if WDPF_PDIR references a 16-character extended tag point directory, point names may be 16 characters.

Standard process point names and dummy point names consist of N characters or less, begin with an alphabetic character, and are followed by alphanumeric characters or the '_' character. Non-standard process point names and dummy point names consist of N characters or less, may begin with a numeric character, may consist only of numeric characters, or may contain characters other than alphanumeric characters and '_'. Examples of such point names are: 1valve, 1234, and A1+A2. The backslash character is required for non-standard point names because certain command arguments (such as point low/high limits) may be integers, real numbers, or process points. The backslash is used to distinguish between the integer 1234 and the process point 1234.

Notes

1. The backslash characters are NOT included in the characters making up the point name. That is, if the point name is limited to 8 characters, the name itself may be 8 characters in addition to the backslashes.
2. There is no space between the backslashes and the point name characters.

2-3.1. Dummy Points

Dummy points are valid point names that do not exist in the System Point Directory (see "System Point Directory User's Guide" (U0-8205) for a definition of a valid point name). Non-standard dummy point names must be preceded and terminated by the backslash (\) character. The backslash is optional for standard dummy point names (see Section 2-3 for a definition of standard and non-standard point names).

Dummy points are used to enable a diagram to be compiled and displayed before the points are added to the System Point Directory. When dummy points are displayed in a diagram, they are inactive. Therefore, final testing of the diagram cannot take place until the dummy points are added to the System Point Directory, and the diagram is recompiled and redisplayed.

Dummy points have no point type associated with them. Since they have no type, some of the error checking that is performed for points in the System Point Directory cannot be performed for dummy points. Therefore, when dummy points are added to the System Point Directory and the diagram is recompiled, there may be errors that were not detected previously. These errors are caused by the point type assigned to the formerly "typeless" point. To minimize these errors, the user should note the point type that will be assigned to the dummy point, and write the source code as if the dummy point actually has this type.

When a file containing dummy points is compiled, the Graphics Builder displays a warning message saying that undefined points exist in the diagram. A list showing all the dummy points used in the diagram is then displayed. Checking this list is important because misspellings and invalid identifiers may be interpreted as dummy points by the graphics language.

For example, operands in a conditional statement may be any of the following types: integer, real constant, point name/record field, pointer/offset, or status identifier. An invalid status identifier (such a OFF16) would be handled as a dummy point named OFF16 (see Appendix B for more information on status words). Therefore, unintended dummy points in the list indicate errors in the diagram.

2-3.2. Pointers

The Graphics Language recognizes the following valid pointers: $P, $G, $H, $W, and $D. Pointers are used as a means of passing data into a diagram. The $W and $D pointers are used as parameters to the diagram. Actual process point names will be passed for these parameters. $G pointers reference points defined externally to the diagram in the system group library file. $P and $H pointers reference a segment offset in the computer memory in the Operator WEStation. Pointers are identified by a dollar sign ($).

The “$P” pointers (valid range = $P1 through $P99) point to locations in memory segments. These data areas are internal to the process diagram system and can be used by any graphic. Data may be passed between graphics using these data areas. A maximum of 99 $P pointers can be defined.

Whenever point names and record fields are used in graphics commands, $P pointers can be used to access data in memory. For example, an area of memory is reserved in the Operator WEStation to store data received from another drop (such as general message data). Using $P pointers allows the user to access and display this data by setting a pointer to begin at a specified segment and offset within the reserved area.

The “$G” pointers (valid range = $G1 through $G250) are used to reference the group points stored in the current point group. A given group can have a maximum of 250 points associated with it ($G1 represents the first point in the current group, $G2 represents the second point, etc.). The $G1 pointer is reserved for Westinghouse use. See “Operator WEStation User’s Guide” (U0-8100) for information on the Group Builder function.

The “$H” pointers (valid range = $H1 through $H99) point to a segment of memory that contains System IDs. This in turn points to the Data Highway record field. A maximum of 99 $H pointers can be defined.

The “$W” pointers (valid range = $W1 through $W99) are used in WINDOW diagrams. The $W pointers mark generic points that should be replaced with valid process points when the diagram is displayed. The POKE_FLD (type 8) graphics command takes a diagram number and a list of up to 99 process points as parameters. When a diagram is displayed via a POKE_FLD (type 8) command, the first process point parameter is substituted in the diagram for all occurrences of $W1, the second parameter is substituted for all occurrences of $W2, and so on up to $W99. See Section 3 for more information on the POKE_FLD command.

The “$D” pointers (valid range = $D1 through $D99) are used in macro diagrams. The MACRO graphics command takes a macro diagram number, and a list of up to 99 process points as parameters. Inside the macro code, The $D pointers mark generic points that should be replaced with valid process points when the macro is added to a graphic. $D pointers are required because a single macro diagram may be added to several graphics with different process point parameters. When the macro is added to the graphic, each process point parameter is substituted in the macro gcodes for all occurrences of the corresponding $D pointer. That is, the first process point is used for $D1, the second process point is used for $D2, and so on.

In addition, the following rules apply to pointers:

- The numbers chosen for the *x* in $P*x* and $H*x* have no relationship to segments used (that is, the numbers are identifiers chosen by the user).
- The group pointer number ($G*x*) indicates the point within a group to be used (that is, the *x*, represents a number between 1 and 250, which indicates the group point number).
- Only the $P and the $H pointers must be initialized with a POINTER command (see Section 3 for more information on the POINTER command).

2-3.3. Record Fields

Process point records consist of predefined record fields. These record fields vary according to the type of the process point (for example, analog, digital, device, packed group, and packed digital), but follow the same naming convention (they are named by a two-character ASCII string). For example, the **Analog Value** record field for an analog process point is named **AV**. See "Record Types User's Guide" (U0-0131) for more information on record fields.

A default record field is defined for each point record type. If the user does not enter a record field in a graphics command when a process point/record field pair is required, the default record field for that type of point is assumed. Note that AV is the default record field for dummy points and $D, $W, $H, and $G pointers. The default for $P pointers is $B0 (see explanation below). See "Record Types User's Guide" (U0-0131) for more information on default record fields.

Record fields specified for "$H", "$G", "$D", and "$W" pointers follow the same format as System Point Directory points. Since "$P" pointers do not reference process points, these pointers use a special offset notation: [1]

- $I*n* — represents a 32-bit integer beginning at an offset of n bytes; n must be on a 32-bit boundary (for example, $I0,$I4,$I8,...).
- $R*n* — represents a 32-bit floating real record field beginning at an offset of n bytes; n must be on a 32-bit boundary (for example, $R0,$R4,$R8,...).
- $B*n* — represents a byte at an offset of n bytes (for example, $B0,$B1,$B2,...).
- $S*n* — represents a 16-bit integer beginning at an offset of n bytes; n must be on a 16-bit boundary (for example, $S0,$S2,$S4,...).
- $A*n*X*l* — represents an ASCII string of length *l* beginning at an offset of *n* bytes.

  where:

  n = (0 through 32,767) and *l* = (0 through 255).

1. All mathematical values follow SUN standards (see applicable vendor documentation).

2-4. Conditionals

Conditional statements allow the value of a graphics command parameter to change based upon a specified condition. Five types of conditional statements are supported in the Graphics Language:

- Simple Expressions.
- Compound Expressions.
- Case Expressions.
- Quality Expressions.
- Set Expressions.

Conditional statements may be used in conjunction with various commands. The type of the conditional value is determined by the conditional parameter for the conditional (for example: the conditional parameter for the COLOR command is the color name; thus the conditional value could be red, blue, green, etc.).

2-4.1. Conditional Parameters

The list below shows the parameter(s) which may be conditional for each command.

Command	Conditional Parameter
ARC	Line pattern and/or fill pattern name.
CIRCLE	Line pattern and/or fill pattern name.
COLOR	Color name parameter.
BLINK	Blink state.
DYNAMIC_LINE	Line pattern.
ELLIPSE	Line pattern and/or fill pattern name.
LINE	Line pattern.
MULTI_TEXT	List of strings (same number as entered for default case)
POLYGON	Line pattern and/or fill pattern name.
PROCESS_PT	Record field.

RECTANGLE	Line pattern and/or fill pattern name.
SHAPE	Shape name.
TEXT	Text string.
TRIG_ON	Trigger number.

Note

The “unfilled” fill pattern cannot be used as a conditional value in any conditional. This is because “unfilled” is not an actual fill pattern, but rather, the lack of a fill pattern. If an item is created with the unfilled option, only the outline of the item is drawn. If the unfilled option was allowed in conditionals when that conditional became true, only the outline of the item would be drawn and any existing fill would not be erased.

2-4.2. Conditional Reference Pages

Each conditional is described on separate reference sheets in this sections.

In the example syntax specifications listed on the following pages, the conditional expression is always optional.

Description

Simple Expression conditionals consist of one or more expressions joined by logical operators. The conditional evaluates to true or false. There is a single conditional value to substitute for the default parameter in the graphics command when the conditional evaluates to true.

Syntax

```
command default_value simple_expression conditional_value
```

where:

command	=	Graphics language command (see Section 2-4.1 for a list of commands that may use conditional statements).
default_value	=	Value which is used if the simple_expression evaluates as False (must be of appropriate type for applicable command parameter).
simple_expression	=	Simple expression. See syntax and explanation below.
conditional_value	=	Value which is used if simple_expression evaluates as True (must be of appropriate type for applicable command parameter).

SIMPLE EXPRESSION
Cont'd

The syntax for the simple_expression is as follows:

```
(operand1 rel_op operand2) [ logic_op (operand1 rel_op
operand2) ... ]
```

where:

operand1, operand2 = Any of the following:

- Integer or real constant.
- Point name (default record field assumed).
- Point name and record field.
- Pointer variable (default record field or offset assumed).
- Pointer variable and record field/offset
- Record Type Name (AI, AL, DM, DC, GP, VC, etc.)
- Status identifier (HDWRFAIL, OFF0, etc.)
- Set variable (SET1, SET12, etc.)
- $SET variable ($SET1,$SET2, etc.)
- $STATUS variable ($STATUS1,$STATUS2,etc.)
- $CONSTvariable ($CONST1,$CONST2,etc.)

rel_op = Relational operator. Any of the following:
<, <>, >, <=, >=, =

logic_op = Logical operator (AND, OR, EOR)
(EOR = exclusive OR; that is, one and only one of the expressions is true)

Operand1, rel_op, and operand2 must be placed within parenthesis. If more than two simple expressions are joined with a logical operator, the entire expression must be parenthesized as shown below:

```
((operand1 rel_op operand2) logic_op (operand1 rel_op
operand2))
```

Also, parenthesis can be used to control the evaluation order of a conditional. If parenthesis are not specified, the conditional is evaluated left to right.

Note

Brackets ([ ]) indicate optional items. The syntax enclosed in brackets can be repeated any number of times in the conditional. The ellipsis (...) indicates a sequence of additional items of the same type.

Rules

1. If one operand in an expression is an integer, the other operand must be a point name/record field pair or a set variable. For example:

   A100 BB = 10
   SET2 = 20

2. If one operand in an expression is a real number, the other operand must be a point name or a point name and record field (if no record field is specified, the default record field for that point type is assumed). For example:

   A200 LL > 100.45

3. Both operands may be point names and/or point names and record fields (if no record field is specified, the default record field for that point type is assumed). For example:

   A100 <= A100 LL

4. If one operand is a status identifier or a record type name (AI, VC, BG, DI, etc.), the other operand must be a point name and record field, where the record field is of type integer. If no record field is specified, AS or DS is assumed for all point types except packed digital. For packed digital, the AV record field is assumed for all PSET and PRESET status identifiers; for all other status identifiers, the AS record field is assumed. For example:

A100 AS = ALARM
D200 DS = RESET
A100 RT = AI
D200 RT = DI

5. If one operand is a set variable, the other operand must be an integer or a point name and integer or a byte record field. For example:

   Set5 > 50
   Set20 = A100 TB

6. The $SET, $CONST, and $STATUS operands are used to pass set variables, integer/real constants, and status words respectively into expressions in a macro file instead of hard-coding them. These operands are only valid in macro files. For example:

   (A100 AV > $CONST1)

   could be used to pass 123 (or 34.56 ,or some other number) for $CONST1 in the expression above in a macro file.

Examples

Example 1

```
BACKGROUND

COLOR FG cyan (A100 AV < 50) yellow

RECTANGLE 4543 4543 6591 8343 4 solid unfilled
```

In this example:

command	=	COLOR		
default_value	=	cyan		
simple_expression	=	See below:		
		operand1	=	A100 AV
		rel_op	=	<

operand2 = 50

conditional_value = yellow

Example 1 illustrates the most basic type of a simple expression conditional. If A100 AV < 50, the foreground color for the rectangle will be yellow. Otherwise, the foreground color for the rectangle will be cyan.

See Section 3 for more information on the COLOR command.

Example 2

```
BACKGROUND

COLOR FG red ((A100 AV > 50) AND (A100 AS = HDWRFAIL) OR
(D200 = ALARM)) blue

RECTANGLE 4543 4543 6591 8343 4 solid unfilled
```

In this example:

command	=	COLOR		
default_value	=	red		
simple_expression	=	See below:		
		first operand1	=	A100 AV
		first rel_op	=	>
		first operand2	=	50
		first logical_op	=	AND
		second operand1	=	A100 AS
		second rel_op	=	=
		second operand2	=	HDWRFAIL
		second logical_op	=	OR
		third operand1	=	D200
		third rel_op	=	=
		third operand2	=	ALARM
conditional_value	=	blue		

Example 2 joins three expressions with logical operators. Even though this conditional is more complex than the first example, it still evaluates to either true or false. If A100 AV > 50 and A100 AS = HDWRFAIL or D200 = ALARM, the foreground color for the rectangle will be blue. Otherwise, the foreground color for the rectangle will be red.

See Section 3 for more information on the COLOR command.

Description

Compound Expression conditionals consist of two or more simple expression conditionals. Each simple expression evaluates to true or false. For each simple expression there is an associated conditional value. The simple expressions are evaluated in the order in which they are entered, and the conditional value for the first expression evaluating to true determines the value for the conditional parameter. Once a condition is met, any following conditionals are ignored.

Syntax

```
command default_value { simple_expression1 conditional_value1
simple_expression2 conditional_value2 ... simple_expressionN
conditional_valueN }
```

where:

command	=	Graphics language command (see Section 2-4.1 for a list of commands that may use conditional statements).
default_value	=	Value which is used if simple_expression evaluates as False (must be of appropriate type for applicable command parameter).
simple _expression1	=	A simple expression conditional. See simple expression reference pages for more information.
conditional_value1	=	Value which is used if simple_expression evaluates as True (must be of appropriate type for applicable command parameter).
simple_expression2	=	Another simple expression conditional.
conditional_value2	=	Another conditional value.
{ }	=	Left and right braces must be used to enclose the entire compound conditional statement.

COMPOUND EXPRESSION
Cont'd

Rules

1. If one operand in an expression is an integer, the other operand must be a point name/record field pair or a set variable.

   A100 BB = 10
   A100 TB = 20

2. If one operand in an expression is a real number, the other operand must be a point name or a point name and record field (if no record field is specified, the default record field for that point type is assumed).

   A200 LL < 100.45

3. Both operands may be point names and/or point names and record fields (if no record field is specified, the default record field for the given point type is assumed).

   A100 <= A100 LL

4. If one operand is a status identifier, the other operand must be a point name and record field, where the record field is of type integer. If no record field is specified, AS or DS is assumed for all point types except packed digital. For packed digital, the AV record field is assumed for all PSET and PRESET status identifiers; for all other status identifiers, the AS record field is assumed.

   A100 AS = ALARM
   D200 DS = RESET

5. If one operand is a set variable, the other operand must be an integer or a point name and integer or byte record field.

   Set5 > 50
   Set20 = A100 TB

6. Use of parentheses around concatenated expressions (expressions joined with logical operators) is optional. If parentheses are not used, the order of precedence of the logical operators is left to right.

Example

```
BACKGROUND

COLOR FG magenta {(A100 AV > 50) blue (A100 AS = HDWRFAIL)
red} BG white

CIRCLE 7671 6349 2145 3 solid unfilled
```

In this example:

command	=	COLOR
default_parameter	=	magenta
simple_expression1	=	(A100 AV > 50)
conditional_value1	=	blue
simple_expression2	=	(A100 AS = HDWRFAIL)
conditional_value2	=	red

In this example, if A100 AV > 50, the circle will be blue. If the first simple expression conditional evaluates as false, the second simple expression will be evaluated. If A100 AS = HDWRFAIL, the circle will be red. If both expressions evaluate as false, the circle will be magenta.

CASE EXPRESSION

Description

Case Expression conditionals are used as a shorthand method of implementing a compound expression in which each of the simple expressions compares a given process point and record field pair to an integer. The integers in the simple expressions are sequential, though not necessarily consecutive. The case conditional is equivalent to the following compound conditional:

{(pt_name rec_fld = value1) conditional_value1

(pt_name rec_fld = value2) conditional_value2

.
.
.

(pt_name rec_fld = valueN) conditional_valueN}

Syntax

```
command default_value (CASE) pt_name rec_fld int_val
increment count conditional_values
```

where:

command	=	Graphics language command (see Section 2-4.1 for a list of commands that use conditional statements).
default_value	=	Value which is used if simple_expression evaluates as False (must be of appropriate type for applicable command parameter).
(CASE)	=	Keyword enclosed in parentheses.
pt_name	=	Point name or pointer variable.
rec_fld	=	Record field or pointer variable record field.
int_val	=	Integer value for default conditional value.
increment	=	Value between successive integer values.

count	=	Total number of conditional values specified.
conditional_value	=	Parameter to use in graphics command if the specified conditional evaluates to true (must be of appropriate type for applicable command parameter).

Example

```
TRIG_ON 1 (CASE) $P5 $I0 1 2 3 2 3 4
```

In this example:

command	=	TRIG_ON
default_value	=	1
required_keyword	=	(CASE)
pt_name	=	$P5
rec_fld	=	$I0
int_val	=	1
increment	=	2
count	=	3
conditional_values	=	2, 3, 4

In this example, a different trigger will execute based on the value for $P5 $I0 as described below:

- If $P5 $I0 < 3, trigger 1 executes.

  With the initial value set at 1 and the increment value set at 2, the first conditional value (2) is set at 3 (1 + 2 = 3). However, if $P5 $I0 is less than 3, trigger 1 (the default_value) will execute.

CASE EXPRESSION
Cont'd

- If $P5 $I0 >= 3 and if $P5 $I0 < 5, trigger 2 executes.

  With the initial value set at 1 and the increment value set at 2, the first conditional value (2) is set at 3 (1 + 2 = 3). The second conditional value (3) is set at 5 (1 +2 + 2 = 5). Therefore, $P5 $I0 must be greater than or equal to 3 and less than 5 in order for trigger 2 to execute.

- If $P5 $I0 >= 5 and if $P5 $I0 < 7, trigger 3 will execute.

  With the initial value set at 1 and the increment value set at 2, the second conditional value (3) is set at 5 (1 + 2 + 2 = 5). The third conditional value (4) is set at 7 (1 + 2 + 2 +2 = 7). Therefore, $P5 $I0 must be greater than or equal to 5 and less than 7 in order for trigger 3 to execute.

- If $P5 $I0 >= 7 and if $P5 $I0 < 9, trigger 4 executes.

  With the initial value set at 1 and the increment value set at 2, the third conditional value (4) is set at 7 (1 + 2+ 2 + 2 = 7). Since a fourth conditional value is not specified (count = 3), the user must add 2 (increment value) to 7 to get the upper boundary for the third conditional value. Therefore, $P5 $I0 must be greater than or equal to 7 and less than 9 in order for trigger 4 to execute.

- If $P5 $I0 >= 9, trigger 1 executes.

  If $P5 $I0 is greater than or equal to 9, trigger 1 (the default_value) executes.

Description

Quality Expression conditionals allow the value of a conditional parameter to change based upon the quality of a process point variable. This type of conditional is a shorthand method of implementing a compound conditional where a given process point is compared with the status identifiers for the fair, poor, bad, and timed-out qualities. The default parameter is displayed when the quality is good. The quality conditional is equivalent to the following compound conditional:

{(pt_name = fair) fair_cond_value (pt_name = poor) poor_cond_value

(pt_name = bad) bad_cond_value (pt_name = hdwrfail) timed_out_cond_value}

QUALITY EXPRESSION
Cont'd

Syntax

```
command good_cond_value (QUALITY) pt_name
fair_cond_value poor_cond_value bad_cond_value
timed_out_cond_value
```

where:

command	=	Graphics language command (see Section 2-4.1 for a list of commands that use conditional statements).
good_cond_value	=	Conditional value which is used if point has good quality. Equivalent to default_value in simple expression.
(QUALITY)	=	Keyword enclosed in parentheses.
pt_name	=	Point name or pointer variable.
fair_cond_value	=	Conditional value which is used if point has fair quality.
poor_cond_value	=	Conditional value which is used if point has poor quality.
bad_cond_value	=	Conditional value which is used if point has bad quality.
timed_out_cond_value	=	Conditional value which is used if point has timed-out quality.

Rules

1. The point specified must be an analog, digital, or device point.
2. The user must specify all four conditional values (fair quality, poor quality, bad quality, and timed-out quality).

Example

```
BACKGROUND

COLOR FG cyan (QUALITY) D200 green yellow red blue

RECTANGLE 4543 4543 6591 8343 3 solid unfilled
```

In this example:

command	=	COLOR
good_cond_value	=	cyan
required_keyword	=	(QUALITY)
pt_name	=	D200
fair_cond_value	=	green
poor_cond_value	=	yellow
bad_cond_value	=	red
timed_out_cond_value	=	blue

In this example:

- If D200 has good quality, the foreground color for the rectangle will be cyan.
- If D200 has fair quality, the foreground color for the rectangle will be green.
- If D200 has poor quality, the foreground color for the rectangle will be yellow.
- If D200 has bad quality, the foreground color for the rectangle will be red.
- If D200 has timed-out quality, the foreground color for the rectangle will be blue.

See Section 3 for more information on the COLOR and RECTANGLE commands.

SET EXPRESSION

Description

Set Expression conditionals allow the value of a parameter to change based upon the value of a set variable. The value of the set can be defined by graphics application programs or by the SETVAL command. Note that sets are global to the current main, subscreen, and window diagrams. Set1 in the main screen is the same as set1 in the subscreen, which is the same as set1 in the window.

This type of conditional is a shorthand method of implementing a compound conditional where a given set is compared with consecutive integer values in each simple expression. The default parameter is used if the value of the set variable is one or is greater than the number of conditional values plus one. The SET conditional is equivalent to the following compound conditional:

{(setX = 2) conditional_value1

(setX = 3) conditional_value2

(setX = 4) conditional_value3

.
.

(setX = N) conditional_valueN-1}

Syntax

```
command default_value (SETx) N conditional_values
```

where:

command	=	Graphics language command (see Section 2-4.1 for a list of commands that use conditional statements).
default_value	=	Value selected when SET x = 1 or does not match any other condition in the list (equivalent to default_value in simple expression).
(SETx)	=	Set variable enclosed in parenthesis, OR $SET variable enclosed in parenthesis.
N	=	Number of conditional values listed (N>0).
conditional_values	=	Parameter to use in graphics command if the specified conditional evaluates to true (must be of appropriate type for applicable command parameter).

Rules

1. The set variable must be used as an operand in a simple expression along with a relative operator and another operand.

   IF (SETx = 2)
   .
   .
   ENDIF

   The following example of usage of the SET conditional is not supported:

   IF (SETx)
   .
   .
   ENDIF

2. The $SET specification is used to pass the set into a macro. If the user wants to pass the set to be evaluated in the conditional into the macro instead of hard-coding the set, the user specifies "($SETn)" instead of "(SETn)".

Example

```
SETVAL 2 5

RECTANGLE 4543 4543 6591 8343 4 solid unfilled

LINE 3646 3965 9901 3965 4355 9705 10230 9705 3 dashed

COLOR FG red (SET2) 4 green blue black white
```

In this example:

command	=	COLOR
non_conditional_value	=	red
SETx	=	(SET2)
N	=	4
conditional value	=	green, blue, black, white

In this example, the SETVAL statement initializes the value of SET 2 to 5. The SET conditional in the COLOR statement specifies that the foreground color is white. This is determined as follows:

- If the value of SET2 = 1 or > 5, the foreground color is red (default value).
- If the value of SET2 = 2, the foreground color is green.
- If the value of SET2 = 3, the foreground color is blue.
- If the value of SET2 = 4, the foreground color is black.
- If the value of SET2 = 5, the foreground color is white.

See Section 3 for more information on the COLOR command.

Section 3. Source Commands Reference

3-1. Section Overview

This section provides a reference for the Graphics Language rules and commands. The following topics are included:

- Definitions of terms (Section 3-2).
- Graphics language rules (Section 3-3).
- Coordinates (Section 3-4).
- Sample file (Section 3-5).
- Graphics language commands (Section 3-6).

3-2. Definitions of Terms

This document assumes that the user is already familiar with certain operating system and window manager terms. Before beginning to use the Graphics Builder, it may be helpful to refer to the applicable operating system and window manger documentation.

This document uses the following terms:

Bitmap Text — Two font types are available for graphics: bitmap and vector text. Bitmap text does not scale on a zoom/resize operation and is not affected by the line width attribute. This font may be used to display multi-byte text (for example, Chinese).

Bitmap text is defined by the eight font sizes found in the **fonts.txt** file. The user may change the font sizes (valid range for font sizes is 2 pt. to 100 pt. text type) as long as eight sizes are defined in the file. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on this file.

Note

The font specified in the **.Xdefaults** file must be a fixed-width font. A proportional font does not function correctly in the Graphics Builder. The standard font style is **lucidasanstypewriter**. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information.

Bitmap_Over Text — Bitmap_Over text is the same as bitmap text but uses the overstrike option. This implies that both the foreground and background of each character cell making up the text string are drawn. If the overstrike option is not used, only the foreground of each character cell making up the text string is drawn on the canvas. This font may be used to display multi-byte text (for example, Chinese).

The overstrike option should always be used with text conditionals. A conditional may be written to have text change on the process diagram depending on the conditions in the plant. If the overstrike option is not used, the new text will not cover the old text (the lines for the old text will still be seen because only the foreground of the text is drawn). If the overstrike option is used, the new text will cover the old text since it has a background associated with it.

Vector Text — Vector text scales accordingly on zoom/resize operations and has an associated line width as specified in the line width array (see explanation below for more information on the line width array). This font CANNOT be used to display multi-byte text (for example, Chinese).

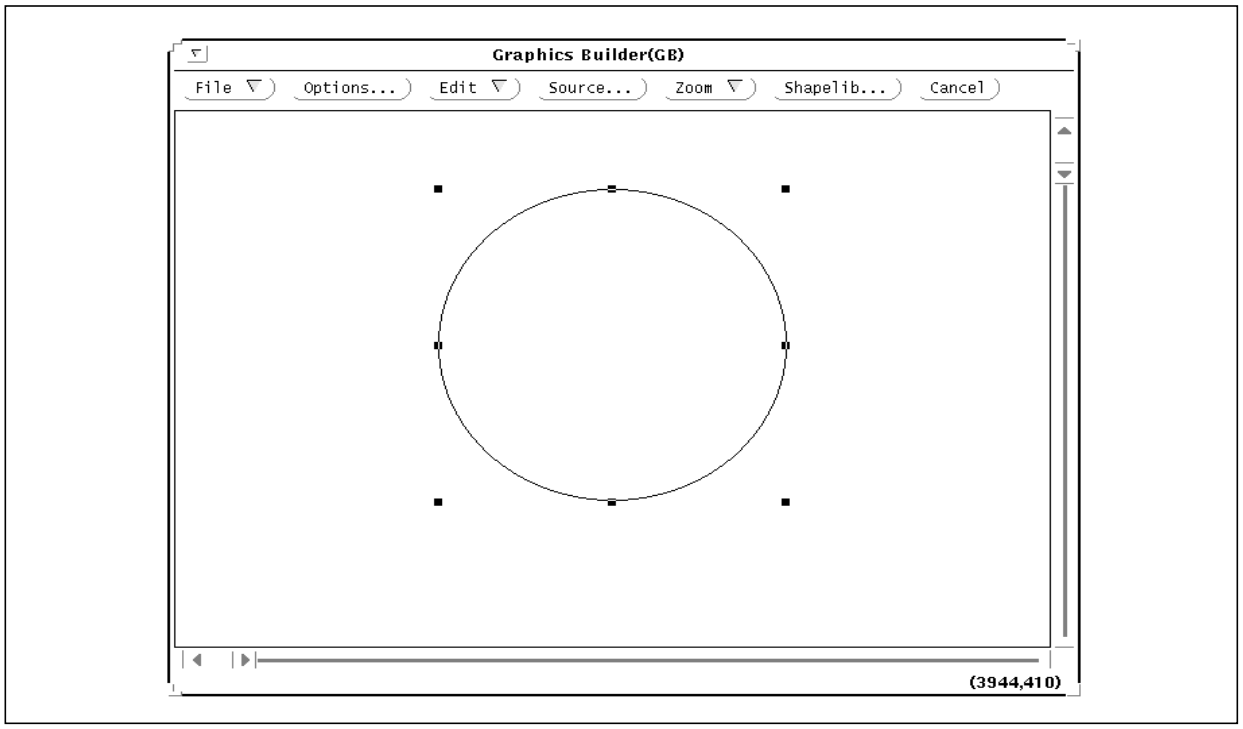
Users specify the font size for vector text by defining the pixel dimensions (width and height) of the character cells. The valid pixel range for characters is 5 through 16,384.

Vector_Over Text — Vector_Over text is the same as vector text but uses the overstrike option. This implies that both the foreground and background of each character cell making up the text string are drawn. If the overstrike option is not used, only the foreground of each character cell making up the text string is drawn on the canvas. This font CANNOT be used to display multi-byte text (for example, Chinese).

The overstrike option should always be used with text conditionals. A conditional may be written to have text change on the process diagram depending on the conditions in the plant. If the overstrike option is not used, the new text will not cover the old text (the lines for the old text will still be seen because only the foreground of the text is drawn). If the overstrike option is used, the new text will cover the old text since it has a background associated with it.

Zoom Extents Rectangle — The zoom extents rectangle defines the area of the canvas that will initially be displayed (and will be considered the “full view”) when the diagram appears on the Operator WEStation. Any items lying outside the zoom extents rectangle are clipped and cannot be seen by the user when the diagram displays on the Operator WEStation (the items are still in the diagram). The default zoom extents rectangle is the entire virtual canvas (0, 0 through 16383, 16383). The default zoom extents rectangle is set in the DIAGRAM command (a DIAGRAM command automatically appears in the source editor when the Graphics Builder initially starts; users can use the given settings or change any of the parameters). To set the zoom extents rectangle interactively, see “WEStation Graphics Builder User’s Guide” (U0-8210).

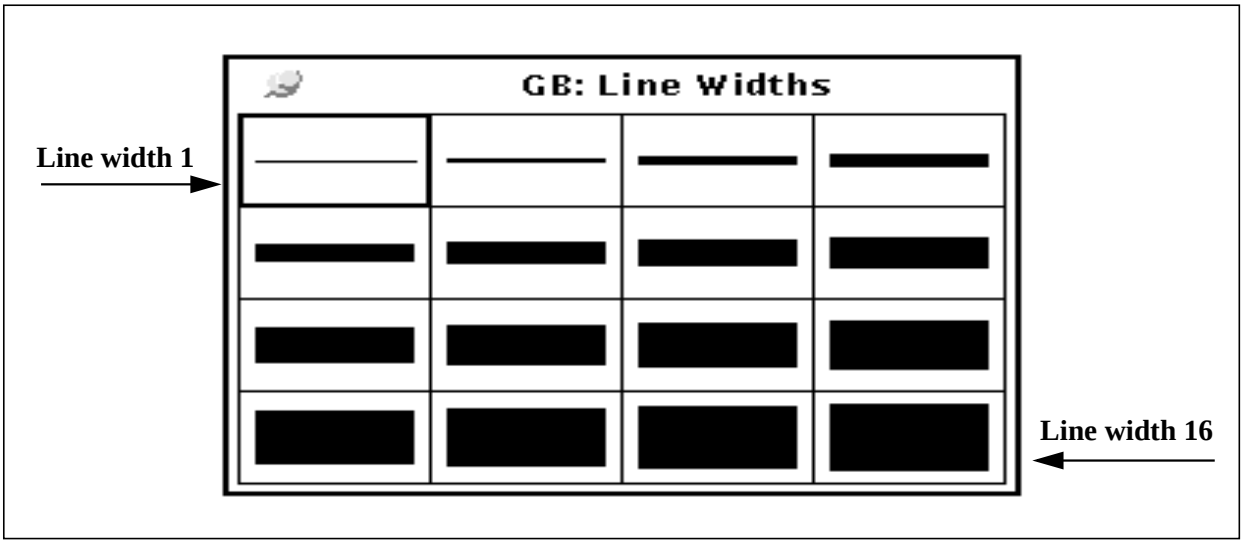
Outlining Rectangle — The outlining rectangle is the smallest rectangle that could be drawn around an item that includes every point on the item. An outlining rectangle surrounds every draw item. When an item is selected on the drawing canvas, eight handles appear around the item. The figure below shows how an outlining rectangle would appear on the Graphics Builder interactive editor window.



If a line were drawn to connect these handles, it would represent the outlining rectangle around the item.

Specifying Line Widths

The **GB:Line Widths** window on the Graphics Builder interactive editor window offers 16 line widths from which to choose (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on this window). The order of the line widths in this window corresponds to an array index that is used in the source editor to specify line widths. See figure below:



The first line width on the **GB:Line Widths** window (first row, first glyph) is assigned the number 1 in the line width index. The second line width (first row, second glyph) is assigned the number 2, and so on, reading the line widths from left to right. The last glyph on the **GB:Line Widths** window (fourth row, fourth glyph) is assigned the number 16 in the line width array index.

3-3. Graphics Language Rules

The following rules apply to the graphics language and source editor:

1. DIAGRAM must be the first command in the graphics language program (except for comment lines and blank lines). See reference page in Section 3-6 for a detailed description of the DIAGRAM command.

2. There must always be one DIAGRAM command per file. Changes to the DIAGRAM command cannot be undone through the undo feature on the source editor (see "WEStation Graphics Builder User's Guide" (U0-8210) for information on using the source editor).

3. Comments cannot be embedded within commands.

4. Blank lines may be placed anywhere in the source file (including within a command).

5. The user cannot modify a keyword that has already been entered into the source editor.

   For example, the user cannot select a CIRCLE command and change it into a LINE command. Once a valid keyword is entered, changes must be made by deleting and re-entering the entire command.

6. If the user wishes to move or copy a loop, the entire loop must be blocked first (see "WEStation Graphics Builder User's Guide" (U0-8210) for information on blocking commands).

7. It is recommended that the user choose a number as the diagram name. The valid diagram ranges are listed below:

Diagram Number	Diagram Type	Screen Type
1 - 699	System	Subwindow
700 - 989	Customer	Subwindow
990 - 999	System	Subwindow
1000 - 1999	System	Main
2000 - 3999	Customer	Main
4000 - 4999	System	Main
5000 - 6999	Customer	Main
7000 - 8499	Customer	Window
8500 - 8999	System	Window
1. Diagram 299 is a standard blank subwindow. 2. Diagram 4999 is a standard blank main window.		

8. Main commands can only be used with the section labels specified in Table 3-1 .

Table 3-1. Use of Main Commands with Section Labels

Main Command	Section Labels				
	BACKGROUND	**DIAGRAM**	**FOREGROUND**	**KEYBOARD**	**TRIGGER**
ARC	×		×		×
BAR	×		×		×
BLINK	×		×		×
CIRCLE	×		×		×
COLOR	×		×	×	×
CURSOR	×		×		×
DATE	×		×		×
DEF_FKEY_ GROUP		×			
DEF_QUAL		×			
DIAG_DISP	×		×		×
DOT	×		×		×
DYNAMIC_LINE	×		×		×
EF_STATE	×		×		×
ELLIPSE	×		×		×
ENTRY_FLD				×	
FKEY_STATE	×		×		×
FORCE_UPDATE			×		
FUNC_KEY				×	
GCODE	×		×		×
GTEXT	×		×		×
IF_CHANGED/ ENDIF	×		×		×
IF/ENDIF	×		×		×

Table 3-1. Use of Main Commands with Section Labels (Cont'd)

Main Command	Section Labels				
	BACKGROUND	DIAGRAM	FOREGROUND	KEYBOARD	TRIGGER
IFELSE/ELSE/ ENDIF	×		×		×
LINE	×		×		×
LOAD_FKEY_ GROUP	×		×		×
LOOP/ ENDLOOP	×		×		×
MACRO	×	×	×	×	×
MATH	×		×		×
MULTI_TEXT	×		×		×
OL_BUTTON				×	
OL_CHECKBOX				×	
OL_CHOICE				×	
OL_CYLINDER	×		×		×
OL_EVENT_ MENU				×	
OL_GAUGE	×		×		×
OL_SLIDER				×	
OL_RECTANGLE	×		×		×
PAGE				×	
PLOT	×		×		×
POINTER	×	×	×		×
POKE_FLD				×	
POKE_STATE	×		×		×
POLYGON	×		×		×
PROCESS_PT	×		×		×
PTR_EQUAL	×	×	×		×

Table 3-1. Use of Main Commands with Section Labels (Cont'd)

Main Command	Section Labels				
	BACKGROUND	DIAGRAM	FOREGROUND	KEYBOARD	TRIGGER
PTR_LOOP/ P_ENDLP	×	×	×		×
PTR_MOVE	×	×	×		×
PTR_VALUE	×	×	×		×
RECTANGLE	×		×		×
RUN_PROGRAMS	×		×		×
SETVAL	×		×		×
SHAPE	×		×		×
SW_ALARM_ COUNT	×		×		×
SW_ALARM_ CUE	×		×		×
SW_ALERT_ LIST	×		×		×
SW_PROCESS_ POINT	×		×		×
TEXT	×		×		×
TIME	×		×		×
TREND	×		×		×
TRIG_ON	×		×		×
XY_PLOT	×		×		×

Note

Macros can be used with all of the section labels (background, diagram, foreground, keyboard, and trigger). See the Macro reference page for more information on using macros.

9. The valid characters in the graphics language are:

Alphabetic	—	A through Z a through z
Numeric	—	0 through 9
Special:	—	See below:

. (period)	—	separates whole and fractional parts of real numbers (decimals)
+ (plus)	—	indicates a positive number or the mathematical symbol for addition.
- (minus)	—	indicates a negative number or the mathematical symbol for subtraction.
/ (slash)	—	used as the mathematical symbol for division.
* (asterisk)	—	denotes a comment or the mathematical symbol for multiplication.
‘ or “ (single or double quotes)	—	denotes the beginning and end of a text string.
() (parentheses)	—	indicates the beginning and end of a simple conditional expression.
{} (braces)	—	indicate the beginning and end of a compound conditional expression.
= (equal)	—	indicates a relational operation within a conditional expression.
< (less than)	—	denotes a relational operation within a conditional expression.
> (greater than)	—	denotes a relational operation within a conditional expression.
__ (underscore)	—	clarifies variable names.

10. Diagrams can be placed in one of three areas: MAIN, WINDOW, or SUBWIN (subwindow). The valid virtual coordinate range for each type of screen is 0 through 16383.

MAIN is the primary area for the diagram. Main screens require the user to enter the x, y (position) and w, h (width and height) of the diagram. (See the DIAGRAM command reference page for more information.) Main screens can be resized at the Operator WEStation.

WINDOWS can be called by poke fields to display menus, data entry fields, or other diagrams associated with the main diagram. Windows require the user to enter the x, y (position) and w, h (width and height) of the diagram. (See the DIAGRAM command reference page for more information.) Once the width and height are set, the operator cannot resize a Window at the Operator WEStation.

SUBWINDOWS are mainly used for the import of WDPF graphics created for 6- or 7-level systems. If subwindow is used, all of the information needed by the DIAGRAM command must be included; however, the numbers entered for x, y, w, h will be ignored. (See the DIAGRAM command reference page for more information.) The width and height (w and h) of subwindow diagrams are hard-coded with fixed values (700 pixels for width and 80 pixels for height). The initial position (x, y) of the subwindow will be determined by the position of the main screen (a subwindow will initially appear above the main window). If the operator moves the subwindow, it will appear in the new position thereafter. Subwindows cannot be resized.

11. Each source line of a graphics program can contain up to 132 characters. The graphics language is free-formatted, which means that statements do not have to begin in a certain column or end with a special character. Statements may use more than one line, but a carriage return must separate the lines if using a text editor other than the Graphics Builder source editor (see "WEStation Graphics Builder User's Guide" (U0-8210) for information on using the source editor).

12. Two types of constants can be used: **logical** and **arithmetic**. Logical constants represent two states: true or false. These constants are represented internally by a 1 (true) or a 0 (false). Arithmetic constants include integer and real constants. These constants are formed as follows:

    Integer Constants assume signed values and are represented in decimal notation. The decimal representation consists of a series of one to five decimal digits (0 through 9) written without a decimal point. The number may be preceded by a sign (plus or minus). If unsigned, the number is assumed to be positive. Any leading zeroes are ignored. Unless otherwise specified, the valid range for integers is 0 through 2,147,483,647.

    The following are valid integer constants:

    1 0 -56 0077 +85

Real Constants are represented in decimal notation. Decimal notation represents the number with a sign (optional), an integer, a decimal point, and a fractional extension. Both the integer and the fraction may contain a sequence of zero to eight decimal digits (0 through 9). Unless otherwise specified, the valid range for real constants is 0.00 through 3.40282346638528860e + 38 -1.

The following are valid real numbers:

3.1415927 0.005 6.3 8.00082

13. The user can specify a conditional statement in the graphics language to allow the value(s) of the parameter(s) to change based on a specified condition (for example, a conditional statement could cause a process point to be displayed in green instead of white if the point value is out of range). The user can build the following types of conditional statements:

    - Simple Conditional statement.
    - Compound Conditional statement.
    - Case Conditional statement.
    - Quality Conditional statement.
    - Set Conditional statement.

    See Section 2 for more information on conditional statements.

14. Valid point names in the graphics language consist of process point names defined in the System Point Directory and the pointer names defined in Section 2.

15. Valid record field names in the graphics language consist of standard record fields (see "Record Types User's Guide" (U0-0131) for more information) and the $ offset pointers used with the $P pointer type defined in Section 2.

16. Text strings may be delimited by single or double quotes. The maximum number of characters allowed in the string (not including the quotes) is 80 characters for TEXT, POKE_FLD, and MACRO strings, 50 characters for the OL_EVENT_MENU strings, and 30 characters for the MULTI_TEXT strings.

3-4. Coordinates

The graphics language uses two types of coordinates to specify x and y locations and widths and heights. An explanation of each follows:

Screen Coordinates — Screen coordinates are defined by the pixel resolution of the CRT monitor screen. The standard CRT monitor screen dimensions are 1152 x 900 pixels. Therefore, the screen coordinate range is 0 through 1151 and 0 through 899 where 0, 0 is the upper left corner of the screen, and 1151, 899 is the lower right corner of the screen (these dimensions apply to the standard CRT monitor; see applicable vendor documentation for information on custom screens). These coordinates are used in conjunction with the DIAGRAM and DIAG_DISP commands only (see the DIAGRAM and DIAG_DISP reference pages for more information on these commands). Screen coordinates are used to define the location/size of the diagram windows displayed on the Operator WEStation monitor screen. Screen coordinates are classic pixels.

Virtual Coordinates — Virtual coordinates define the drawing surface of a diagram. The drawing surface is defined to be 16384 x 16384 pixels. Therefore, the virtual coordinate range is 0 through 16383 and 0 through 16383 where 0, 0 is the upper left corner of the diagram, and 16383, 16383 is the lower right corner of the diagram. All of the graphics commands which draw items on the diagram (ARC, CIRCLE, BAR, LINE, etc.), specify x, y, w, h parameters in terms of virtual coordinates (see the applicable reference pages for more information on these commands).

Virtual coordinates are required to implement software pan and zoom. Since it is not possible to display a diagram that is 16384 x 16384 pixels on a CRT screen which is only 1152 x 900 pixels, virtual coordinates are mapped (scaled) to screen coordinates when items are displayed on the drawing surface. The current zoom level determines the number of virtual pixels which map to a single screen pixel. As the user zooms in further, the number of virtual pixels mapping to a single pixel decreases until one virtual pixel maps to one screen pixel.

Two types of virtual coordinates are supported:

- Absolute.
- Relative.

Absolute virtual coordinates specify a virtual offset from 0, 0 (upper left corner of the diagram). Relative virtual coordinates are used to specify a virtual offset from the previous display item (circle, bar, text, rectangle, etc.), or from 0, 0 if there is no previous item in the diagram section. Absolute virtual coordinates are specified as 0 through 16383. Relative virtual coordinates are specified as -16383 through 16383. Note that relative coordinates **must be enclosed in brackets**.

Refer to the examples below:

Example 1: Rectangle specified with absolute virtual coordinates:

```
RECTANGLE 4000 4000 6000 2000 1 solid solid
```

Example 2: Rectangle specified with relative virtual coordinates:

```
RECTANGLE [-1000] [3000] 6000 2000 1 solid solid
```

For this example, the source line in Example 2 immediately follows the source line in Example 1, as shown below:

```
BACKGROUND

COLOR FG RED

LINE 1000 2000 5000 5000 9000 2000 1 solid

RECTANGLE 4000 4000 6000 2000 1 solid solid

RECTANGLE [-1000] [3000] 6000 2000 1 solid solid
```

In the source above, the second rectangle would be relative to the first rectangle. The second rectangle would be 1000 pixels left of the first, and 3000 pixels below the first rectangle. If the user interactively moves the first rectangle, the position of the second automatically shifts also. If the user deletes the first rectangle, the second rectangle becomes relative to the line. If the user then deletes the line, the second rectangle becomes relative to 0, 0. If the user then adds a new item after the color command, the rectangle becomes relative to the new item.

For those command that have more than one x, y coordinate pair (LINE and POLYGON), only the first x, y coordinate pair is relative to the previous display item. If relative coordinates are specified for any other x, y pair (not the first pair), those relative coordinates are relative to the previous coordinate pair. See example below:

```
LINE 1000 2000 [5000] [5000] 9000 2000 1 solid
```

In the example above, the relative coordinates [5000] [5000] are relative to the previous coordinate pair (1000, 2000).

The following rules apply to relative coordinates:

1. Items can only be defined with relative coordinates through the source editor. The user cannot interactively draw items with relative coordinates. The user may interactively draw items and then change to relative coordinates by editing the command in the source editor. Once the item is created with relative coordinates, it may be edited (moved, copied, resized, rotated, inverted, and/or deleted) through the interactive editor.

2. Items specified with relative coordinates cannot be interactively grouped, and their place cannot be changed.

3. Relative coordinates can only be used to specify the x, y position of display items. Relative coordinates cannot be used to specify size (width and height). The user may specify x and y coordinates independently (x may be absolute and y may be relative). The LINE and POLYGON commands may specify any number of coordinates as relative. The x,y position of the overall DYNAMIC_LINE may be relative; the individual x,y pairs may not.

4. Relative coordinates may only be used to specify the position of the following commands: ARC, BAR, CIRCLE, DATE, DOT, DYNAMIC_LINE,ELLIPSE, GTEXT, LINE,MULTI_TEXT, OL_CYLINDER, OL_GAUGE, OL_RECTANGLE, PLOT, POLYGON, PROCESS PT, RECTANGLE, SHAPE, SW_ALARM_COUNT, SW_ALARM_CUE, SW_ALERT_LIST, SW_PROCESS_PT, TEXT, TIME, TREND, and XY_PLOT. See the examples below:

```
ARC [222] [3333] 1989 9999 123 -220 1 solid unfilled

BAR [2222] [3434] 1377 2401 up a100 av 2 22

CIRCLE [1222] [1111] 898 1 solid unfilled

DATE [999] [233] 1 vector 207 944 1

DOT [444] [555] large

DYNAMIC_LINE [1000] [4000] 2342 1078 0 100 0 100 NOT_FITTED
NOSCALE 1 2 \A100\ AV 50 \A200\ AV 50 1 solid

ELLIPSE [2323] [1212] 7776 7677 1 solid unfilled

GTEXT [233] [985] 1 7 horz vector 207 944 1

LINE [111] [2222] [3333] [3434] [3432] [1212] 1 solid

MULTI_TEXT [2000] [2000] 0 0 3 “string1” “string2”
“string3” VECTOR 140 257 1
```

```
OL_CYLINDER [0] [-1000] 550 5000 a200 0 100

OL_GAUGE [1000] [2000] 5000 550 left testai01 av testai01
ll testai01 hl

OL_RECTANGLE [-500] [500] 3000 1000 normal

PLOT [222] [3333] 1234 8765 up a100 av 2 33

POLYGON [2322] [6565] [4545] [5656] [3333] [2323] [454]
[233] 1 solid unfilled

PROCESS_PT [2323] [2222] 10 -1 right0 on horz vector 207
944 1 A100 AV

RECTANGLE [2222] [6778] 232 999 1 solid unfilled

SHAPE [222] [3333] 1260 3434 tmp 0 none

SW_ALARM_COUNT [2345] [5432] 5 HORZ VECTOR 413 2059 1
“LIST1”

SW_ALARM_CUE [7210] [5549] 1377 2401 TESTDIO1

SW_ALERT_LIST [6710] [5239] 689 3001 “LIST2”

SW_PROCESS_PT [8100] [6972] 10 -1 RIGHT0 HORZ VECTOR 413
2059 1 A100 AL D200 DL

TEXT [333] [232] “help” horz vector 207 944 1

TIME [333] [323] 1 vector 207 944 1

TREND [3456] [233] 2865 7654 horz a100 av 2 33 10 60 scale

XY_PLOT [2222] [3333] 1234 7654 a100 av 2 44 a200 av 3 44
scale
```

5. Items defined with relative coordinates are relative to the previous item. The point on the previous item to which an item is relative depends on the previous item. This point is defined as the origin. For example, the origin of a circle is the center. If a rectangle was specified to be relative to the circle, then the rectangle would be relative to the x, y point at the center of the circle. The origin for each draw item is listed in Table 3-2 .

Table 3-2. Origin by Draw Item

Draw Item	Relative Origin
Alarm Cue	Upper left corner.
Alert List	Upper left corner.
Arc	Center of the circle on which the arc lies.
Bar	Upper left corner (down or right or bias).
	Lower left corner (up).
	Upper right corner (left).
Circle	Center.
Date	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector text).
Dot	x, y position.
Dynamic_line	Lower left corner.
Ellipse	Center.
Entry Field	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector text).
Gtext	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector text).
Line	First vertex.

Table 3-2. Origin by Draw Item (Cont'd)

Multi_text	Baseline position of the first character of the first text string (for bitmap text).
	Upper left corner of the first character of the first text string (for vector text).
OL Button	Upper left corner of outlining rectangle (x, y from command syntax).
OL Checkbox	Upper left corner of outlining rectangle (x, y from command syntax).
OL Choice	Upper left corner of outlining rectangle (x, y from command syntax).
OL Cylinder	Upper left corner of outlining rectangle (x, y from command syntax).
OL Event Menu	Upper left corner of outlining rectangle (x, y from command syntax).
OL Gauge	Upper left corner of outlining rectangle (x, y from command syntax).
OL Rectangle	Upper left corner of outlining rectangle (x, y from command syntax).
OL Slider	Upper left corner of outlining rectangle (x, y from command syntax).
Plot	Low limit vertex which is different for each plot direction.
Poke Field	Upper left corner.
Polygon	First vertex.
Process Point	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector text).
Rectangle	Upper left corner.
Shape	Defined origin.

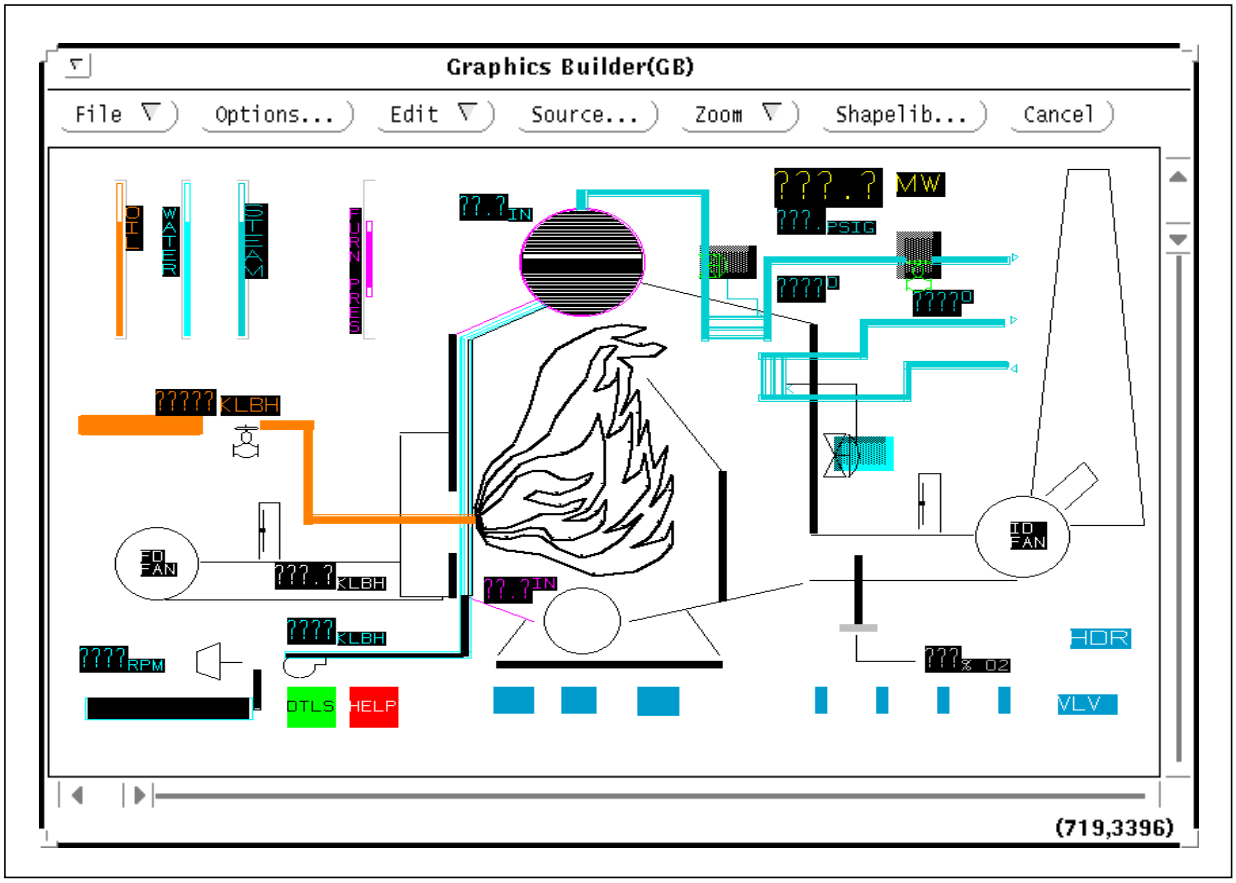
Table 3-2. Origin by Draw Item (Cont'd)

Super Process Point	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector text).
Text	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector).
Time	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector).
Trend	Lower right corner (for horizontal).
	Upper right corner (for vertical).
Unacknowledged Alarm Count	Baseline position of the first character of the text string (for bitmap text).
	Upper left corner of the first character of the text string (for vector text).
XY Plot	Lower left corner.

3-5. Sample File

The figure below shows a partial source program. The following figure shows the diagram that is produced from the file. The required syntax and definition for each graphics language command are defined in the reference pages in Section 3-6.

```
 1.
 2. DIAGRAM MAIN 0 96 282 844 610 white ZOOMABLE 0 0 0 16383 16383 1 DEFAULT_POSITION
 3. DEFAULT_SIZE
 4.
 5. KEYBOARD
 6. COLOR FG black BG white BLINK FG OFF BG OFF
 7. POKE_FLD 12544 2234 538 1117 ON 7 2 117 8102 5 0 79 2 0 0 6 0 5 A004X046 ID A004X046 ID  1 9 2
 8. POKE_FLD 9677 2607 717 744 ON 7 2 117 8107 5 0 79 2 0 0 6 0 5 A004X028 ID A004X027 ID  1 9 2
 9. COLOR FG cyan BG black BLINK FG OFF BG OFF
10. POKE_FLD 11628 7820 757 744 ON 7 2 117 8108 5 0 79 2 0 0 6 0 5 A004X038 ID A004X037 ID 1 9
11. 2
12.
13. PAGE 2713 2704 3700 2705 0 0 0 0
14.
15. BACKGROUND
16.
17. COLOR FG white BG black BLINK FG OFF BG OFF
18. TEXT 4301 11543  "KLBH" HORZ VECTOR_OVER 179 372 1
19. COLOR FG black BG green BLINK FG OFF BG OFF
20. TEXT 3584 14522  "    " HORZ VECTOR_OVER 179 372 1
21. TEXT 3584 14895  "DTLS" HORZ VECTOR_OVER 179 372 1
22. TEXT 3584 15267  "    " HORZ VECTOR_OVER 179 372 1
23. COLOR FG white BG red BLINK FG OFF BG OFF
24. TEXT 4480 14522  "    " HORZ VECTOR_OVER 179 372 1
25. TEXT 4480 14895  "HELP" HORZ VECTOR_OVER 179 372 1
26. TEXT 4480 15267  "    " HORZ VECTOR_OVER 179 372 1
27. COLOR FG white BG DeepSkyBlue3 BLINK FG OFF BG OFF
28. TEXT 14855 14738  "VLV " HORZ VECTOR_OVER 226 548 1
29. TEXT 15034 12945  "HDR" HORZ VECTOR_OVER 304 548 1
30. COLOR FG gray75 BG black BLINK FG OFF BG OFF
31. LINE 1050 745 1203 745 1203 5048 1050 5048 1 solid
32. LINE 2150 745 2022 745 2022 5048 2150 5048 1 solid
33. LINE 2842 745 2995 745 2995 5048 2842 5048 1 solid
34. LINE 4838 745 4685 745 4685 5048 4838 5048 1 solid
35. CURSOR 384 827
36. TRIG_ON 9
37. CURSOR 947 827
38. TRIG_ON 9
39. CURSOR 2176 827
40. TRIG_ON 9
41. CURSOR 2739 827
42. TRIG_ON 9
43. CURSOR 4864 827
44. TRIG_ON 9
45. COLOR FG darkturquoise BG black BLINK FG OFF BG OFF
46. TEXT 2947 1372  "STEAM" VERT VECTOR_OVER 367 419 1
47. COLOR FG magenta BG black BLINK FG OFF BG OFF
48. TEXT 4480 1489  "FURN PRES" VERT VECTOR_OVER 179 372 1
49. TEXT 7168 11543  "IN" HORZ VECTOR_OVER 179 372 1
50. COLOR FG black BG black BLINK FG OFF BG OFF
51. RECTANGLE 5939 4923 102 4261 1 solid solid
52. RECTANGLE 5939 10881 102 1200 1 solid solid
53. RECTANGLE 6630 13819 3302 165 1 solid solid
54. RECTANGLE 9882 8647 102 3517 1 solid solid
55. RECTANGLE 11213 4675 102 5668 1 solid solid
56. COLOR FG magenta BG black BLINK FG OFF BG OFF
57. LINE 6093 11998 7168 12743 1 solid
58. COLOR FG black BG black BLINK FG OFF BG OFF
59. LINE 6656 13653 7040 12702 1 solid
60. LINE 5222 12081 5222 7696 1 solid
```



3-6. Graphics Language Commands

The Graphics Language consists of two types of commands:

- **Section labels** define the major portions of a diagram. Section labels correspond to "Place" in the Graphics Builder (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on place). See Section 3-6.1 for more information on section labels.
- **Main commands** define the displayable and non-displayable items and their properties. See Section 3-6.2 for more information on main commands.

3-6.1. Section Labels

The following section labels define the major portions of the diagram source file:

- BACKGROUND.
- DIAGRAM.
- EXIT.
- FOREGROUND.
- KEYBOARD.
- TRIGGER.

Each section label is described on separate reference sheets, arranged alphabetically, in this section.

Description

The BACKGROUND command defines the background section of a diagram. The commands in the background section of the diagram are executed when the diagram is first displayed and when a portion of the window becomes uncovered. The commands in the background section do not update every second.

Syntax

```
BACKGROUND
```

Rules

1. The BACKGROUND command can be used only once in a diagram.
2. The background section of the diagram ends when a FOREGROUND, TRIGGER, KEYBOARD, or EXIT command is encountered or when the end of the file is reached.

Note

See Table 3-1 for a list of commands that may be placed in the BACKGROUND section.

DIAGRAM

Description

The DIAGRAM command defines the section of the diagram that contains commands that apply to the overall diagram (as opposed to only the display items in the background, foreground, keyboard, or trigger sections). A DIAGRAM command automatically appears in the source editor when the Graphics Builder initially starts. Users can use the given settings or change any of the parameters.

Syntax

```
DIAGRAM diag_type icon_num x y w h bg_color zoom_flag
revision_num x_extents y_extents width_extents height_extents
update_rate positioning sizing subscreen_num
```

where:

diag_type = Diagram type. Choose one of the following:

- MAIN
- SUBWIN
- WINDOW

See Section 3-3 for more information on diagram types.

icon_num = Allows the user to specify a unique icon to be displayed for a diagram when the user "closes" the diagram window. Custom icons can be created in the icon editor (see " WEStation Graphics Builder User's Guide" (U0-8210) and applicable Solaris documentation for more information on the icon editor). The icon is specified by a number. Valid range = 0 through 65,535.

x = x coordinate that corresponds to the upper left corner of the diagram on the Operator WEStation. Not used if DEFAULT_POSITION is selected for the positioning parameter (see positioning parameter below). Valid range = 0 through 1151 (entered in screen coordinates; see Section 3-4 for more information on coordinates).

y = y coordinate that corresponds to the upper left corner of diagram on the Operator WEStation. Not used if DEFAULT_POSITION is selected for the positioning parameter (see positioning parameter below). Valid range = 0 through 899 (entered in screen coordinates; see Section 3-4 for more information on coordinates).

w = Width of the diagram to be displayed at the Operator WEStation. Not used if DEFAULT_SIZE is selected for the sizing parameter (see sizing parameter below). Valid range = 100 through 1152 (entered in screen coordinates; see Section 3-4 for more information on coordinates).

h = Height of the diagram to be displayed at the Operator WEStation. Not used if DEFAULT_SIZE is selected for the sizing parameter (see sizing parameter below). Valid range = 100 through 900 (entered in screen coordinates; see Section 3-4 for more information on coordinates).

bg_color = Background color for the diagram. Users can choose one of the eight standard colors (black, blue, green, cyan, red, magenta, yellow, and white) or one of the 248 possible custom colors. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on defining custom colors.

zoom_flag = Zoom flag. Choose one of the following:

- ZOOMABLE - the diagram can be zoomed at the Operator WEStation.

- NON_ZOOMABLE - the diagram cannot be zoomed at the Operator WEStation.

revision_num = Revision level of the diagram (user-defined). Valid range = 0 through 2,147,483,647.

x_extents = x virtual coordinate that corresponds to the upper left corner of the zoom extents rectangle. See Section 3-2 for more information on the zoom extents rectangle.

y_extents = y virtual coordinate that corresponds to the upper left corner of the zoom extents rectangle. See Section 3-2 for more information on the zoom extents rectangle.

width_extents = Width that corresponds to the virtual width of the zoom extents rectangle when zoom extents is activated. See Section 3-2 for more information on the zoom extents rectangle.

height_extents = Height that corresponds to the virtual height of the zoom extents rectangle when zoom extents is activated. See Section 3-2 for more information on the zoom extents rectangle.

update_rate = Number of seconds before an update of foreground items will occur.
Valid range = 1 through 60,
0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9.

positioning = Positioning. Choose one of the following:

- DEFAULT_POSITION - the diagram will be positioned at the same position as the last displayed diagram.
- FIXED_POSITION - the diagram will be positioned at the x and y coordinates specified by the x and y parameters in this command.

sizing = Sizing. Choose one of the following:

- DEFAULT_SIZE - the diagram will use the same size as the last displayed diagram.

- FIXED_SIZE - the diagram will use the size specified by the width and height parameters (w and h) in this command.

[subscreen_num] = Subscreen diagram number.
Valid range = 0 through 65535. This argument is optional. It specifies a subwindow diagram to display whenever the diagram displays. This argument cannot be specified if diag_type = SUBWIN.

Rules

1. The DIAGRAM command statement is required. It must be the first command in the graphic (with the exception of the optional TITLE command). It can only be used once in the graphic. (The TITLE command is available for backward compatibility to 7.0 and earlier. Refer to "WDPF Graphics Reference Manual" (U0-0286) for additional information.)

2. The diagram section ends when a BACKGROUND, FOREGROUND, TRIGGER, KEYBOARD, or EXIT command is encountered or when the end of the file is reached.

3. A warning message is generated if the user selects a diagram update rate **less than 1 second** since this may degrade processor performance.

Note

See Table 3-1 for a list of commands that may be placed in the DIAGRAM section.

Example

```
DIAGRAM MAIN 1 395 0 631 588 WhiteSmoke ZOOMABLE 1 0 0
16383 16383 3 DEFAULT_POSITION DEFAULT_SIZE 2500
```

In this example:

diag_type = MAIN

```
        icon_num  =  1
               x  =  395
               y  =  0
               w  =  631
               h  =  588
        bg_color  =  WhiteSmoke
       zoom_flag  =  ZOOMABLE
    revision_num  =  1
        x_extents  =  0
        y_extents  =  0
    width_extents  =  16383
   height_extents  =  16383
     update_rate  =  3
     positioning  =  DEFAULT_POSITION
          sizing  =  DEFAULT_SIZE
   subscreen_num  =  2500
```

This statement defines a main screen diagram with an icon number of 1. The x, y coordinates of the diagram are 395, 0 and the width and height are 631, 588. The background color is WhiteSmoke. The diagram is zoomable at the Operator WEStation. The revision number is 1.

The zoom extents rectangle uses 0, 0 as the x, y position of the upper left corner and 16383, 16383 as the width and height (zoom extents rectangle will be the entire virtual canvas). The foreground items in the diagram update every three seconds. This diagram displays at the same position as the last displayed diagram and uses the same size as the last displayed diagram.

A subwindow diagram (defined by the number 2500) displays along with this main diagram.

Description

The EXIT command is an optional command that defines the end of the diagram. It is primarily found in older graphics files that were not created using the Graphics Builder source editor. When creating new files via the Graphics Builder, the EXIT command is not supported (EOF is used to terminate the file).

Syntax

```
EXIT
```

Rules

1. The EXIT command can be used only once in a diagram.
2. If the EXIT command is not used, end-of-file terminates the diagram.
3. If the EXIT command is used, nothing (including comments) is allowed after it.
4. When the Graphics Builder loads a source file containing the EXIT command, it removes it.

FOREGROUND

Description

The FOREGROUND command defines the foreground section of the diagram. The commands in the foreground section are executed when the diagram is initially displayed and periodically thereafter at a frequency determined by the update_rate parameter in the DIAGRAM command (see DIAGRAM command reference page).

Syntax

```
FOREGROUND
```

Rules

1. The FOREGROUND command can be used only once in a diagram.
2. The foreground section of the diagram ends when a BACKGROUND, KEYBOARD, TRIGGER, or EXIT command is encountered.

Note

See Table 3-1 for a list of commands that may be placed in the FOREGROUND section.

Description

The KEYBOARD command defines the keyboard section of the diagram. The commands in the keyboard section define the operator interfaces to the diagram.

Syntax

```
KEYBOARD
```

Rules

1. The KEYBOARD command can be used only once in a diagram.
2. The keyboard section of the diagram ends when a BACKGROUND, FOREGROUND, TRIGGER, or EXIT command is encountered.

Note

See Table 3-1 for a list of commands that may be placed in the KEYBOARD section.

TRIGGER

Description

The TRIGGER command defines a trigger section of the diagram. Trigger sections of the diagram are activated on demand by a graphics application program (see Section 4) or by the user-specified TRIG_ON command (see the TRIG_ON reference page).

Syntax

```
TRIGGER n
```

where:

n = Trigger region number.
Valid range = 1 through 255

Rules

1. There can be a maximum of 255 trigger regions defined in a diagram.
2. The trigger region is only executed on demand.
3. The trigger section of the diagram ends when another TRIGGER command or a BACKGROUND, FOREGROUND, KEYBOARD, or EXIT command is encountered.

Note

See Table 3-1 for a list of commands that may be placed in the TRIGGER section.

Example

```
TRIGGER 255

COLOR FG black BG white BLINK FG OFF BG OFF

LINE 3291 1068 8481 997 2943 4452 8597 4452 3314 1104 1 solid

KEYBOARD
```

In this example:

n = 255

Trigger area 255 is shown in this example. It contains the main command statements COLOR and LINE. These commands execute when this trigger is called. This trigger area ends when the KEYBOARD command is encountered.

3-6.2. Main Commands

The graphics main commands define display items, get data, set data, and control diagram execution. They appear in the source file after the section labels. The main commands are listed below and are described later in reference pages:

ARC	IF_ENDIF	POLYGON
BAR	IFELSE/ELSE/ENDIF	PROCESS_PT
BLINK	LINE	PTR_EQUAL
CIRCLE	LOAD_FKEY_GROUP	PTR_LOOP/P_ENDLOOP
COLOR	LOOP/ENDLOOP	PTR_MOVE
CURSOR	MACRO	PTR_VALUE
DATE	MATH	RECTANGLE
DEF_FKEY_GROUP	MULTI_TEXT	RUN_PROGRAMS
DEF_QUAL	OL_BUTTON	SETVAL
DIAG_DISP	OL_CHECKBOX	SHAPE
DOT	OL_CHOICE	SW_ALARM_COUNT
DYNAMIC_LINE	OL_CYLINDER	SW_ALARM_CUE
EF_STATE	OL_EVENT_MENU	SW_ALERT_LIST
ELLIPSE	OL_GAUGE	SW_PROCESS_PT
ENTRY_FLD	OL_RECTANGLE	TEXT
FKEY_STATE	OL_SLIDER	TIME
FORCE_UPDATE	PAGE	TREND
FUNC_KEY	PLOT	TRIG_ON
GCODE	POINTER	XY_PLOT
GTEXT	POKE_FLD	
IF_CHANGED/ENDIF	POKE_STATE	

Note

SW_ALARM_COUNT, SW_ALARM_CUE, SW_ALERT_LIST, and SW_PROCESS_PT are only used with the Alarm Management system (see "Alarm Management System User's Guide" (U0-8115) for more information).

Description

The ARC command draws a filled or unfilled circular or elliptical arc.

Syntax

```
ARC x y w h angle1 angle2 line_width line_pat
<line_conditional_expression> fill_pat
<fill_conditional_expression>
```

where:

x = Upper left corner of the outlining rectangle around the circle or ellipse on which the arc lies (note that this is not the same as the arc's outlining rectangle). Valid range = -16383 through 16383 (a negative number indicates that the upper left corner of the outlining rectangle will be off the screen).

y = Upper left corner of outlining rectangle (see definition for x above).
Valid range = -16383 through 16383 (a negative number indicates that the upper left corner of the outlining rectangle will be off the screen).

w = Width of outlining rectangle (see definition for x above).
Valid range = 1 through 32,767.

h = Height of outlining rectangle (see definition for x above).
Valid range = 1 through 32,767.

angle1 = Start angle in degrees.
Valid range = -360 through 360.

angle2 = Number of degrees to draw the arc.
Valid range = -360 through 360. (Negative degrees will result in clockwise drawing of the arc. Positive degrees will result in counterclockwise drawing of the arc.)

line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.
line_pat	=	Line pattern name.
<line_conditional_ expression>	=	Optional conditional for line pattern.
fill_pat	=	Fill pattern name.
<fill_conditional_ expression>	=	Optional conditional for fill pattern.

Note

See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on defining line patterns and fill patterns.

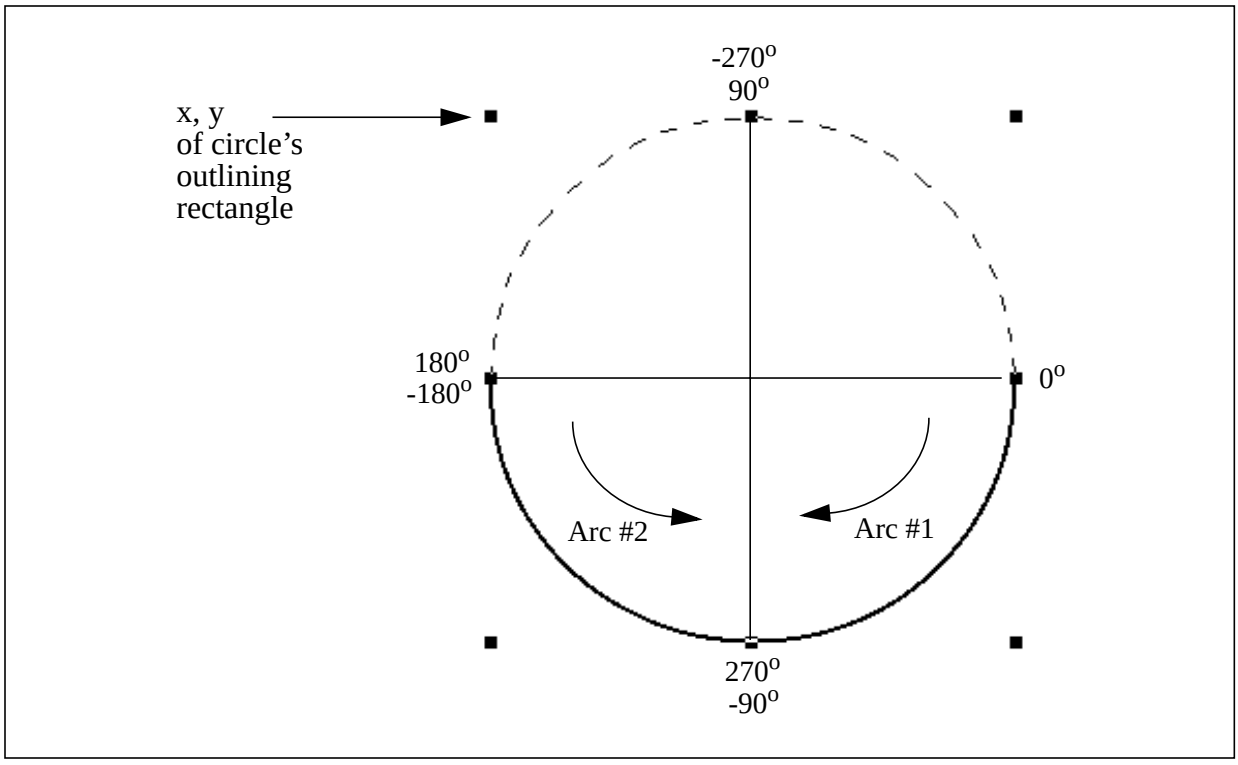
Rules

1. The x and y coordinates can be relative.

2. The arc's origin is the center of the circle or ellipse on which the arc lies.

3. The arc is drawn the number of degrees specified for the angle2 parameter (see syntax above) beginning at the start angle. The sign of angle2 determines whether the arc is drawn clockwise or counterclockwise.

   The figure below shows the outlining rectangle of a circle divided into 90^o segments to illustrate the effect of positive and negative angles for the arc. If the angle1 parameter is 0 and the angle2 parameter is -90, the arc will be drawn as shown by Arc #1 (the arc is drawn clockwise since angle2 is negative). In contrast, if the angle1 parameter is 180 and the angle2 parameter is 90, the arc will appear as shown by Arc #2 (the arc is drawn counterclockwise since angle2 is positive).



4. Both the line pattern name and the fill pattern name may be conditional parameters (the user may specify a conditional to determine which line pattern or fill pattern is used when displaying the arc).

5. The ARC command may be used in the background, foreground, or trigger sections of a diagram.

Examples

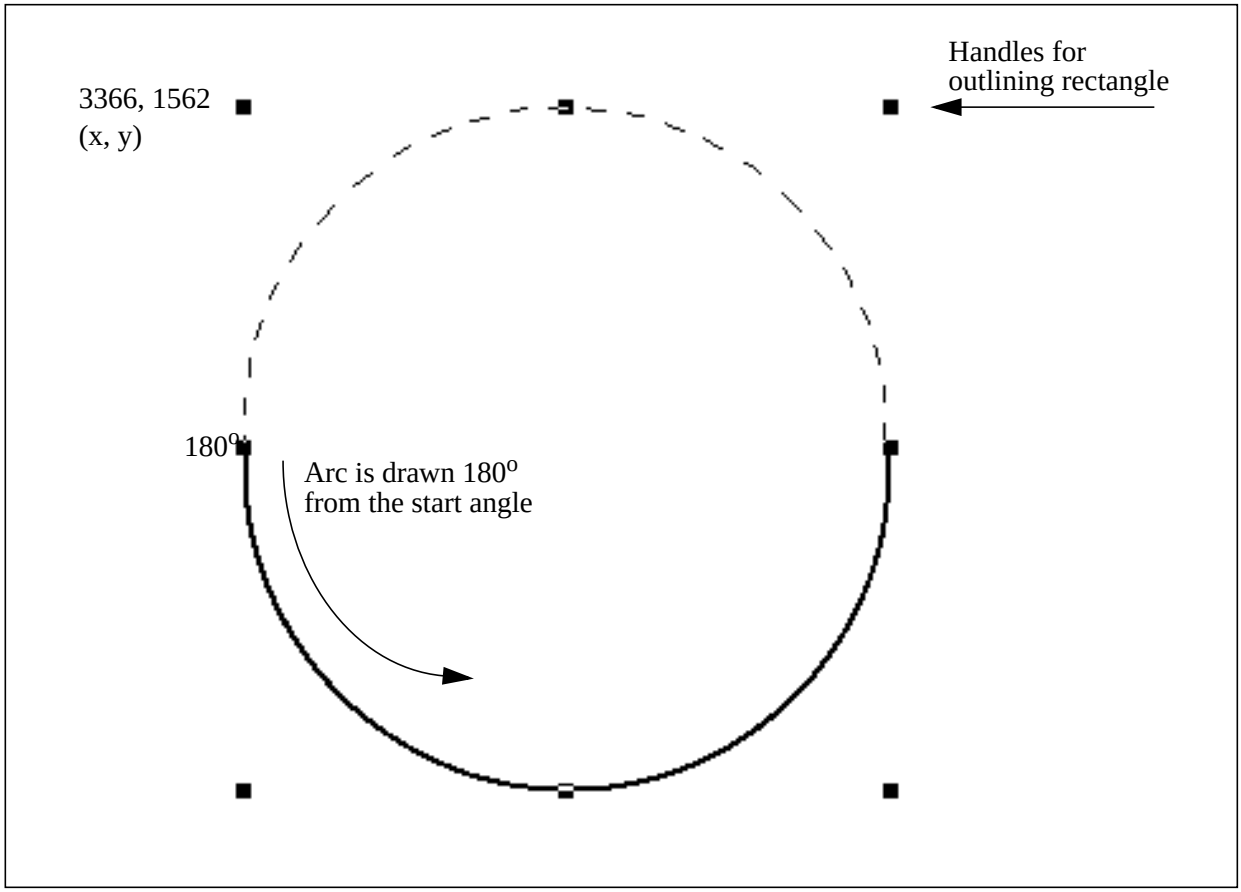
Example 1

```
ARC 3366 1562 6262 10683 180 180 2 solid unfilled
```

In this example:

x coordinate	=	3366
y coordinate	=	1562
w (width)	=	6262
h (height)	=	10683
angle1	=	180
angle2	=	180
line_width	=	2
line_pat	=	solid
fill_pat	=	unfilled

This statement defines an arc with the upper left corner of the outlining rectangle (around the circle or ellipse on which the arc lies) at 3366, 1562 (to show this more clearly, an outlining rectangle for the circle is drawn with the arc in the figure below). The width and height of the outlining rectangle is 6262, 10683. The first angle starts at 180 degrees, and the arc is drawn 180 degrees (the arc is drawn counterclockwise since angle2 is positive; see Rules for more information on drawing angles). The line width is 2. The line pattern is solid and the fill pattern is unfilled. See figure below.



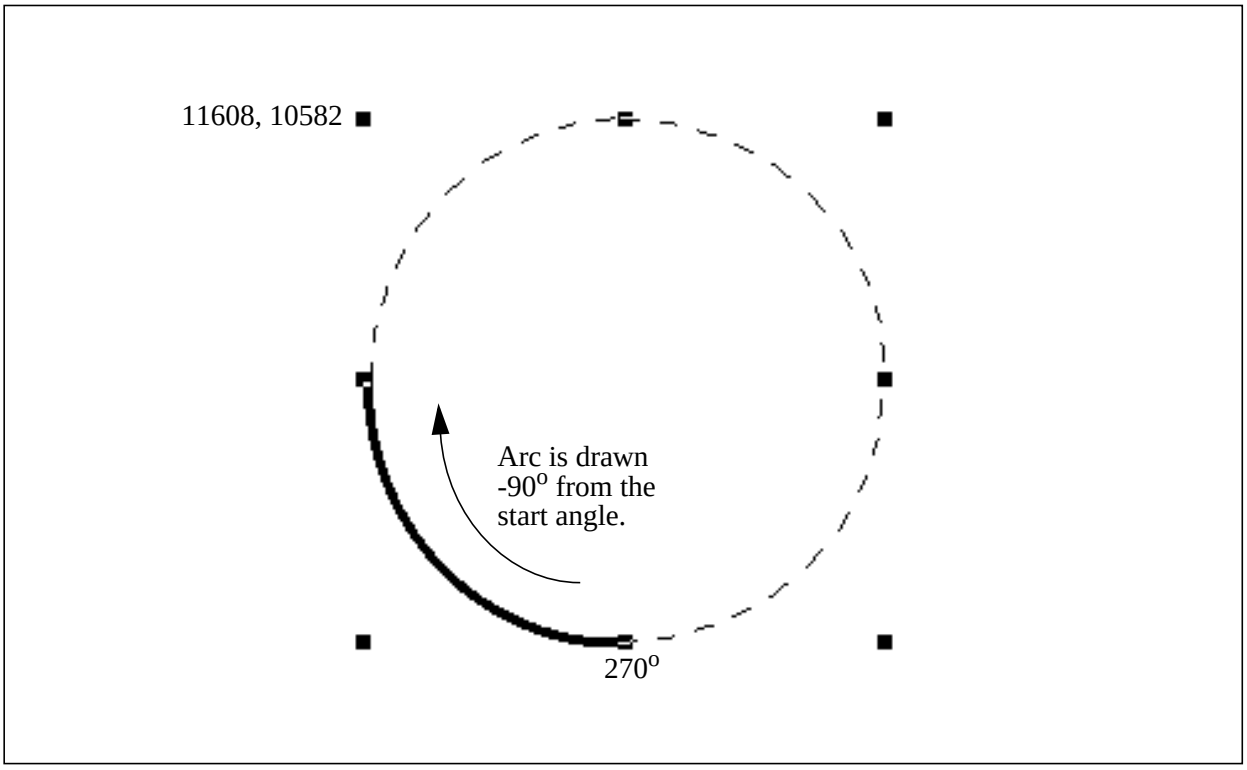
Example 2 (Conditional)

```
ARC 11608 10582 5034 8924 270 -90 3 solid (A404 AV <= 100)
dashed unfilled (A404 = ALARM) blocks
```

In this example:

x coordinate	=	11608
y coordinate	=	10582
w (width)	=	5034
h (height)	=	8924
angle1	=	270
angle2	=	-90
line_width	=	3
line_pat	=	solid
line_pattern_conditional	=	(A404 AV <= 100) dashed
fill_pat	=	unfilled
fill_pattern_conditional	=	(A404 = ALARM) blocks

This statement defines an arc with the upper left corner of the outlining rectangle (around the circle or ellipse on which the arc lies) at 11608, 10582. The width and height of the outlining rectangle is 5034, 8924. The first angle starts at 270 degrees, and the arc is drawn -90 degrees (the arc is drawn clockwise from the start angle since angle 2 is negative). The line width is 3. See the figure below:



A line pattern conditional is used. If A404 AV <= 100, the line pattern for the arc will be dashed. Otherwise, the line pattern will be solid.

A fill pattern conditional is also specified. If A404 = ALARM, the fill pattern for the arc will be blocks. Otherwise, the fill pattern will be unfilled.

See Section 2 for more information on conditionals.

BAR

Description

The BAR command defines a solid rectangular shape (bar) on a diagram. The bar represents the current value of a process point, scaled between the low and high limits defined for that point.

Syntax

```
BAR x y w h direction pt_ name rec_fld low high
```

where:

x = Origin of bar.
Valid range = 0 through 16,383.

y = Origin of bar.
Valid range = 0 through 16,383.

w = Width of bar (size of bar in direction perpendicular to fill direction).
Valid range = 1 through 16,383.

h = Height of bar (maximum size of bar in direction that it will be filled).
Valid range = 1 through 16,383.

direction = Direction in which bar is filled. The choices are:

- LEFT = Right to left.
- RIGHT = Left to right.
- DOWN = Top to bottom
- UP = Bottom to top
- BIAS = Up or down from the 0 value position

pt_name = Name of point to be represented by the bar.

rec_fld = Optional. Name of record field to be scaled by the bar. If no record field is specified, the default record field for the given point type is assumed (see Section 2 for more information on default record fields).

low = Low limit used to scale value. Valid values are

- Integers in the range -2,147,483,647 through 2,147,483,647.
- Real constants
- Point names/record fields
- Pointers/offsets

high = High limit used to scale value. Valid values are

- Integers in the range -2,147,483,647 through 2,147,483,647.
- Real constants
- Point names/record fields
- Pointers/offsets

Rules

1. The x and y coordinates can be relative.
2. The origin will be one of the following:
   - Upper left corner of the bar (down or right or bias).
   - Lower left corner of the bar (up).
   - Upper right corner of the bar (left).
3. Opposite ends of the bar represent the low and high scale values of the process point.
4. The actual process point value is scaled between the low and high limits of the bar, and the bar is filled from the low limit end in the direction specified, up to the high scale process point value.
5. The bias bar fills from the 0 value position. The 0 position is determined by the low and high limits; 0 must fall somewhere between these limits. If the actual bar point value is > 0, the bar fills up from the 0 position. If the actual bar point value is < 0, the bar fills down from the 0 position. The bar fills up OR down and the 0 position may be dynamic.
6. The current erase color (ER) is used to display the unfilled portion of the maximum size of the bar. That is, if a UP type bar has a height of n, and the bar value is scaled to a height of n/4, then one quarter of the bar will be displayed in the current foreground color (FG), and the remaining three quarters of the bar will be drawn in the current erase color. The current erase color should be the same as the background color over which the bar is displayed. The Graphics Builder does not use the erase color when displaying the bar; the process diagram system does.
7. The BAR command may reside in the background, foreground, or trigger sections of a diagram.

Note

If BAR is used in the background section of a diagram, it will not update.

Example

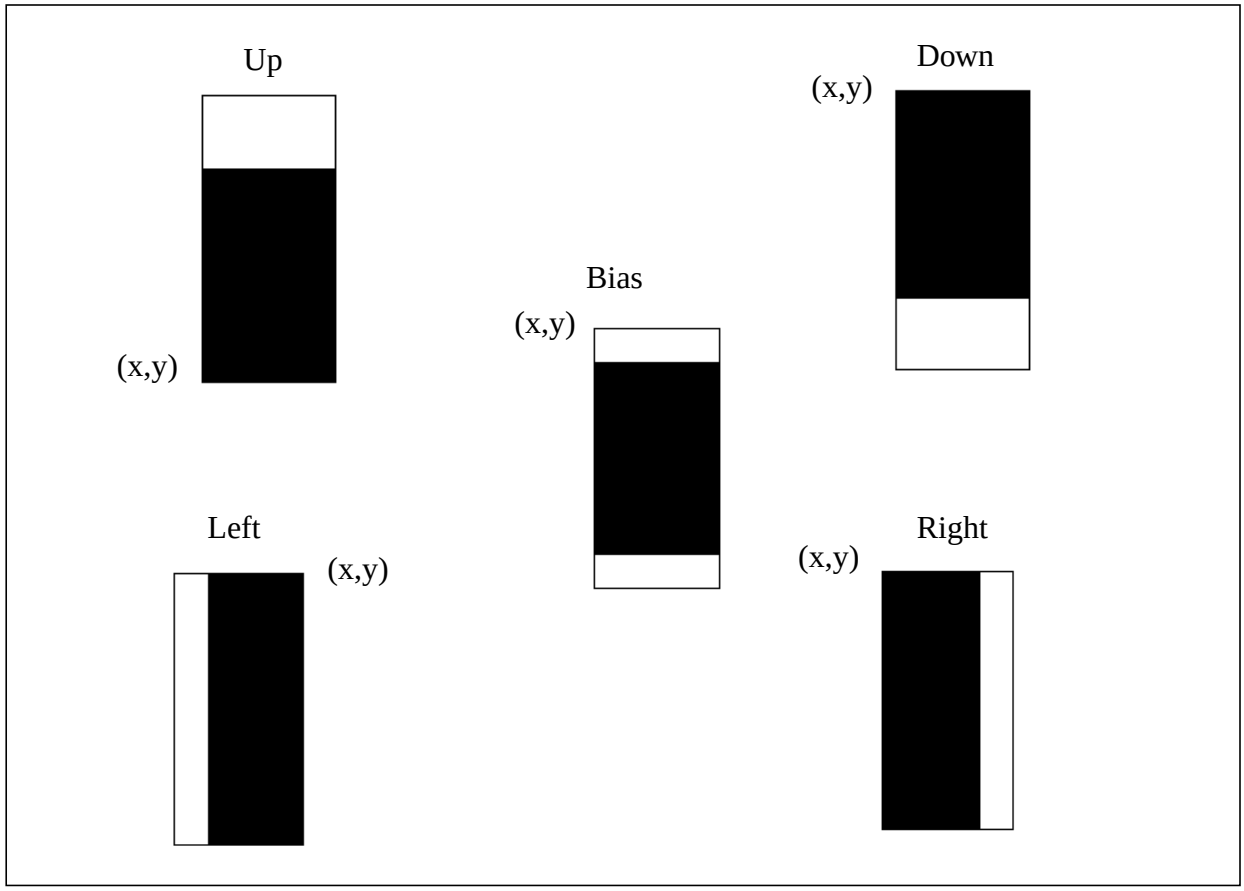
```
BAR 5951 13305 2179 10488 UP A3890 AV 20 80
```

In this example:

x coordinate	=	5951
y coordinate	=	13305
w (width)	=	2179
h (height)	=	10488
direction	=	UP
pt_name	=	A3890
rec_fld	=	AV
low	=	20
high	=	80

In this statement, the bar has an origin of 5951, 13305. The width of the bar is 2179 and the height is 10488. The direction of the bar fill is up. The bar is representing
process point A3890 and record field AV (analog value). The bar will be scaled from a low limit of 20 to a high limit of 80.

The figure below shows the five different types of bars when displayed on the Graphics Builder interactive editor (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on using the interactive editor). Notice that an arbitrary point value is assumed for bars in the Graphics Builder. The bars are not displayed at maximum value to distinguish them form solid filled rectangles.When the bar is displayed in the process diagram, the bar will fill to a variable amount, depending on the current value of the process point/record field specified.



Description

The BLINK command resets the foreground and/or background blink attributes for all subsequent display items in a graphic until the next BLINK command.

Caution

The use of blink in graphics is highly discouraged because there is no hardware support for blink on the current workstation environment. The software implementation of blink degrades the diagram update rate.

Syntax

```
BLINK FG blink_state <conditional expression> BG blink_state
<conditional expression>
```

where:

FG	=	The letters ‘FG’ for foreground.
BG	=	The letters ‘BG’ for background.
blink_state	=	Blink state. The choices are: • ON (blink will be turned on). • OFF (blink will be turned off).
<conditional_expression>	=	Optional conditional for blink state.

Rules

1. The foreground and background blink states are conditional parameters (the user may specify a conditional expression to determine whether or not the display items will blink).

2. The BLINK command sets the blink attributes for all display items until another BLINK command is encountered or until a new diagram section is defined.

3. The BLINK command may be used in the background, foreground, or trigger sections of a diagram.

4. The BLINK command may be combined with the COLOR command (that is, may be on the same line). For example: "COLOR FG RED BLINK FG ON", or "BLINK FG ON BG ON COLOR FG RED BG GREEN". Either command may be appended to the other command (that is, BLINK after COLOR or COLOR after BLINK). These commands are valid individually or when combined. When the .diag file format of a graphic is loaded into the Graphics Builder, and when color is assigned interactively in the Graphics Builder, the BLINK command is automatically appended to the COLOR command in the source file.

Examples

Example 1

```
BLINK FG OFF BG OFF
```

In this example:

FG blink_state = OFF

BG blink_state = OFF

Blink will be off in both the foreground and the background sections.

Example 2 (Conditional)

```
BLINK FG OFF {(D007 LL = 101.5) ON (PG1001 ID >= PG 1010) ON}
BG OFF
```

In this example:

FG blink_state = OFF

conditional (FG) = {(D007 LL = 101.5) ON (PG1001 ID >= PG1010) ON}

BG blink_state = OFF

If D007 LL = 101.5 the foreground blink state will be ON. If the first conditional evaluates to false, the second conditional will be considered. If PG1001 ID >= PG1010, the foreground blink state will be ON. If the second conditional also evaluates to false, the foreground blink state will be OFF (default). The background blink state will be OFF (no conditional specified).

See Section 2 for more information on conditionals.

CIRCLE

Description

The CIRCLE command draws a circle with a specified center and radius.

Syntax

```
CIRCLE cx cy r line_width line_pat <line_conditional_
expression> fill_pat <fill_conditional_expression>
```

where:

cx	=	x coordinate of center of circle. Valid range = 0 through 16,383.
cy	=	y coordinate of center of circle. Valid range = 0 through 16,383.
r	=	Radius of circle (see Rules below). Valid range = (r > 0, cx + r <= 16383 and cx - r >= 0; cy + r <= 16383, cy - r >= 0)
line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.
line_pat	=	Line pattern name.
<line_conditional_expression>	=	Optional conditional for line pattern.
fill_pat	=	Fill pattern name.
<fill_conditional_expression>	=	Optional conditional for fill pattern.

Note

See "WEStation Graphics Builder User's Guide" (U0-8210) for information on defining line patterns and fill patterns.

Rules

1. The x and y coordinates can be relative.

2. The circle's origin is the center of the circle.

3. The line pattern and fill pattern names are conditional parameters (the user may specify conditionals to determine which line pattern or fill pattern is used to display the circle).

4. The following rules apply to the "r" argument:

   The **r** argument on the source command line is specified in virtual coordinates. The drawing routine that draws a circle on the screen uses screen coordinates (see Section 3-4 for more information on coordinates). Unique conversion factors exist for converting virtual to screen coordinates for the x and y dimensions.

   These conversion factors are based on the drawing area width and height. If the drawing area width and height are the same (for a square drawing area), the conversion factors are the same. If the width and height are not the same (for a non-square drawing area) the conversion factors are different.

   If the user is adding a circle to a square drawing area, the **r** argument is used for both the x and y radii because (after converting to screen coordinates) the x and y radii are identical.

   However, if the user is adding the circle to a non-square drawing area, the x and y radii are not identical (after converting to screen coordinates) because the conversion factors are different. Thus, the item would display as an ellipse. To allow the user to draw a circular item on a non-square screen, the **r** argument is read as the x radius, and then a unique y radius is calculated such that the conversion to screen coordinates produces identical x and y radii.

The following formula for the y virtual radius is used:

ry = rx * (w/h)

where:

ry = virtual y radius

rx = virtual x radius ["r"]

w = drawing area width = diagram width - 19

h = drawing area height = diagram height - 52

5. The CIRCLE command may be used in the background, foreground, or trigger sections of a diagram.

Examples

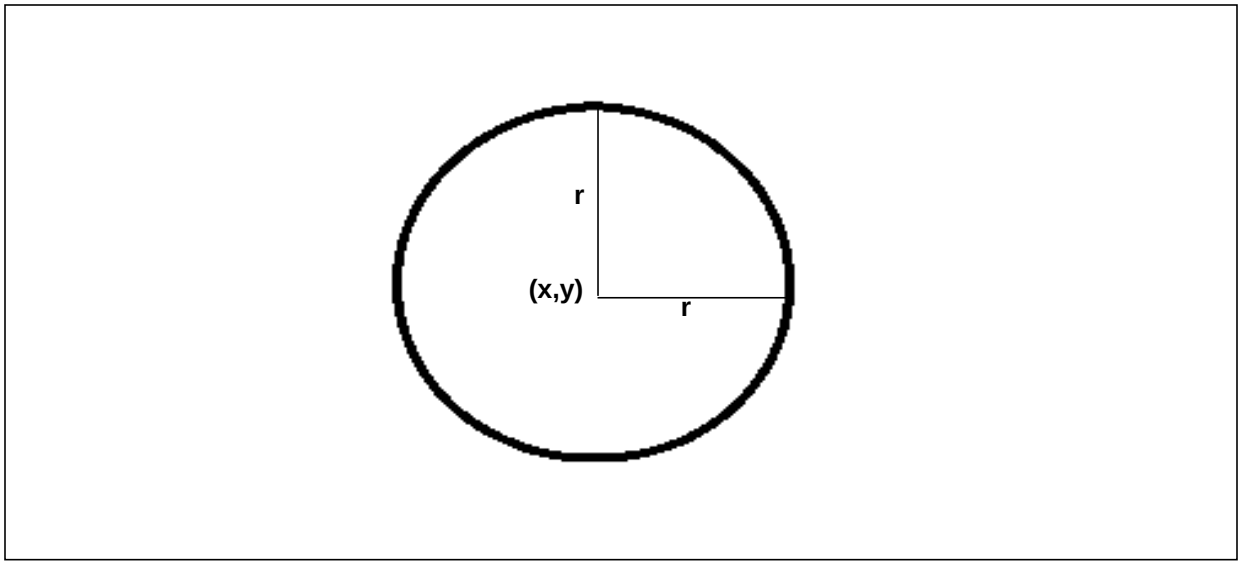
Example 1

```
CIRCLE 7671 6349 2145 3 solid unfilled
```

In this example:

cx	=	7671
cy	=	6349
r	=	2145
line_width	=	3
line_pat	=	solid
fill_pat	=	unfilled

The statement defines a circle with the center defined at 7671, 6349. The radius of the circle is 2145. The line width is 3. A solid line pattern is used for the circle, and it is unfilled. See the figure below.



Example 2 (Conditional)

```
CIRCLE 10582 2392 1132 2 solid unfilled (CASE) D050 LL 1 1 2
asterisks solid
```

In this example:

cx	=	10582
cy	=	2392
r	=	1132
line_width	=	2
line_pat	=	solid
fill_pat	=	unfilled
fill _pattern_ conditional	=	(CASE) D050 LL 1 1 2 asterisks solid

The statement defines a circle with the center defined at 10582, 2392. The radius of the circle is 1132. The line width is 2. A solid line pattern is used for the circle.

A fill pattern conditional is used in this example. The fill pattern for the circle will change based on the value of D050 LL as described below:

- If D050 LL < 2, the fill pattern will be unfilled (default parameter).
- If D050 LL >= 2 and if D050 LL < 3, the fill pattern will be asterisks.
- If D050 LL >= 3 and if D050 LL < 4, the fill pattern will be solid.
- If D050 LL >= 4, the fill pattern will be unfilled (default parameter).

See Section 2 for more information on conditionals.

Description

The COLOR command sets the foreground, background, erase, and OPEN LOOK drawing colors for subsequent display items in a graphic.

Syntax

```
COLOR FG color <conditional expression> BG color
<conditional expression> ER color <conditional expression> OL
ol_color_index <conditional expression>
```

where:

FG	=	The letters ‘FG’ for foreground.
BG	=	The letters ‘BG’ for background.
ER	=	The letters ‘ER’ for erase.
color	=	Color name from the colors.txt file. This file allows the user to defines up to 233 colors in addition to the eight standard colors (black, blue, green, cyan, red, magenta, yellow, and white). See “WEStation Graphics Builder User’s Guide” (U0-8210) for more information on the colors.txt file.
OL	=	The letters ‘OL’ for OPEN LOOK ™.
ol_color_index	=	Integer color index associated with the OL_COLOR command in the colors.txt file. Valid range = 0 through 4 See “WEStation Graphics Builder User’s Guide” (U0-8210) for information on the colors.txt file.
<conditional _ expression>	=	Optional conditional for foreground, background, erase, or OPEN LOOK color.

Rules

1. The foreground, background and erase color names are conditional parameters (the user may specify a conditional expression to determine which colors are used under differing circumstances). The OPEN LOOK color index is also a conditional parameter

2. OL color is only used for OPEN LOOK draw items.

3. The COLOR command sets the colors for all display items until another COLOR command is encountered or until a new diagram section is defined.

4. FG, BG, ER, and OL colors do not need to be defined in each COLOR statement. If one of these parameters is not defined, the previous color specified for that parameter will be used.

5. Erase color is typically used only in the process diagram display. It is used for the display of bars, shapes, OL Checkboxes, OL Sliders, and OL Cylinders on the process diagrams. It is only used for the OL Cylinder and OL Checkbox display in the Graphics Builder.

   Erase color is used to draw the "invisible" portion of a bar (that is, it is used for the unfilled portion of a bar). It is used to erase the existing shape when a shape changes as result of a conditional expression change. It is used to erase the portion of the checkmark which extends outside of the box for the OL Checkboxes. It is used to erase the slider control as the user drags/moves the control for the OL Slider, and it is used to erase some intermediate drawing in the lower right corners of the OL Cylinder.

6. The COLOR command may reside in the background, foreground, keyboard, or trigger sections of a diagram.

7. The BLINK command may be combined with the COLOR command (that is, may be on the same line). For example: "COLOR FG RED BLINK FG ON", or "BLINK FG ON BG ON COLOR FG RED BG GREEN". Either command may be appended to the other command (that is, BLINK after COLOR or COLOR after BLINK). These commands are valid individually or when combined. When the .diag file format of a graphic is loaded into the Graphics Builder, and when color is assigned interactively in the Graphics Builder, the BLINK command is automatically appended to the COLOR command in the source file.

Examples

Example 1

```
COLOR FG red BG white ER yellow OL 0
```

In this example:

color (foreground)	=	red
color (background)	=	white
color (erase)	=	yellow
color (OPEN LOOK)	=	Color associated with OL_COLOR 0 in colors.txt file (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on this file.

The statement indicates that the foreground color is red, the background color is white, the erase color is yellow, and the OPEN LOOK color is the color defined for OL_COLOR 0 in the colors.txt file for the display items that follow this command.

Example 2 (Conditional)

```
COLOR FG cyan BG magenta (QUALITY) D200 green yellow red
blue ER black OL 1
```

In this example:

color (foreground)	=	cyan
good_cond_value	=	magenta
required_keyword	=	(QUALITY)
pt_name	=	D200
fair_conf_value	=	green
poor_cond_value	=	yellow
bad_cond_value	=	red
timed_out_cond_value	=	blue
color (erase)	=	black
color (OPEN LOOK)	=	Color associated with OL_COLOR 1 in colors.txt file (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on this file.

The statement indicates that the foreground color is cyan, the erase color is black, and the OPEN LOOK color is the color associated with OL_COLOR 1 in the colors.txt file for the display item(s) that follows this command. The background color changes based on the quality of D200 as described below:

- If D200 has good quality, the foreground color for subsequent items will be magenta.
- If D200 has fair quality, the foreground color for subsequent items will be green.
- If D200 has poor quality, the foreground color for subsequent items will be yellow.
- If D200 has bad quality, the foreground color for subsequent items will be red.
- If D200 has timed out quality, the foreground color for subsequent items will be blue

See Section 2 for more information on conditionals.

Description

The CURSOR command places the drawing cursor at a specific location on the diagram. The CURSOR command is used in conjunction with display items built with relative coordinates to initialize the drawing position (see Section 3-4 for more information on relative coordinates). Subsequent drawings on the diagram will begin at this location.

Syntax

```
CURSOR x y
```

where:

x = x coordinate for the cursor.
Valid range = 0 through 16,383.

y = y coordinate for the cursor.
Valid range = 0 through 16,383.

Rules

1. The CURSOR command may be used in the background, foreground, and trigger sections of the diagram.

Example

```
CURSOR 7878 3232
```

In this example:

x = 7878

y = 3232

The statement defines the x,y position for the cursor as 7878, 3232.

DATE

Description

The DATE command displays the current date on a diagram.

Syntax

```
DATE x y format font_type
```

where:

x = The x coordinate for the baseline position of the first character of the date display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the date display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of first character of the date display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the date display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

format = One of the following date displays:

- 0 - mm/dd/yy
- 1 - mmm dd, yyyy
- 2 - mm/dd

- 3 - mm-dd-yy
- 4 - dd/mmm/yy

where:

- mm = month (01 through 12)
- dd = date (01 through 31)
- yy = year (00 through 99) (the first two s are 19).
- mmm = month (JAN, FEB, MAR, APR, MAY, JUN, JLY, AUG, SEP, OCT, NOV, DEC)

font_type = Font types. The choices are:

- BITMAP (followed by bitmap_font_number).
- BITMAP_OVER (followed by bitmap_font_number).
- VECTOR (followed by w, h, line_width).
- VECTOR_OVER (followed by w, h, line_width).

See Section 3-2 for information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number.
Valid range = 1 through 8.

w = Width of a vector character cell in pixels.
Valid range = 3 through 16,383.

h = Height of a vector character cell in pixels.
Valid range = 3 through 16,383.

line_width = Line width.
Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array.

Rules

1. The x and y coordinates can be relative.
2. The origin will be one of the following:
   - Baseline position of the first character of the text string (for bitmap).
   - Upper left corner of the first character of the text string (for vector).
3. The DATE command may be used in the background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
DATE 4128 3000 0 VECTOR 161 336 1
```

In this example:

x	=	4128
y	=	3000
format	=	0
font_type	=	VECTOR
w	=	161
h	=	336
line_width	=	1

This date uses vector text and will be in the format mm/dd/yy (for example, 11/25/91) as defined by the 0. The upper left corner of the date display will be positioned at 4128, 3000. The width of the date display is 161 and the height is 336. The line width is 1.

Example 2

```
DATE 2752 3397 2 BITMAP 3
```

In this example:

x	=	2752
y	=	3397
format	=	2
font_type	=	BITMAP
bitmap_ font_number	=	3

This date uses bitmap text and will be in the format mm/dd (for example, 11/25) as defined by the 2. The baseline of the first character of the month will be positioned at 2752, 3397. The bitmap font number is 3.

DEF_FKEY_GROUP

Description

The DEF_FKEY_GROUP command defines a set of function keys. There may be up to four such sets defined in a graphic. The purpose of these sets is to override the default function keys defined by the FUNC_KEY command and/or to dynamically change the current set of active function keys. This command is used in conjunction with the LOAD_FKEY_GROUP command to override/change the default function keys.

Syntax

```
DEF_FKEY_GROUP group key1 state n prog1 diag1 m1 args1
                                  prog2 diag2 m2 args2
                                  ...
                                  progn diagn mn argsn

                          key2 state n prog1 diag1 m1 args1
                                  prog2 diag2 m2 args2
                                  ...
                                  progn diagn mn argsn

                                  ...

                          keyn state n prog1 diag1 m1 args1
                                  prog2 diag2 m2 args2
                                  ...
                                  progn diagn mn argsn
```

where:

group	=	Function key group number. Valid range is 1-4.
keyi	=	Function key number. Valid range is 1 -10.
state	=	ON-current function key is active when diagram displays OFF-current function key is inactive when diagram displays.
n	=	Number of application programs to run from current key. Valid range is 1 - 65,535.

progi = Application program number (see Section 4 for valid programs)

diagi = Diagram number. Valid range is 0 - 65,535.

mi = Number of arguments to pass to progi (varies per program)

argsi = List of arguments to pass to progi. Valid argument types are:

- string(80 chars or less)
- integer
- real
- point name/record field
- pointer/offset
- set
- status word

Rules

1. The DEF_FKEY_GROUP command is only valid in the Diagram section of a graphic.
2. There is no display associated with this item.
3. All 10 function keys may be defined in the DEF_FKEY_GROUP command, or a single function key may be defined, or any number of keys between 1 and 10. All of the keys to be in a given function key group must be defined in the same DEF_FKEY_GROUP command. There may be only one DEF_FKEY_GROUP command per function key group in a graphic.

Example

```
DEF_FKEY_GROUP 2
                1 ON 3 117 7001 7 5002 60 22 0 2 \A001Z001\
ID \A001Z002\ ID 66 0 4 1 0 1 "Enter Alarm Status" 6 0 5
\A001Z001\ ID \A001Z002\ ID 1 1 2
                3 ON 1 6 0 5 \A001Z003\ ID \A001Z004\ ID 1 1 2
                6 ON 1 6 0 5 \A001Z005\ ID \A001Z006\ ID 1 1 2
```

In this example, function keys 1,3, and 6 are defined for function key group #2. Function key 3 will run 3 application programs:117,66, and 6. Function keys 3 and 6 will each run 1 application program: 6. Each of the function keys defined by this group are initially active when this group is loaded as the state is ON.

See the **POKE_FLD** (type 7) or the **FUNC_KEY** commands for more examples using application program and argument lists.

Description

The DEF_QUAL command defines the default colors, blink states, and ASCII characters used to display the single character quality flags for process point values. There are five possible qualities to represent:

- Good.
- Fair.
- Poor.
- Bad.
- Timed-out.

For more information on quality, see “Record Types User’s Guide” (U0-0131).

Syntax

```
DEF_QUAL
fgcolor1 bgcolor1 blink1 “string1”
fgcolor2 bgcolor2 blink2 “string2”
fgcolor3 bgcolor3 blink3 “string3”
fgcolor4 bgcolor4 blink4 “string4”
fgcolor5 bgcolor5 blink5 “string5”
```

where:

fgcolor*N*	=	Foreground color name for "stringN"
bgcolor*N*	=	Background color name for "stringN"
blink*N*	=	Blink state. The choices are: • OFF — blink for stringN is off. • ON — blink for stringN is on
"string*N*"	=	One-character ASCII string representing the quality flag, enclosed in double quotes

Note

In all arguments, 1 = good quality; 2 = fair quality; 3 = poor quality; 4 = bad quality; and 5 = timed-out quality.

Rules

1. The DEF_QUAL command can only be used in the DIAGRAM section.
2. Only one DEF_QUAL command may be used per diagram.
3. All of the arguments must be specified.
4. The string parameter must be enclosed in double or single quotes. If a string is not to be displayed for a quality state, a space (" " or ' ') must be used.
5. The DEF_QUAL command is also used in conjunction with the TREND and XY_PLOT commands. When a non-good quality is detected for the trend / xy_plot point, the colors defined in the DEF_QUAL command are used to display the trend/xy_plot. When the trend/xy_plot point has good quality, the color specified in the applicable COLOR command is used to display the trend/ xy_plot.

Example

```
DEF_QUAL green black OFF “G” cyan black OFF “F” yellow black
OFF “P” magenta black OFF “B” red black OFF “T”

FOREGROUND

COLOR FG WHITE

PROCESS_PT 1 1 9 A100 AV
```

In this example:

fgcolor1	=	green
bgcolor1	=	black
blink1	=	OFF
string1	=	“G”
fgcolor2	=	cyan
bgcolor2	=	black
blink2	=	OFF
string2	=	“F”
fgcolor3	=	yellow
bgcolor3	=	black
blink3	=	OFF
string3	=	“P”
fgcolor4	=	magenta
bgcolor4	=	black
blink4	=	OFF
string4	=	“B”
fgcolor5	=	red
bgcolor5	=	black
blink5	=	OFF
string5	=	“T”

This example defines the following:

Point Status	Quality Character
Good	A green on black, non-blinking "G" is displayed in the last position of the process point field.
Fair	A cyan on black, non-blinking "F" is displayed in the last position of the process point field.
Poor	A yellow on black, non-blinking "P" is displayed in the last position of the process point field.
Bad	A magenta on black, non-blinking "B" is displayed in the last position of the process point field.
Timed Out	A red on black, non-blinking "T" is displayed in the last position of the process point field.

In this example, the color of the process point value (white) is not changed, regardless of point quality. To change the color of the process point value, a conditional COLOR statement would be used.

Description

The DIAG_DISP command replaces the current diagram with a new diagram at a specified location on the screen.

Syntax

```
DIAG_DISP diag_type diag_num group_num [POSITION x y]
[SIZE w h] num_sids pt_name1 pt_name2 ... pt_nameN
```

where:

diag_type = Diagram type. The choices are:

- MAIN
- SUBWIN
- WINDOW

See Section 3-3 for more information on diagram types.

diag_num = Diagram number to display.
Valid range = 0 through 65,535.

group_num = Group to display with diagram. Groups are defined in the Group Builder (see "Operator WEStation User's Guide (U0-8100) for more information on the Group Builder). Valid range = 0 through 5,000. Entering 0 does not allow the current group to be changed.

POSITION = ASCII string indicating that x, y will follow (optional, but must be used if x and y are to be defined).

x = x coordinate of upper left corner of diagram (optional; only valid after the ASCII string, POSITION).
Valid range = 0 through 1,151 (entered in screen coordinates; see Section 3-4 for more information on coordinates).

y	=	y coordinate of upper left corner of diagram (optional; only valid after the ASCII string, POSITION). Valid range = 0 through 899 (entered in screen coordinates; see Section 3-4 for more information on coordinates).
SIZE	=	ASCII string indicating that w, h will follow (optional, but must be used if w and h are to be defined).
w	=	Width of diagram to display (optional; only valid after the ASCII string, SIZE). Valid range = 100 through 1,152 (entered in screen coordinates; see Section 3-4 for more information on coordinates).
h	=	Height of diagram to display (optional; only valid after the ASCII string, SIZE). Valid range = 100 through 900 (entered in screen coordinates; see Section 3-4 for more information on coordinates).
num_sids	=	Number of system IDs (points) within the diagram. Valid range = 0 through 99.
pt_name1, pt_name2, pt_nameN	=	Point names of the points to be substituted for $W points in the diagrams (see Section 2 for information on $W pointers).

Rules

1. The specified diagram may be displayed on top of the current diagram or at any other location on the screen.

2. The user can specify the initial size of the diagram to display with this command.

3. The specified diagram's position (x, y) and size (w, h) are optional parameters. If these parameters are not used, the x, y, w, and h of the previously displayed diagram will be used.

4. The DIAG_DISP command may be used in the background, foreground, or trigger sections of the diagram.

Note

If DIAG_DISP is used in the background section of a diagram, it will not update.

5. The system IDs passed to this routine will be used to set the $W points for the system. For example, if 5 points are passed, the user sets $W1, $W2, $W3, $W4, and $W5. Any diagram created with $W points will then use the current system IDs defined for these points. This command will update the current value of $W points within the system if system IDs are passed.

Example

```
DIAG_DISP MAIN 1200 2000 POSITION 100 100 SIZE 500 500 1 A100
```

In this example:

diag_type	=	MAIN
diag_num	=	1200
group_num	=	2000
POSITION	=	POSITION
x	=	100
y	=	100
SIZE	=	SIZE
w	=	500
h	=	500
num_sids	=	1
pt_name1	=	A100

In this example, a new main screen diagram (number 1200) with group number 2000 will display at the x,y position 100, 100. The width is 500 and the height is 500. The system ID, A100, will be used for $W1. Any diagram displayed which uses $W1 will then be referencing A100.

Description

The DOT command displays a dot on the diagram.

Syntax

```
DOT x y size
```

where:

x = x coordinate for dot.
Valid range = 0 through 16,383.

y = y coordinate for dot.
Valid range = 0 through 16,383.

size = Size of the dot. The choices are:

- SMALL
- MEDIUM
- LARGE

See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on dot sizes.

Rules

1. The x and y coordinates can be relative
2. The origin is the x, y position of the dot.
3. The DOT command may be used in the background, foreground, or trigger sections of the diagram.

Example

```
DOT 688 1200 LARGE
```

In this example:

x = 688

y = 1200

size = LARGE

This statement defines a large dot at 688, 1200.

Description

The DYNAMIC_LINE command displays a line within a specified rectangular area which may have dynamic (that is, changing) vertex points. Dynamic vertex points are specified by process point values including $P points. The x,y coordinates making up the line may be fixed or dynamic; each coordinate is independent. All coordinates (fixed or dynamic) are scaled between the low and high limits to fit within the rectangular area.

Syntax

```
DYNAMIC_LINE x y w h x_low x_high y_low y_high line_style
scale update_rate n x1 y1 x2 y2 ... xn yn line_width line_pat
<line_pat_cond>
```

where:

x = x coordinate of lower left corner of boundary area of dynamic line.
Valid range = 0 through 16,383.

y = y coordinate of lower left corner of boundary area of dynamic line.
Valid range = 0 through 16,383.

w = Width of boundary area of dynamic line.
Valid range= 1 through 16,383

h = Height of boundary area of dynamic line.
Valid range = 1 through 16,383

x_low = Low limit used to scale x coordinates.
x_high = High limit used to scale x coordinates.
y_low = Low limit used to scale y coordinates.
y_high = High limit used to scale y coordinates.
Valid values for limits are:

- Integers(-2,147,483,647 ... 2,147,483,647)
- Real constants
- Point names/record fields

- Pointers/offsets

line_style	=	Specifies the line join style for the vertices NOT_FITTED - draws straight line between vertices (same as the LINE command) FITTED - draws curve that best fits vertices
scale	=	Specifies whether x_low,x_high,y_low,y_high values should be displayed or not SCALE - display scales NOSCALE - do not display scales
update_rate	=	How often the dynamic line should be redisplayed (in seconds) Valid range is 1 through 32767.
n	=	Number of x,y coordinate pairs (vertices) Valid range is 2 through 50.
xi, yi	=	Coordinate pairs making up the line. (x and y may be fixed or dynamic; same as limits. See valid values for x_low, x_high, y_low, y_high for xi, yi.)
line_width	=	Line width from GB:Line Widths window to use to draw line. Valid range is 1 through 16.
line_pat	=	Line pattern name
<line_pat_cond>	=	Optional line pattern conditional expression.

Rules

1. The x and y coordinates of the rectangular boundary area of the dynamic line may be relative; the individual vertex points making up the line may not be relative.

2. The origin is the lower left corner of the boundary rectangle of the dynamic line (specified by the **x** and **y** parameters in the source syntax).

3. The DYNAMIC_LINE command may be used in the background, foreground, or trigger sections of the diagram.

4. The line pattern name is a conditional parameter (the user may specify a conditional expression to determine what line pattern is used to draw the line).

5. If the *scale* parameter is set to SCALE, the x_low, x_high, y_low, and y_high parameters are displayed at predefined locations and with predefined text attributes (size and style). This is NOT configurable. If locations or attributes other than the defaults are desired, set the *scale* parameter to NOSCALE and display the scale values independently of the DYNAMIC_LINE command using the TEXT or PROCESS_PT commands.

6. The update_rate parameter overrides the overall diagram update rate. The update_rate for the dynamic line **cannot be less** than the overall diagram update rate (although this is not reported as an error, the dynamic line update rate will default to the diagram update rate).

7. It is possible to calculate vertex points to be used in the dynamic line at runtime by storing the calculated values in the scratch pad area (accessed by $P pointers) and using $P points to specify the **xi**, **yi** vertex points. The MATH command or some application program may be used to calculate the values and store them in the scratch pad area.

8. The dynamic line will not update unless it is in the FOREGROUND diagram section, despite the update_rate parameter.

Examples

Example 1

```
DYNAMIC_LINE 3604 6318 3770 2412 0 100 25 50 NOT_FITTED
NOSCALE 2 2 \A100\ AV 50 \A200\ AV \A300\ AV 1 solid
```

In this example:

x	=	3604
y	=	6318
w	=	3770
h	=	2412
x_low	=	0
x_high	=	100
y_low	=	25
y_high	=	50
line_style	=	NOT_FITTED
scale	=	NOSCALE
update_rate	=	2
n	=	2
x1,y1	=	\A100\ AV, 50
x2,y2	=	\A200\ AV, \A300\ AV
line_width	=	1
line_pat	=	solid

This statement defines a dynamic line with 2 vertices. The origin of the dynamic line is at 3604,6318. The dynamic line occupies a width of 3770 virtual pixels, and a height of 2412 virtual pixels. The x coordinate of the first vertex is dynamic, as are the x and y coordinates of the second vertex. The y coordinate of the first vertex is fixed. The x coordinates will be scaled between 1 and 100; the y coordinates will be scaled between 25 and 50. The vertex points will be joined by a solid straight line of line width 1. No scale values will be displayed by this command. The dynamic line will update every 2 seconds (assuming the overall diagram update rate is <= 2).

Example 2

```
DYNAMIC_LINE 3604 6318 3770 2412 \A100\ LL \A100\ HL \A200\
LL \A200\ HL FITTED SCALE 5 4 12.45 \A200\ AV \A100\ HL 0 20
\A200\ AV 100 \A200\ HL 3 dashed (testai01 av > 6.5) dotted
```

In this example:

x	=	3604
y	=	6318
w	=	3770
h	=	2412
x_low	=	\A100\ LL
x_high	=	\A100\ HL
y_low	=	\A200\ LL
y_high	=	\A200\ HL
line_style	=	FITTED
scale	=	SCALE
update_rate	=	5
n	=	4
x1,y1	=	12.45, \A200\ AV
x2,y2	=	\A100\ HL, 0
x3,y3	=	20, \A200\ AV
x4,y4	=	100, \A200\ HL
line_width	=	3
line_pat	=	dashed
<line_pat_cond>	=	(testai01 av > 6.5) dotted

This statement displays a dynamic line with four vertices. A best fit curve is drawn to join the vertices as opposed to a normal line. Scale values are displayed. The curve is drawn with line width #3 from the GB:Line Widths window, in either the dashed or dotted line pattern. The line pattern is dotted if the AV field of testai01 is greater than 6.5, and dashed otherwise. Assuming that the diagram update rate is <= 5, the dynamic line will be only be updated every 5 seconds. The x coordinates of the vertices are scaled between the low and high limits of \A100\; the y coordinates are scaled between the low and high limits of \A200\. Each of the four vertices have one coordinate fixed, and one dynamic.

EF_STATE

Description

The EF_STATE command is used in conjunction with the ENTRY_FLD command to activate/deactivate an entry field (see ENTRY_FLD command). When an entry field is initially created, it takes a state parameter which defines whether it is active/inactive when the diagram is displayed. THe EF_STATE command can change the state of an entry field while the diagram is displayed. An entry field can be activated/deactivated when some condition is met and/or from a trigger.

Syntax

```
EF_STATE entry_fld state
```

where:

entry_fld	=	Entry field number. Valid range = 1 through 254.
state	=	ON =entry field active OFF =entry field not active

Rules

1. The EF_STATE command may be used in the background, foreground, or trigger sections of the diagram.

Example

```
FOREGROUND
IF (testai01 av > 0.5)
   EF_STATE 9 ON
END_IF

KEYBOARD
ENTRY_FIELD 3440 4200 10 9 OPERATOR ASCII OFF VECTOR 321 734 3
```

In this example:

entry_fld = 9

state = ON

This statement activates entry field #9 if the analog value field (AV) of testai01 becomes greater than 0.5. The entry field is initially inactive (state set to OFF in ENTRY_FLD command).

ELLIPSE

Description

The ELLIPSE command displays an ellipse given a center, a horizontal radius, and a vertical radius.

Syntax

```
ELLIPSE cx cy hr vr line_width line_pat <line_conditional_
expression> fill_pat <fill_conditional_expression>
```

where:

cx	=	x coordinate of the center of the ellipse. Valid range = 0 through 16,383.
cy	=	y coordinate of the center of the ellipse. Valid range = 0 through 16,383.
hr	=	Horizontal radius. Valid range = (hr > 0, cx + hr <= 16,383, cx - hr >= 0)
vr	=	Vertical radius. Valid range = (vr > 0, cy + vr <= 16,383, cy - vr >= 0)
line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.
line_pat	=	Line pattern name.
<line_conditional_ expression>	=	Optional conditional for line pattern.
fill_pat	=	Fill pattern name.
<fill_conditional_ expression>	=	Optional conditional for fill pattern.

Note

See "WEStation Graphics Builder User's Guide" (U0-8210) for information on defining line patterns and fill patterns.

Rules

1. The x and y coordinates can be relative.
2. The origin is the center of the ellipse.
3. The line and fill pattern names are conditional parameters (the user may specify conditional expressions to determine what line pattern or fill pattern is used to display the ellipse).
4. The ELLIPSE command may be used in the background, foreground, or trigger sections of a diagram.

Examples

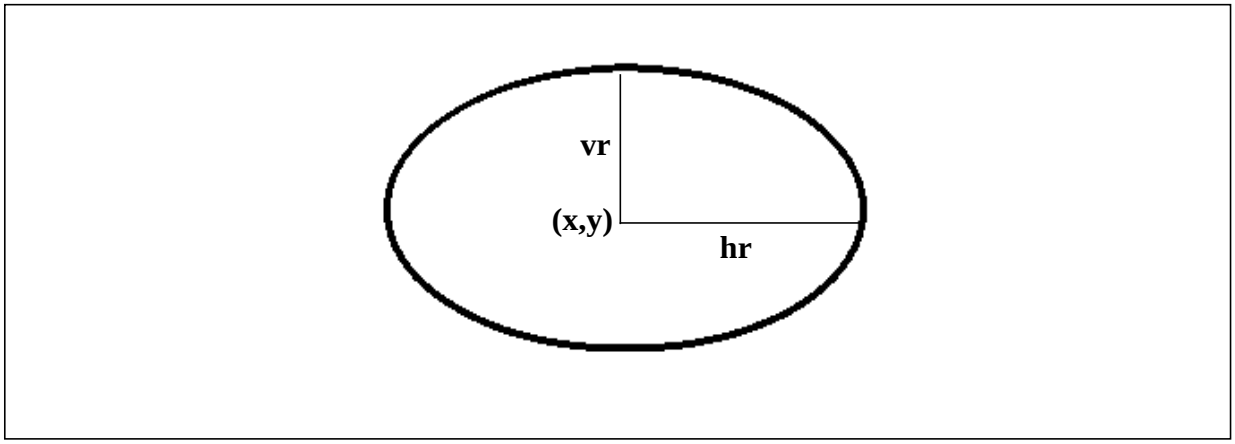
Example 1

```
ELLIPSE 9089 6245 3563 4524 3 solid unfilled
```

In this example:

cx	=	9089
cy	=	6245
hr	=	3563
vr	=	4524
line_width	=	3
line_pat	=	solid
fill_pat	=	unfilled

This statement defines an ellipse with a center at 9089, 6245. The horizontal radius is at 3563, and the vertical radius is at 4524. The line width is 3, and the line pattern is solid. The ellipse is not filled. See the figure below.



Example 2 (Conditional)

```
ELLIPSE 1093 6500 292 572 4 dashed {(A100 AV > 100) dotted
(LOWALARM = A200 AS) solid} unfilled
```

In this example:

cx	=	1093
cy	=	6500
hr	=	292
vr	=	572
line_width	=	4
line_pat	=	dashed
conditional _ expression	=	{(A100 AV > 100) dotted (LOWALARM = A200 AS) solid}
fill_pat	=	unfilled

This statement defines an ellipse with a center at 1093, 6500. The horizontal radius is at 292, and the vertical radius is at 572. The line width is 4.

A line pattern conditional is specified. If A100 AV > 100, the line pattern will be dotted. If the first conditional evaluates to false, the second conditional will be considered. If LOWALARM = A200 AS, the line pattern will be solid. If the second conditional evaluates to false, the line pattern will be dashed (default). The fill pattern is unfilled.

See Section 2 for more information on conditionals.

ENTRY_FLD

Description

The ENTRY_FLD command defines an area of the diagram for adding information that is entered by an operator and/or obtained from a graphics application program.

Syntax

```
ENTRY_FLD x y l number type format state font_type
```

where:

x = The x coordinate for the baseline position of the first character of the entry field display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the entry field display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of the first character of the entry field display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the entry field display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

l = Maximum number of characters that can be entered in the entry field.
Valid range = 1 through 80.

number = Number associated with the entry field (buffer number of where the entered data will be stored at the Operator WEStation).
Valid range = 1 through 254.

type = Defines the "writer" for an entry field. The choices are:

- OPERATOR = Allows an operator to enter data in the entry field, but not an application program.
- PROGRAM = Allows an application program to enter data in the entry field, but not an operator.
- BOTH = Allows either an operator or an application program to enter data in the entry field.

format = Format of the data to be entered in the entry field. The choices are:

- ASCII
- INT (integer)
- REAL
- BYTE
- HEX (hexadecimal)
- EXPONENTIAL

state = State. The choices are:

- ON = Entry field will be active when the diagram is initially displayed.
- OFF = Entry field will be inactive when the diagram is initially displayed.

font_type = Font types. The choices are:

- BITMAP (followed by bitmap_font_number)
- BITMAP_OVER (followed by bitmap_font_number)
- VECTOR (followed by w, h, line_width)
- VECTOR_OVER (followed by w, h, line_width)

See Section 3-2 for information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number.
Valid range = 1 through 8.

w = Width of a vector character cell in pixels.
Valid range = 3 through 16,383.

h = Height of a vector character cell in pixels.
Valid range = 3 through 16,383.

line_width = Line width.
Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

Rules

1. There may be a maximum of 254 entry fields per diagram.
2. OPERATOR accepts only alphanumeric inputs from the keyboard. PROGRAM accepts only data from the Operator WEStation application programs. BOTH accepts keyboard input and application program data.
3. The ENTRY_FLD command can only be used in the keyboard section of a diagram.

Examples

Example 1

```
ENTRY_FLD 3440 4200 10 9 PROGRAM ASCII OFF VECTOR 321 734 3
```

In this example:

x	=	3440
y	=	4200
l	=	10
number	=	9
type	=	PROGRAM
format	=	ASCII
state	=	OFF
font_type	=	VECTOR
w	=	321
h	=	734
line_width	=	3

This statement defines an entry field at 3440, 4200. The maximum number of characters that can be entered into this entry field is 10. The number associated with this entry field is 9. Only a graphics application program can add information to this field. The information will be entered in ASCII format. The entry field will not be activated (state = off) when the diagram initially displays. The font type used will be vector text. The width and height of each character cell will be 321, 734 respectively. The line width is 3.

Example 2

```
ENTRY_FLD 13072 8400 25 1 BOTH HEX ON BITMAP 5
```

In this example:

x	=	13072
y	=	8400
l	=	25
number	=	1
type	=	BOTH
format	=	HEX
state	=	ON
font_type	=	BITMAP
bitmap_font_number	=	5

This statement defines an entry field at 13072, 8400. The maximum number of characters that can be entered into this entry field is 25. The number associated with this entry field is 1. Both a graphics application program and the operator can add information to this field. The information will be entered in hexadecimal format. The entry field will be activated (on) when the diagram initially displays. The font type used is bitmap. The bitmap font number is 5.

Description

The FKEY_STATE command is used in conjunction with the FUNC_KEY, DEF_FKEY_GROUP, and LOAD_FKEY_GROUP commands to activate/ inactivate a function key (see these commands for details). When a function key is initially created, it takes a *state* parameter which defines whether it is active/ inactive when the diagram is displayed. The FKEY_STATE command can change the state of a function key while the diagram is displayed. A function key can be activated/deactivated when some condition is met and/or from a trigger.

Syntax

```
FKEY_STATE key state
```

where:

key	=	Function key number. valid range is 1-10.
state	=	ON - function key is active OFF - function key is inactive

Rule

1. The FKEY_STATE command may be used in the background, foreground, or trigger sections of the diagram.

Example

```
FKEY_STATE 1 ON
```

In this example:

key = 1

state = ON

This statement activates function key 1.

Description

The FORCE_UPDATE command is used in the foreground section of a diagram to control whether or not display items are redrawn if their current value/attributes have not changed since the last update. Items in the foreground are always redrawn if their current value/attributes have changed; items are not typically redrawn if their current value/attributes remain the same. Sometimes it is necessary (in the case of overlapping items for example) to redraw one or more items if some underlying item is redrawn even if the top items haven't changed. This command, when the state is turned on, will force all subsequent items in the foreground section to be redrawn whether their value/attributes change or not. This remains in effect until a subsequent FORCE_UPDATE command is processed which turns the state off.

Syntax

```
FORCE_UPDATE state
```

where:

state	=	TRUE/ON - henceforth all items will be redrawn FALSE/OFF -henceforth only items whose value/ attributes change will be redrawn

Rules

1. The FORCE_UPDATE command may only be used in the foreground section of the diagram.
2. There is no display associated with this command.
3. This command overrides the default state of the foreground section. The default state is FALSE/OFF.
4. TRUE and ON are interchangeable; FALSE and OFF are interchangeable.

5. When the state is turned ON, the state remains on until it is turned off by another FORCE_UPDATE command. Note that the state does not get reset the next time the foreground updates - it will remain on as long as the diagram is displayed.

Example

```
FORCE_UPDATE TRUE
```

In this example:

state = TRUE

This statement causes all display items after it in the foreground section to be redrawn whether their current value/attributes change or not.

Description

The FUNC_KEY command is used in conjunction with the membrane keyboard (see " Operator WEStation User's Guide" (U0-8100), Using the Membrane Keyboard) to run one or more application programs from one of the eight programmable function keys(P1 through P8). This command defines which application program(s) run when a given function key is pressed. It is similar in function to a poke type #7 (see POKE_FLD command).

The association of the application program(s) to the given function key is local to the graphic in which the FUNC_KEY command exists. When a function key on the membrane keyboard is pressed, the application programs defined by the FUNC_KEY command in the graphic currently active for keyboard input are executed. The graphic currently active for keyboard input on the WEStation has a check in the **Select** box in the upper left corner. If no graphic is currently active for keyboard input, or if the graphic does not contain an applicable FUNC_KEY command, then no application programs run when the function key is pressed.

The currently active function keys defined in a graphic may be dynamically changed by using the DEF_FKEY_GROUP and the LOAD_FKEY_GROUP commands (see these commands for details).

Syntax

```
FUNC_KEY key state n prog1 diag1 m1 args1
                     prog2 diag2 m2 args2
                     ...
                     progn diagn mn argsn
```

where:

key	=	Function key number. Valid range is 1 -8.
state	=	ON - key is active when diagram initially displays OFF - key is inactive when diagram initially displays
n	=	Number of application programs to run. Valid range is 1 - 65,535.
progi	=	Application program number (see Section 4 for valid programs)

diagi = Diagram number. Valid range is 0 - 65,535.

mi = Number of arguments to pass to progi (varies per program)

argsi = List of arguments to pass to progi. Valid argument types are:

- string(80 chars or less)
- integer
- real
- point name/record field
- pointer/offset
- set
- status word

Rules

1. The FUNC_KEY command may only be used in the keyboard section of the diagram.
2. There is no display associated with this command.
3. Function keys 9,10, and 11 are defined, but are for Westinghouse internal use only. Using these keys will not generate an error, but they cannot be implemented by a customer as there are only 8 keys on the membrane keyboard.

Example

```
FUNC_KEY 1 ON 3 117 7001 7 5002 60 22 0 2 \A001Z001\ ID
\A001Z002\ ID
             66 0 4 1 0 1 "Enter Alarm Status"
             6 0 5 \A001Z001\ ID \A001Z002\ ID 1 1 2
```

In this example:

key	=	1
state	=	ON
n	=	3
prog1	=	117
diag1	=	7001
m1	=	7
args1	=	5002,60,22,0,2,\A001Z001\ ID,\A001Z002\ ID
prog2	=	66
diag2	=	0
m2	=	4
args2	=	1,0,1,"Enter Alarm Status"
prog3	=	6
diag3	=	0
m3	=	5
args3	=	\A001Z001\ ID,\A001Z002\ ID, 1,1, 2

This statement associates three application programs with function key number 1. When the P1 key is pressed on the membrane keyboard, and the graphic containing this FUNC_KEY command is selected for keyboard input, diagram 7001 will pop-up, "Enter Alarm Status" will be written to entry field #1 on that diagram, and control will be enabled for algorithms A001Z001 and A001Z002.

GCODE

Description

The GCODE command is used to call an internal function.

Syntax

```
GCODE n arg_list
```

where:

n	=	Number of arguments to follow. Valid range = 1 through 255.
arg_list	=	List of arguments (integer, real, process point and record field pairs, or text strings) delimited by spaces; arg1 arg2... argn

Rules

1. The GCODE command is reserved for Westinghouse use only.

Description

The GTEXT command displays one of the three group text strings defined in the current group. Group text strings are defined in the current group library by the Group Builder. See "Operator WEStation User's Guide" (U0-8100) for more information on the Group Builder.

Syntax

```
GTEXT x y string_number number_of_chars orientation
font_type
```

where:

x = The x coordinate for the baseline position of the first character of the group text display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the group text display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of first character of the group text display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the group text display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

string_number = The group text string to display.
Valid range = 1 through 3.

number_of_chars = Number of characters of the group text string to display.
Valid range = 1 through 80.

orientation = Orientation of group text. The choices are:

- HORZ (horizontal)
- VERT (vertical)

font_type = Font type. The choices are:

- BITMAP (followed by bitmap_font_number).
- BITMAP_OVER (followed by bitmap_font_number).
- VECTOR (followed by w, h, line_width).
- VECTOR_OVER (followed by w, h, line_width).

See Section 3-2 for information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number.
Valid range = 1 through 8.

w = Width of a vector character cell in pixels.
Valid range = 3 through 16,383.

h = Height of a vector character cell in pixels.
Valid range = 3 through 16,383.

line_width = Line width.
Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

Rules

1. The x and y coordinates can be relative.
2. The origin can be one of the following:
   - Baseline position of the first character of the text string (for bitmap).
   - Upper left corner of the first character of the text string (for vector).
3. The GTEXT command can be used in the background, foreground, or trigger sections of a diagram.

Example

```
GTEXT 2064 6600 1 20 HORZ VECTOR 161 336 1
```

In this example:

x	=	2064
y	=	6600
string_number	=	1
number_of_chars	=	20
orientation	=	HORZ (horizontal)
font_type	=	VECTOR
w	=	161
h	=	336
line_width	=	1

This statement displays the first group text string from the group currently loaded at position 2064, 6600. There are 20 characters in group text string. The string displays horizontally in vector text. The width and height of the string are 161, 336. The line width is 1.

IF_CHANGED/ENDIF

Description

The IF_CHANGED/ENDIF commands operate as a pair. These commands are used to define a set of graphic commands that are executed whenever one of the operands change in value. When the user enters the IF_CHANGED command in the source editor, a corresponding ENDIF command is automatically displayed on the next line.

Syntax

```
IF_CHANGED deadband operand1 operand2 ... operandN

.
.

ENDIF
```

where:

deadband	=	Tolerance (only used for analog points; deadband is used in conjunction with other alarm limits to prevent a point from oscillating into and out of alarm when the current value is hovering around the value of a limit). For more information on deadband, see "Record Types User's Guide" (U0-0131). This must be a real number (for example, 0.3, 3.0, 3,6, etc.). For all other types of operands, enter 0.0. Note that real numbers are required to have a digit before AND after the decimal point!
operand1, operand2, operand*N*	=	Up to 50 operands can be defined. Valid operands are: Point name (point name, point name and record field, pointer name, pointer name and offset). Status word (see Appendix B for more information on status words). Set values. Valid range = 1 through 255.

Rules

1. A status word cannot be the first operand.
2. If a status word is used, the previous operand must be a valid system ID. That is, the previous point is checked for that quality.
3. When a status word operand is used in conjunction with the IF_CHANGED command, the test for change is based on the bits set in the status word mask (see Appendix B for more information on status word masks). If any of the bits set in the status word mask change, the commands within the IF_CHANGED loop will execute. Changes in the bits NOT set in the status word mask will not cause the IF_CHANGED commands to execute.
4. All commands following the IF_CHANGED command up to the matching ENDIF command are executed if any value changes, initially at start-up, and whenever a refresh of the window is made.
5. The IF_CHANGED/ENDIF commands can be used in the background, foreground, or trigger sections of a diagram.

Examples

Example 1

```
IF_CHANGED 3.3 A100 AV

COLOR FG RED BG WHITE

RECTANGLE 3342 2504 6534 8141 3 solid unfilled

ENDIF
```

In this example:

deadband = 3.3

operand1 = A100 AV

This statement indicates that if the current value of A100 AV has changed more than 3.3 analog values from the last time the diagram displayed, all of the statements within the IF_CHANGED/ENDIF loop (color and rectangle in this example) will execute. See the COLOR and RECTANGLE reference pages for more information on these commands.

Example 2

```
IF_CHANGED 0.0 A100 A2 ON11

COLOR FG RED BG WHITE

RECTANGLE 3342 2504 65 34 8141 2 dot_dashed unfilled

ENDIF
```

In this example:

deadband = 0.0

operand1 = A100 A2

operand2 = ON11

This statement indicates that if bit 11 of A100 A2 changes from the last time the diagram displayed, the commands within the IF_CHANGED/ENDIF loop (color and rectangle in this example) will execute. In this example, OFF11 could have been used in place of ON11 with the same result as ON11 and OFF11 have identical masks (see Appendix B for more information on status word masks).

Description

The IF/ENDIF commands operate as a pair. These commands are used to define a set of graphics commands that are executed when a condition is met. When the user enters the IF command in the source editor, a corresponding ENDIF command is automatically displayed on the next line.

Syntax

```
IF (conditional_expression)

.
.

ENDIF
```

where:

(conditional_expression) = Simple conditional expression.

Rules

1. All commands following the IF command up to the matching ENDIF command are executed if the condition is met.
2. COLOR and BLINK commands should be used within the IF/ENDIF command to ensure appropriate display of color and blink for the display items within IF/ENDIF.
3. Each conditional expression must be enclosed in parentheses.
4. The IF/ENDIF commands can be used in the background, foreground, or trigger sections of a diagram.

Example

```
IF (D5BLR01 < 100)

COLOR FG green BG white

BLINK FG OFF BG OFF

TEXT 7696 4784 "IN NORMAL RANGE" HORZ VECTOR 167 815 1

ENDIF

IF (D5BLR01 >= 100)

COLOR FG red BG white

BLINK FG OFF BG OFF

TEXT 8177 5980 "IN ALARM" HORZ VECTOR 242 1970 1

ENDIF
```

In this example:

conditional_expression	=	(D5BLR01 < 100)
commands executed if condition is met	=	COLOR FG green BG white BLINK FG OFF BG OFF TEXT 7696 4784 "IN NORMAL RANGE" HORZ VECTOR 167 815 1
conditional_expression	=	(D5BLR01 >= 100)
commands executed if condition is met	=	COLOR FG red BG white BLINK FG OFF BG OFF TEXT 8177 5980 "IN ALARM" HORZ VECTOR 242 1970 1

These statements describe the conditions that will be met depending on the value of D5BLR01. If D5BLR01 < 100, the green text "IN NORMAL RANGE" will display. If D5BLR01 >= 100, the red text "IN ALARM' will display. (See COLOR, BLINK, and TEXT command reference pages for more information.)

Description

The IFELSE/ELSE/ENDIF commands operate as a trio. These commands are used to define a set of graphic commands that are executed when a condition is met and another set of graphic commands that are executed when the same condition is not met. When the user enters the IFELSE command in the source editor, the corresponding ELSE/ENDIF commands are automatically displayed on the next lines.

Syntax

```
IFELSE (conditional_expression)

.
.

ELSE

.
.

ENDIF
```

where:

(conditional_expression) = Simple conditional expression.

IFELSE/ELSE/ENDIF
Cont'd

Rules

1. All commands following the IFELSE command up to the ELSE command are executed if the condition is met, and all commands following the ELSE command up to the ENDIF command are executed if the same condition is not met.

2. COLOR and BLINK commands should be used within the IFELSE/ELSE/ ENDIF command to ensure appropriate display of color and blink for display items within IFELSE/ELSE/ENDIF.

3. Each conditional expression must be enclosed in parentheses.

4. The IFELSE/ELSE/ENDIF commands can be used in the background, foreground, or trigger sections of a diagram.

Example

```
IFELSE (A1001 > $p2)

COLOR FG magenta BG white

BLINK FG OFF BG OFF

CIRCLE 6192 3600 344 3 solid unfilled

ELSE

COLOR FG CYAN BG WHITE

BLINK FG OFF BG OFF

ELLIPSE 13072 4200 1313 1378 5 solid back_slash

ENDIF
```

In this example:

conditional_expression	=	(A1001 > $P2)
command executed if condition is met	=	COLOR FG magenta BG white BLINK FG OFF BG OFF CIRCLE 6192 3600 344 3 solid unfilled
command executed if condition is not met	=	COLOR FG cyan BG white BLINK FG OFF BG OFF ELLIPSE 13072 4200 1313 1378 5 solid unfilled

These statements describe the commands that will be executed depending on the value of A1001. If A1001 > $P2, a magenta circle will display. If A1001 is <= $P2, a cyan ellipse will display. See the COLOR, BLINK, CIRCLE, and ELLIPSE command reference pages for more information on these commands.

LINE

Description

The LINE command draws a line defined by a set of coordinate pairs.

Syntax

```
LINE x1 y1 x2 y2 ... xn yn line_width line_pat
<line_conditional_expression>
```

where:

x1	=	x coordinate for start of line. Valid range = 0 through 16383.
y1	=	y coordinate for start of line. Valid range = 0 through 16383.
x2	=	Next x coordinate.
y2	=	Next y coordinate.
x*n*	=	x coordinate for end of line. The maximum number of points per line is 255 (1 < *n* <= 255).
y*n*	=	y coordinate for end of line. The maximum number of points per line is 255 (1 < *n* <= 255).
line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.
line_pat	=	Line pattern name.
<line_conditional_ expression>	=	Optional conditional for line pattern.

Note

See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on defining line patterns.

Rules

1. The x and y coordinates can be relative.
2. The line's origin is the first vertex (x, y) of the line.
3. The line pattern name is a conditional parameter (the user may specify a conditional to determine what line pattern is used).
4. The LINE command may be used in the background, foreground, or trigger sections of a diagram.

Examples

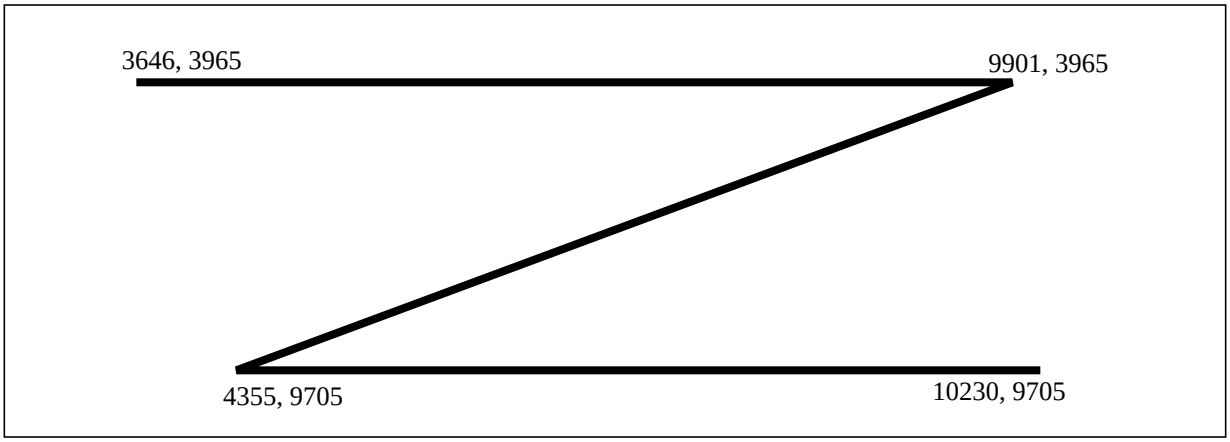
Example 1

```
LINE 3646 3965 9901 3965 4355 9705 10230 9705 3 solid
```

In this example:

x1	=	3646
y1	=	3965
x2	=	9901
y2	=	3965
x3	=	4355
y3	=	9705
x4	=	10230
y4	=	9705
line_width	=	3
line_pat	=	solid

This statement defines a line with a beginning point of 3646, 3965. The second vertex of the line is at 9901, 3965. The third vertex of the line is at 4355, 9705. The ending point of the line is at 10230, 9705. The line width is 3, and the line pattern is dashed. See figure below.



Example 2 (Conditional)

```
LINE 1443 10166 3367 1196 5291 10166 1443 10166 1 solid
(CASE) A5514 HL 1 1 2 near_solid dashed
```

In this example:

x1	=	1443
y1	=	10166
x2	=	3367
y2	=	1196
x3	=	5291
y3	=	10166
x4	=	1443
y4	=	10166
line_width	=	1
line_pat	=	solid
conditional_expression	=	(CASE) A5514 HL 1 1 2 near_solid dashed

This statement defines a line with a beginning point of 1443, 10166. The second vertex of the line is at 3367, 1196. The third vertex of the line is at 5291, 10166. The ending point of the line is at 1443, 10166. The line width is 1.

A line pattern conditional is used in this example. The line pattern for the line will change based on the value of A5514 HL as described below:

- If A5514 HL < 2, the fill pattern will be solid.
- If A5514 HL >= 2 and if A5514 HL < 3, the line pattern will be near_solid.
- If A5514 HL >= 3 and if A5514 HL < 4, the line pattern will be dashed.
- If A5514 HL >= 4, the fill pattern will be solid.

See Section 2 for more information on conditionals.

LOAD_FKEY_GROUP

Description

The LOAD_FKEY_GROUP command is used in conjunction with the DEF_FKEY_GROUP command to activate a new set of function keys. The DEF_FKEY_GROUP command defines the set of function keys and associates them with a group tag number (1-4). The LOAD_FKEY_GROUP command then activates the set of function keys referenced by the given group tag number. The current function keys defined are replaced with the given group. That is, if function keys 1-10 are currently defined, and fkey group #2 defines function keys 1-5, only function keys 1-5 will be active after the LOAD_FKEY_GROUP command for group #2.

Syntax

```
LOAD_FKEY_GROUP fkey_group
```

where:

fkey_group	=	Function key group number. valid range is 1-4.

Rules

1. The LOAD_FKEY_GROUP command may be used in the background, foreground, or trigger sections of the diagram.

2. A DEF_FKEY_GROUP command should exist in the graphic which defines the respective fkey group before this command is used. If an associated DEF_FKEY_GROUP command does not exist, then no function keys are defined for that group. Thus, that group is loaded, no function keys will be active. No error/warning is generated by GB or by the Process Diagram run-time code.

Example

```
LOAD_FKEY_GROUP 2
```

This example activates the function keys defined by fkey group #2. A separate DEF_FKEY_GROUP command must exist in the graphic to define fkey group #2, or no function keys will be active.

LOOP/ENDLOOP

Description

The LOOP/ENDLOOP commands operate as a pair. These commands are used to define a set of graphic commands that are executed a given number of times. When the user enters the LOOP command in the source editor, a corresponding ENDLOOP command is automatically displayed on the next line.

Syntax

```
LOOP n

.
.

ENDLOOP
```

where:

n = Number times to execute the loop (1 < n < 32,767)

Rules

1. The LOOP/ENDLOOP commands may be used in the background, foreground, or trigger sections of a diagram.

Examples

Example 1

```
LOOP 5

COLOR FG BLUE BG WHITE BLINK FG OFF BG OFF

LINE [12558] [11400] 8755 10400 3 solid

ENDLOOP
```

In this example:

n = 5

This example defines 5 blue lines.

Example 2

```
POINTER $P1 9 0

LOOP 10

   COLOR FG GREEN BG WHITE

   PROCESS_PT [9620] [1794] 10 3 RIGHT0 ON HORZ VECTOR 321 734
   1 $P1 $I0 -5 10

   PROCESS_PT [1234] [1234] 10 3 RIGHT ON VERT BITMAP 3 $P1 $R4
   0 100

   POINTER $P1 0 +8

ENDLOOP
```

In this example:

n = 10

The loop is executed 10 times. During the first loop, $P1 $I0 points to the integer value at byte offset 0, and $P1 $R4 points to the real value at byte offset 4. The POINTER statement in the loop increments the byte offset of $P1 by 8 bytes (integer value = 4 bytes and real value = 4 bytes). Therefore, during the second loop, $P1 $I0 points to the integer value at byte offset 8, and $P1 $R4 points to the real value at byte offset 12.

See the POINTER and PROCESS_PT reference pages for more information on these commands. See Section 2 for more information on pointers and offsets.

Rules

1. Macro graphic files must be named in the macro directory by the following convention:

   macro*N* (where N is the macro number).

2. Macros are built within the Graphics Builder and can contain all of the graphics commands defined in the graphics language (see "WEStation Graphics Builder User's Guide (U0-8210) for more information on using macros).

3. The macro parameters are:

   - A list of process points that will be substituted for occurrences of $D points in the macro graphic.

     For example, if the user specifies five process point names to be substituted, the first name specified is substituted for all occurrences of $D1 point parameters in any gcode, the second name is substituted for all occurrences of $D2 point parameters, etc., until the process point names have all been substituted or no more occurrences of $D points exist in the macro.

   - A list of up to 50 foreground text strings and a list of up to 50 background text strings that will be substituted for occurrences of text strings within the TEXT/MULTI_TEXT commands in the foreground or background sections of the macro. The text string size limit is 80 characters. As MULTI_TEXT strings are limited to 30 characters, substitution strings will be truncated to 30 characters if more than 30 characters are entered.

     If the user specifies five foreground text strings to substitute, the first string specified is substituted for all occurrences of any "$T1" text string within a TEXT/MULTI_TEXT command in the foreground. The second string is substituted for all occurrences of any "$T2" text string within a TEXT/MULTI_TEXT command, etc., until the text strings have all been substituted or no more occurrences of $T text strings exist in the macro. The same process occurs for background text strings.

4. Once a macro has been created, the entire macro diagram should be interactively grouped and moved to position 0, 0 before saving. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on grouping and moving items.

Example

```
MACRO 5 1234 15242 2 1 0 A100 A250 "ALARM STATUS"
```

In this example:

macro_number	=	5
x	=	1234
y	=	15242
num_pt_par	=	2
num_fg_tx	=	1
num_bg_tx	=	0
pt_par_list	=	A100
		A250
fg_tx_list	=	"ALARM STATUS"
bg_tx_list	=	Not used in this example

This statement indicates that macro 5 displays at x,y position 1234, 15242. Two point parameters, A100 and A250, will replace the $D1 and $D2 pointers ($D pointers can be specified in any command that accepts these types of pointers). One foreground text string, "ALARM STATUS" will be substituted for $T1 (specified in TEXT/MULTI_TEXT statements). There are no background text strings to substitute.

MACRO

Description

Macros are graphic files that contain display items and logic which will be repeated throughout other graphic files. Macros automate the tedious and repetitive tasks that would otherwise require the user to continually redraw the same diagram or diagram section.

The MACRO command is used to integrate a separate diagram into the main diagram. Macro files reside in a configurable directory specified by the WDPF_MACRODIR environment variable (the default macro directory is $WDPF_HOME/mmi/diag). See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on the macro directory.

Syntax

```
MACRO macro_num x y wscale hscale num_pts num_other_tx
num_bg_tx num_sets num_consts num_statuses pt_list
other_tx_list bg_tx_list set_list const_list status_list
```

where:

macro_ number	=	Macro number. Valid range = 0 through 65,535.
x	=	x coordinate at which to place upper left corner of macro. Valid range = 0 through 16,383.
y	=	y coordinate at which to place upper left corner of macro. Valid range = 0 through 16,383.
wscale	=	Width scale factor for sizing the macro (positive real number). 1.0 is the as-built width of the macro.
hscale	=	Height scale factor for sizing the macro (positive real number). 1.0 is the as-built height of the macro.

num_pts	=	Number of point parameters to substitute for $D points in the macro file. Valid range = 0 through 99.
num_other_tx	=	Number of text strings to substitute for $T text in the foreground, triggers, or keyboard sections of the macro file. Valid range = 0 through 50.
num_bg_tx	=	Number of text strings to substitute for $T text in the background section of the macro file. Valid range = 0 through 50.
num_sets	=	Number of set numbers to substitute for $SET arguments in the macro file. Valid range = 0 through 256.
num_consts	=	Number of integer/real constants to substitute for $CONST arguments in the macro file. Valid range is 0 through 256.
num_statuses	=	Number of status words to substitute for $STATUS arguments in the macro file. Valid range is 0 through 256.
pts_list	=	List of point parameters (no record fields!) if num_pts > 0.
other_tx_list	=	List of quoted strings if num_other_tx > 0.
bg_tx_list	=	List of quoted strings if num_bg_tx > 0.
set_list	=	List of set numbers if num_sets > 0.
const_list	=	List of int/real constants if num_consts > 0.
status_list	=	List of status words if num_statuses > 0.

Rules

1. Macro graphic files must be named in the macro directory by the following convention:

   macro*N*.src, macro*N*.diag (where N is the macro number).

2. Macros are built within the Graphics Builder and can contain all of the graphics commands defined in the graphics language (see "WEStation Graphics Builder User's Guide (U0-8210) for more information on using macros).

3. The macro parameters which can be passed and substituted in the macro are:

   - **A list of process points** that will be substituted for occurrences of $D points in the macro graphic. This is a list of point names only - not point name/ record field pairs. For example, if the user specifies five process point names to be substituted, the first name specified is substituted for all occurrences of $D1 point parameters in any gcode, the second name is substituted for all occurrences of $D2 point parameters, etc., until the process point names have all been substituted or no more occurrences of $D points exist in the macro.

   - **A list of up to 50 text strings** to be substituted for $T strings in the specified commands in the foreground, triggers, or keyboard sections. The specific commands are TEXT, MULTI_TEXT, RUN_PROGRAMS, POKE_FLD (type 7,23), OL_BUTTON (poke type 7,23), and FUNC_KEY commands. $T strings are valid arguments to these commands wherever string arguments are expected. The text string size limit is 80 characters. The strings must be delimited by single or double quotes. Strings substituted for $T strings in MULTI_TEXT commands will be truncated to 30 characters.

     If the user specifies five text strings to substitute, the first string specified is substituted for all occurrences of any "$T1" text string within any of the six commands above in the foreground, any of the triggers, and the keyboard section; the second string is substituted for all occurrences of any "$T2" text string within these commands, etc., until the text strings have all been substituted or no more occurrences of $T text strings exist in the macro.

- **A list of up to 50 text strings** to be substituted for $T strings in the specified commands in the background section. The specific commands are TEXT, MULTI_TEXT, and RUN_PROGRAMS. The text string size limit is 80 characters. The strings must be delimited by single/double quotes. Strings substituted for $T strings in MULTI_TEXT commands will be truncated to 30 characters.

- **A list of up to 256 set numbers** to be substituted for $SET occurrences in the macro graphic. Valid set numbers are 1 through 255. The first set will be substituted for all occurrences of $SET1 in the file, the second set will be substituted for $SET2 in the file, etc.

  $SET arguments may be used in the macro file to specify:

  a) the set number in the SETVAL command
  b) the set number in poke type 23 POKE_FLD and poke type 23 OL_BUTTON commands
  c) the SETn of a set conditional
  d) the SETn operand in an expression conditional
  e) in a poke list to specify a set argument (SETn) in the POKE_FLD (poke type 7,23), OL_BUTTON (poke type 7,23), RUN_PROGRAMS, and FUNC_KEY commands.

  In following examples, 5 is passed for $SET1:

  a) "SETVAL $SET1 45" => "SETVAL 5 45"
  b) "POKE_FLD 1000 1000 2000 2000 ON 23 $SET1 0 1 6 0 5 A001X001 ID A001X002 ID 1 2 2"=>"POKE_FLD 1000 1000 2000 2000 ON 23 5 0 1 6 0 5 A001X001 ID A001X002 ID 1 2 2"
  c) "COLOR FG RED ($SET1) 3 GREEN CYAN BLUE"=>"COLOR FG RED (SET5) 3 GREEN CYAN BLUE"
  d) "COLOR FG RED ($SET1 = 3) GREEN" => "COLOR FG RED (SET5 = 3) GREEN"

- **A list of up to 256 integer or real numbers** to be substituted for $CONST occurrences in the macro graphic. Remember that real must have a digit before and after the decimal point! The first int/real will be substituted for all occurrences of $CONST1 in the file, the second set will be substituted for $CONST2 in the file, etc. $CONST arguments may be used in the macro file to specify int/real operands in conditional expressions, and to specify int/real arguments in a poke list in the POKE_FLD (type 7, 23), OL_BUTTON (type 7,23), RUN_PROGRAMS, and FUNC_KEY commands. $CONST arguments must be used in poke type 7,23 argument lists where an integer set number is expected; $SET arguments are used in a poke type 7,23 list when a set number in the "SETn" format is expected.

- **A list of up to 256 status words** to be substituted for $STATUS occurrences in the macro graphic (see Appendix B for a list of valid status words). The first status word will be substituted for all occurrences of $STATUS1 in the file, the second status word will be substituted for $STATUS2 in the file, etc. $STATUS identifiers may be used in the macro file to specify status word operands in conditional expressions, and to specify status word arguments in a poke list in the POKE_FLD, OL_BUTTON, RUN_PROGRAMS, and FUNC_KEY commands.

4. If the macro is to be displayed in its as-built size, set wscale = hscale = 1.0;

   If the macro should be displayed half its default size, set wscale = hscale = 0.5;

   If the macro should be displayed twice its default size, set wscale = hscale = 2.0. The scale parameters may be any positive real number. The upper left corner of the macro (the **x,y** parameters) remains fixed when the macro is sized.

5. If multi-byte strings (Chinese, Japanese, etc.) are to be passed to the macro file, it is imperative that the font type be set to **bitmap** or **bitmap_over** in the commands in the macro file wherever the "$T" strings exist. Vector type font does NOT support multi-byte strings.

Example

```
MACRO 5 1234 5242 1.0 0.5 1 3 2 4 2 2 A100 “string 1” “string
2” “string 3” “bg string 1” “bg string 2” 23 45 66 89 21.76
89 HDWRFAIL ALARMACK
```

In this example:

macro_number	=	5
x	=	1234
y	=	5242
wscale	=	1.0
hscale	=	0.5
num_pts	=	1
num_other_tx	=	3
num_bg_tx	=	2
num_sets	=	4
num_consts	=	2
num_statuses	=	2
pts_list	=	A100
other_tx_list	=	“string 1” “string 2” “string 3”
bg_tx_list	=	“bg string 1” “bg string 2”
set_list	=	23 45 66 89
const_list	=	21.76 89
status_list	=	HDWRFAIL ALARMACK

This statement indicates that macro 5 displays at x,y position 1234, 5242. It displays in its as-built width, and half its as-built height. Point 'A100' will be substituted for all occurrences of $D1 in the file. "string 1","string 2", and "string 3" will be substituted for all occurrences of "$T1", "$T2", and "$T3" respectively in the foreground, triggers, or keyboard sections. "bg string 1" and "bg string 2" will be substituted for all occurrences of "$T1" and "$T2" respectively in the background section.

'23' (or 'SET23' where appropriate) will be substituted for all occurrences of $SET1 in the macro; '45' (or 'SET45') will be substituted for all occurrences of $SET2 in the macro; '66' (or 'SET66') will be substituted for all occurrences of $SET3 in the macro; and '89' (or 'SET89') will be substituted for all occurrences of $SET4 in the macro. The real number '21.76' will be substituted for all occurrences of $CONST1 in the file; the integer 89 will be substitute for all occurrences of $CONST2. HDWRFAIL will be substituted for all occurrences of $STATUS1 in the file, and ALARMACK will be substituted for all occurrences of $STATUS2 in the file.

If this is macro5.src before the substitution:

```
DIAGRAM MAIN 0 448 39 700 700 gray80 ZOOMABLE 1 0 0 16384 16384 1
DEFAULT_POSITION DEFAULT_SIZE

BACKGROUND
SETVAL $SET1 45
COLOR FG WHITE BG CYAN ($SET2 > 45) blue
TEXT 444 71  "default bg" ($SET1) 2 "$T1" "$T2" HORZ VECTOR_OVER 152
562 1

FOREGROUND
COLOR FG RED ($D1 AV > $CONST1) BLUE BG WHITE (($D1 AS =
$STATUS1) OR ($D1 AS = $STATUS2)) YELLOW
MULTI_TEXT 701 2107 0 0 3 "$T1"  "$T2" "$T3" VECTOR_OVER 140 257 1

KEYBOARD
COLOR FG BLACK ($SET2 = 7) 3 RED WHITE BLUE
POKE_FLD 398 655 1171 1078 ON 23 $SET4 0 3 6 0 5 A001X001 ID
A001X002 ID 1 $CONST2 2 117 7001 7 5000 60 22 0 2 A001X011 ID
A001X002 ID 119 0 4 1 0 3 "$T3"
```

OL_BUTTON 2107 655 HORZ ROUNDED SHAPE_LABEL 1000 1000 500 500 pointery 90 NONE EXEC_POKE 23 $SET3 0 2 6 0 5 A001X011 ID A001X012 ID 1 66 2 30 0 0

This will be the resultant code added to the graphic after the substitution:

DIAGRAM MAIN 0 448 39 700 700 gray80 ZOOMABLE 1 0 0 16384 16384 1 DEFAULT_POSITION DEFAULT_SIZE

BACKGROUND
SETVAL 23 45
COLOR FG WHITE BG CYAN (SET45 > 45) blue
TEXT 444 71 "default bg" (SET23) 2 "bg string 1" "bg string 2" HORZ
VECTOR_OVER 152 562 1

FOREGROUND
COLOR FG RED (\A100\ AV > 21.76) BLUE BG WHITE ((\A100\ AS = HDWRFAIL) OR (\A100\ AS = ALARMACK)) YELLOW
MULTI_TEXT 701 2107 0 0 3 "string 1" "string 2" "string 3" VECTOR 140 257 1

KEYBOARD
COLOR FG BLACK (SET45 = 7) 3 RED WHITE BLUE
POKE_FLD 398 655 1171 1078 ON 23 89 0 3 6 0 5 A001X001 ID A001X002 ID 1 89 2 117 7001 7 5000 60 22 0 2 A001X011 ID A001X002 ID 119 0 4 1 0 3 "string 3"

OL_BUTTON 2107 655 HORZ ROUNDED SHAPE_LABEL 1000 1000 500 500 pointery 90 NONE EXEC_POKE 23 66 0 2 6 0 5 A001X011 ID A001X012 ID 1 66 2 30 0 0

Note

It is no longer necessary to group the items in the macro file and move them to the upper left corner of the window before saving the macro file. The upper left corner of the outlining rectangle around the macro items will automatically map to the mouse position during the draw/create with this release.

MATH

Description

The MATH command evaluates a mathematical equation and then stores the result in an area of memory in the Operator WEStation called the scratch pad. The scratch pad is addressed by a "$P" pointer with a defined base address and specified offset (see Section 2 for more information on pointers).

Syntax

```
MATH pointer offset equation
```

where:

pointer	=	$P1 through $P99
offset	=	Any valid $P offset ($B, $I, $R, or $S)
equation	=	Infix mathematical equation (infix notation displays the operator between the operands, as opposed to postfix notation, which displays the operator after the operands (that is, AB+).

Rules

1. The valid mathematical operators that can be used in the equation are:
   - Multiplication (*).
   - Division (/).
   - Addition (+).
   - Subtraction (-).
   - Exponentiation (E)
   - Square root (Sqrt).
   - Natural log (Ln).
   - Trigonometric functions (cosine [cos], sine [sin], tangent [tan], arcsine [asin], arccos [acos], and arctan [atan]).

   Note

   Square root, natural log, exponentiation, cosine, sine, tangent, arcsine, arccos, and arctan are not case sensitive. Any combination of upper and lower case letters may be used.

2. The valid operands are integers, reals, process points/record fields, and pointers/offsets.
3. The data stored in the scratch pad buffer could be a previously calculated math result. This data can then be used as an operand to any conditional in the graphics language.
4. Math conditionals are implemented by writing the math result to the scratch pad and then using the scratch pad as an input to a conditional.
5. The mathematical equation is entered in standard infix notation.
6. The math results are stored as real constants.
7. The MATH command may be used in the background, foreground, or trigger sections of a diagram.

Examples

Example 1

```
MATH $P1 $R0 A100 + A200
```

In this example:

pointer	=	$P1
offset	=	$R0
equation	=	A100 + A200

This statement indicates that the analog value of A100 and A200 (default record field = AV) should be added together and the result should be placed in $P1 offset $R0.

Example 2

```
Math $P5 Sqrt ($P1 $R0)
```

In this example:

pointer	=	$P5
offset	=	none (default assumed [$B0] since none is specified)
equation	=	Sqrt ($P1 $R0)

This statement says to take the square root of $P1 $R0 and place it in $P5.

Description

The MULTI_TEXT command displays multiple strings, which are intended to be treated as a single entity, on the diagram. This command automatically handles the justification (left, right, or center) and the inter-line spacing of the strings. The strings are always aligned vertically. The position of the first string is specified in this command, and the positions of the other strings are relative to the first. Note that a MULTI_TEXT command with only 1 string is identical in function to a TEXT command (with horizontal orientation).

Syntax

```
MULTI_TEXT x y justify spacing n string1 string2 ... stringn
<string_cond> font_type font_params
```

where:

x	=	x coordinate of baseline position of first character of first string (if font_type=BITMAP,BITMAP_OVER). x coordinate of upper left corner of outlining rectangle of First string (if font_type=VECTOR,VECTOR_OVER) Valid range is 0 - 16,383.
y	=	y coordinate of baseline position of 1st char of 1st string (if font_type=BITMAP,BITMAP_OVER). y-coordinate of upper left corner of outlining rectangle of First string (if font_type=VECTOR,VECTOR_OVER) Valid range is 0 - 16,383.
justify	=	String justification flag 0 - left justified 1 - right justified 2 - center justified
spacing	=	Interline spacing in screen pixels. Valid range is 0-150.
n	=	Number of strings to be treated as single entity. Valid range is 1-10.

stringi	=	List of n strings in single/double quotes. Strings are limited to 30 characters.
<string_ cond>	=	Optional conditional expression which can display another set of n strings
font_type	=	Font style flag. Choices are: • BITMAP (non-scalable system font) • BITMAP_OVER (non-scalable system font with background of character cell drawn) • VECTOR (scalable Westinghouse font) • VECTOR_OVER (scalable Westinghouse font with background of character cell drawn)
font_params	=	Bitmap font number (if font_type = BITMAP, BITMAP_OVER); w,h,line_width(if_font_type=VECTOR, VECTOR_OVER)
bitmap font number	=	Font size index from fonts.txt. valid range is 1-8.
w	=	Character cell width (in virtual pixels). Valid range is 3-16,383.
h	=	Character cell height (in virtual pixels). Valid range is 3-16,383.
line_width	=	Line width choice from GB:Line Widths window.Valid range is 1-16.

Rules

1. The x and y coordinates for the first string may be relative.
2. The origin is the x,y position in the command.
3. Individual MULTI_TEXT strings always assumed to have horizontal orientation; vertical orientation is not supported. The set of strings are aligned vertically - one below the other.
4. The series of n strings is conditional. The user may specify a different set of strings to be displayed when some condition is met. The same number of strings(n) must be specified in the conditional. The strings are conditional collectively - not individually. The same justification and interline spacing is used for the conditional strings.
5. Strings are limited to 30 characters in the MULTI_TEXT command.
6. If multi-byte text (Chinese, Japanese, etc.) is specified, the f**ont_type** must be set to **bitmap** or **bitmap_over**. Vector font types are NOT supported for multi-byte text strings. Be careful to set the font type appropriately if creating a "$Tn" string for a macro file if multi-byte text will be passed to the macro to replace "$Tn".
7. The MULTI_TEXT command may be used in the background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
MULTI_TEXT 2177 3090 0 0 3 "string 1" "of" "series of 3" VECTOR
140 257 1
```

In this example:

x	=	2177
y	=	3090
justify	=	0(left)
spacing	=	0
n	=	3
strings	=	"string 1", "of", "series of 3"
font_type	=	VECTOR
w	=	140
h	=	257
line_width	=	1

This statement displays the three strings listed above, left justified at x position 2177 on the diagram. The first string is at y=3090. Since there is 0 interline spacing, the second strings is at y=3347(3090+257), and the third string is at y=3604(3347+257). Vector text is used to display the strings, so they will scale when zoomed, and they can be resized manually. The character cell dimensions are 140x257. Single line width is used.

Example 2

```
MULTI_TEXT 2177 3090 1 0 3 “default” “series” “of strings”
(a100 av > 100) “conditional” “series” “of strings”
BITMAP_OVER 3
```

In this example:

x	=	2177
y	=	3090
justify	=	1(right)
spacing	=	0
n	=	3
strings	=	“default”, “series”, “of strings”
<string_cond>	=	(a100 av > 100) “conditional” “series” “of strings”
font_type	=	BITMAP_OVER
bitmap font number	=	3

This statement displays the three strings listed above, right justified at x position 2177 on the diagram. The first character of the first string is at x = 2177. The right justification point will be determined internally from this. Note that x=2177 is NOT the right justification point!

The first string is at baseline position y=3090. There is 0 interline spacing. Bitmap_over text is used to display the strings, so they will not scale when zoomed, and they cannot be resized manually. The third font size from fonts.txt is used to display the text. The character cell backgrounds will be drawn. If the analog value(AV) of a100 is greater than 100, the conditional strings will display instead of the default.

OL_BUTTON

Description

The OL_BUTTON command draws an OPEN LOOK style button. It mimics an OPEN LOOK button in appearance and functionality, but it is a WDPF graphics item (see applicable vendor documentation for more information on OPEN LOOK style buttons).

The ol_button may have a shape or a text label, may be oriented vertically or horizontally, may have rounded or squared endcaps, and may have a poke or a trigger functionality associated with it. A trigger functionality specifies that a user-defined trigger will execute when the user presses and releases the ol_button. A poke functionality specifies that any of the standard poke functions will execute when the user presses and releases the ol_button. Either a poke or a trigger functionality is associated with an ol_button — not both. When the user presses the ol_button, it appears depressed until the user releases it (by releasing the button on the mouse) or moves the mouse off the ol_button.

Note

See the POKE_FLD and TRIGGER reference pages for more information on these items.

Syntax

```
OL_BUTTON x y orientation endcap label_type label
function_type function
```

where:

x = x coordinate of the upper left corner of the outlining rectangle around the button. See Section 3-2 for more information on outlining rectangles.
Valid range = 0 through 16,383.

y = y coordinate of the upper left corner of the outlining rectangle around the button. See Section 3-2 for more information on outlining rectangles.
Valid range = 0 through 16,383.

orientation = Button orientation. The choices are:
- HORZ (endcaps are on the left and right sides).
- VERT (endcaps are on the top and bottom).

endcap = Endcap style. The choices are:
- ROUNDED
- SQUARED

label_type = Specifies a shape or text label to be put on the button. The choices are:
- SHAPE_LABEL
- TEXT_LABEL

label = Varies according to the entry for label_type.
- SHAPE_LABEL = button_w button_h shape_w shape_h shape_name rotation inversion

where:

button_w = Width of button.
Valid range=1 through 16,383.

button_h = Height of button.
Valid range = 1 through 16,383.

shape_w = Width of shape label on button.
Valid range = 1 through button_w -1 (for example, if button_w = 500 then the valid range for the shape width is 500 -1 or 499).

shape_h = Height of shape label on button.
Valid range = 1 through button_h -1 (for example, if button_h = 500 then the valid range for the shape height is 500 -1 or 499).

shape_name = ASCII name of shape as defined in the Shape Library. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on the Shape Library.

rotation = Integer shape rotation. The choices are:
0, 90, 180, 270, -90, -180, -270
(Minus degrees will result in clockwise rotation. Plus degrees will result in counterclockwise rotation.)

inversion = Shape inversion flag. The choices are:
- NONE (no inversion)
- TTB (top to bottom)
- RTL (right to left)
- BOTH (TTB and RTL)

- TEXT_LABEL =

VECTOR char_w char_h line_width string
or
BITMAP font# string

where:

char_w = Width of one character.
Valid range = 3 through 16,383

char_h = Height of one character.
Valid range = 3 through 16,383

line_width = Line width
Valid range = 1 through 16
The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

font# = bitmap font number
Valid range = 1 through 8.

string = ASCII label string, enclosed in single or double quotes.Label is limited to 30 characters.

function_type = Specifies poke or trigger functionality. The choices are:

- EXEC_POKE
- EXEC_TRIG

function = Varies according to entry for function_type.

- EXEC_TRIG = trigger

where:

trigger = Trigger to execute when the button is released (see TRIGGER reference page for more information).
Valid range = 1 through 255

- EXEC_POKE = poke_type poke_args

where:

poke_type = One of the standard poke types (see POKE_FLD reference page for more information). The choices are:
0, 2, 3, 6, 7, 8, 9, 20, 23

poke_args = Argument list for the specified poke_type (see the POKE_FLD reference page for more information).

Rules

1. For an OL_BUTTON with a text label, the button orientation also defines the text orientation. The character width, character height, and line width define the size of the button for VECTOR type labels; the font# defines the size for BITMAP type labels. Leading and/or trailing spaces may be included in the text label. However, it is not possible to vary the amount of space around the top and bottom of the text label for horizontal (HORZ) buttons, or the space around the left and right of the label for vertical (VERT) buttons.

2. Vector or Bitmap text is used for text button labels. Bitmap_over and Vector_over text are not supported for buttons. If a multi-byte label (Chinese, Japanese, etc.) is specified, the text label font type must be set to **bitmap**. Vector font type is NOT supported for multi-byte text labels.

3. For an OL_BUTTON with a shape label, the shape is always centered in the middle of the button. The shape origin is ignored (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on shapes).

4. When an OL_BUTTON is drawn, the label (shape or text) is drawn in the current FG color, and the button itself is drawn using the current OL color from the COLOR command (see COLOR reference page for more information).

5. Text button labels are limited to 30 characters.

6. The origin of the button is the upper left corner of the button's outlining rectangle (x, y).

7. The OL_BUTTON is only valid in the keyboard section of the diagram.

Examples

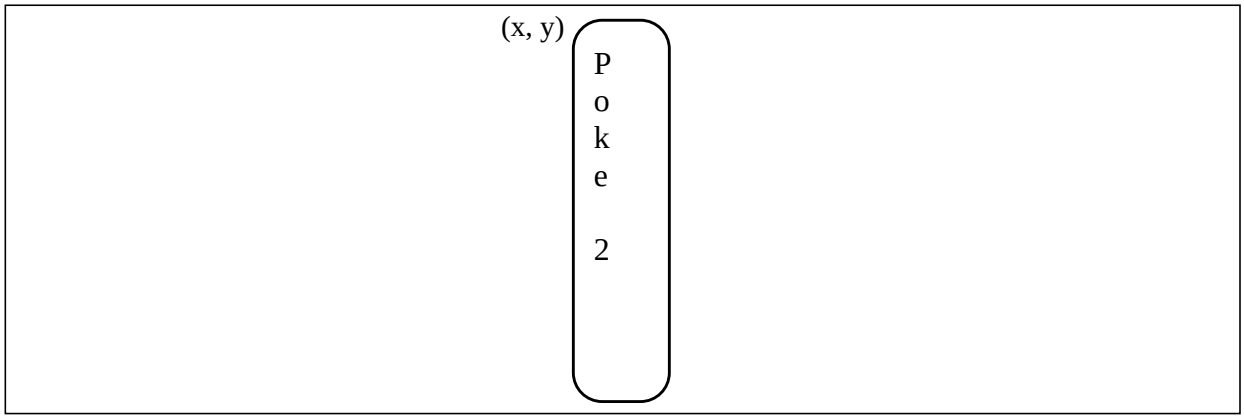
Example 1

```
OL_BUTTON 4915 13739 VERT SQUARED TEXT_LABEL VECTOR 140 257 1
"poke 2" EXEC_POKE 2 3000 5000
```

In this example:

x	=	4915
y	=	13739
orientation	=	VERT
endcap	=	SQUARED
label_type	=	TEXT_LABEL
label	=	VECTOR 140 (char_w) 257 (char_h) 1 (line_width) "poke 2" (string)
function_type	=	EXEC_POKE
poke_type	=	2
poke_args	=	3000 (diagram #) 5000 (group #)

This example specifies a vertical button with a vector type text label of "poke 2". The text is vertically oriented. The text character width is 140, and the character height is 257. A line width of 1 is used for the text. The button has squared endcaps on the top and button of the button. The upper left corner of the button is at 4915, 13739. When the user presses and releases this button, a poke type 2 function occurs. A poke type 2 displays a diagram and group. Diagram 3000 will be displayed with group 5000 when this button is pressed. See figure below:



Example 2

```
OL_BUTTON 7302 3160 HORZ ROUNDED SHAPE_LABEL 1000 1000 500 500
valve 0 NONE EXEC_TRIG 3
```

In this example:

x	=	7302
y	=	3160
orientation	=	HORZ
endcap	=	ROUNDED
label_type	=	SHAPE_LABEL
label	=	1000 (button_w) 1000 (button_h) 500 (shape_w) 500 (shape_h) valve (shape_name) 0 (rotation) NONE (inversion)
function_type	=	EXEC_TRIG
function	=	3 (trigger)

This example specifies a horizontal button (endcaps are on the left and right sides as opposed to top and bottom) with a shape label. The upper left corner of the button is at 7302, 3160. The button dimensions are 1000 x 1000. The valve shape is not rotated (rotation = 0) and is not inverted (inversion = NONE). The dimensions of the shape are 500 x 500. The button endcaps are rounded. Trigger 3 will execute when the user presses and releases the button.

Description

The OL_CHECKBOX command displays OPEN LOOK style checkboxes on a diagram. It mimics the OPEN LOOK checkboxes in appearance and functionality, but it is a WDPF graphics item (see applicable vendor documentation for more information on OPEN LOOK style checkboxes).

The user specifies the number of checkboxes, the size of the checkboxes, the orientation for the series of checkboxes, and the spacing between the checkboxes. The user also specifies whether the checkboxes will be exclusive (only one box may be checked at a time) or non-exclusive (multiple boxes may be checked at the same time).

An integer offset into the scratch pad area (a $P pointer and $I offset) is associated with each checkbox item (see Section 2 for more information on pointers and offsets). The value of the checkbox is read from and written to this location. The value of the ol_checkbox item is consistent with the OPEN LOOK checkbox.

If the checkbox is exclusive, the value is defined as the ordinal number of the selected box beginning at 0. If the checkbox is non-exclusive, the value is a bit mask where each set bit represents a checked box and each unset bit represents an unchecked box (the low order bit represents the first box, the second bit to the right represents the second box, and so forth).

A trigger functionality is associated with the checkbox item (see TRIGGER reference page for more information). When the user selects or deselects a box, the value of the item is updated in the scratch pad area, and the specified trigger is executed. Coding of the trigger is user-configurable. The value of the checkbox item is read each time the foreground of the diagram updates. If the value changes, the display of the checkbox item changes accordingly.

When the user selects a box for an exclusive item, a checkmark is drawn inside that box, and the existing checkmark is erased. When the user selects a box for a non-exclusive item, a checkmark is drawn in the box if the box is not currently checked, regardless of whether of not another box is already checked. To deselect a box, the user clicks on a box that is checked which erases the checkmark (non-exclusive items allow the user to toggle the state of the checkboxes).

Syntax

```
OL_CHECKBOX x y w h boxes spacing orientation type value
trigger
```

where:

x = x coordinate of the upper left corner of first checkbox area (includes invisible portion reserved for checkmark).
Valid range = 0 through 16,383.

y = y coordinate of the upper left corner of first checkbox area (includes invisible portion reserved for checkmark).
Valid range = 0 through 16,383.

w = Width of checkbox area (includes invisible portion reserved for checkmark).
Valid range = 1 through 16,383.

h = Height of checkbox area (includes invisible portion reserved for checkmark).
Valid range = 1 through 16,383.

boxes = Number of checkboxes in the list.
Valid range = 1 through 32.

spacing = Virtual pixels between successive boxes (minimum = 0).
Valid range = 0 through 16,383.

orientation = Orientation of the checkboxes. The choices are:
- HORZ (horizontal)
- VERT (vertical)

type = Exclusive or non-exclusive flag. The choices are:
- EXCL (only one box may be checked at a time)
- NON_EXCL (multiple boxes may be checked at the same time)

value = Scratch pad area ($P1 - $P99 pointer and an integer $I offset) from which to read checkbox item value and to which to write checkbox item value. See Section 2 for more information on pointers and offsets.

trigger = Trigger to execute when the value of the checkbox item changes.
Valid range = 0 through 255.

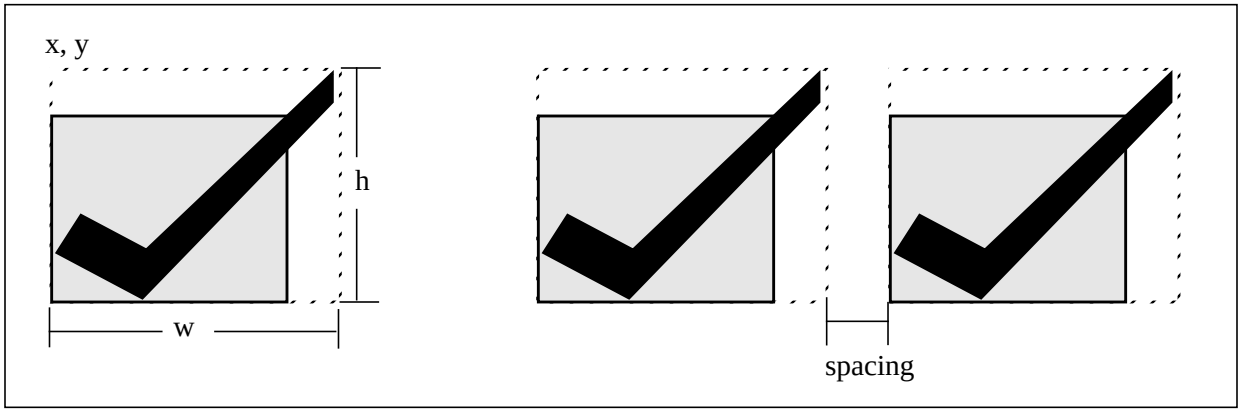
Rules

1. If a trigger of 0 is specified, then no trigger is executed when the value of the checkbox item changes.

2. The checkboxes are drawn in the current OL color from the COLOR command. The checks are drawn in the current FG color from the COLOR command. The ER color from the current COLOR command is used to erase the upper right portion of the checkmark that extends outside of the checkbox when the checkmark is erased. The user should set the ER color to the current BG color on which the checkbox item lies so that this erasure is not visible. See COLOR reference page for more information on the COLOR command.

3. The Graphics Builder always displays ol_checkbox with the first box checked (only the first box). The value of the scratch pad will be read when the diagram is initially displayed to determine which boxes are initially checked on the Operator WEStation at runtime.

4. The checkbox width and height include the invisible portion reserved for the upper end of the checkmark. The spacing is the distance between the right/ bottom edge of one checkbox area (including the invisible part) to the left/top edge of the successive checkbox area (including the invisible part). See below:



5. The possible values for an exclusive type checkbox item are 0 through boxes - 1 (for example, if boxes = 5 then the upper limit is 5 -1 or 4). The possible values for a non-exclusive type checkbox are 0 through 2^{boxes} - 1 (for example, if boxes = 3, then the upper limit is 8 - 1 or 7).

6. The origin of the checkbox item is the upper left corner of the checkbox outlining rectangle (x, y).

7. This item is only valid in the keyboard section of the diagram.

Examples

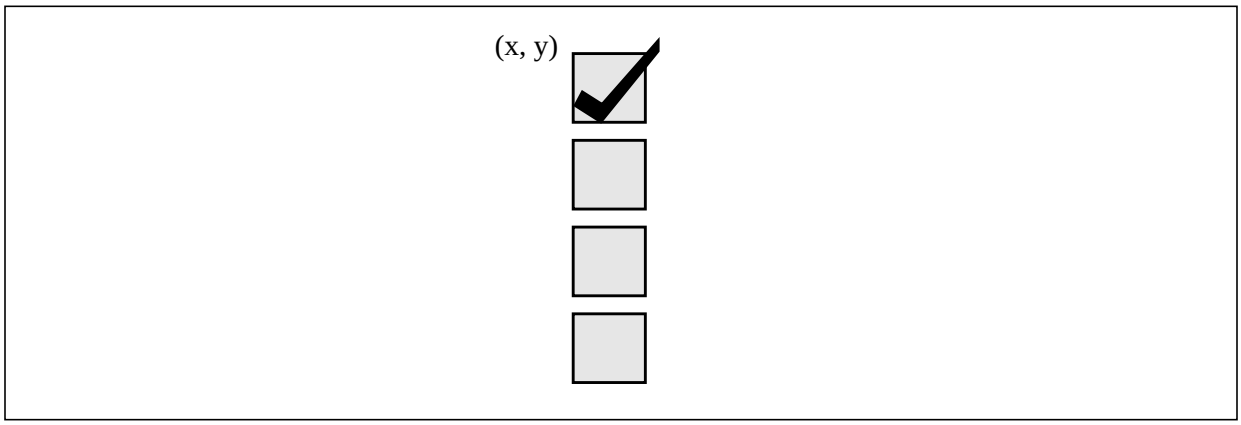
Example 1

```
OL_CHECKBOX 100 100 500 500 4 100 VERT NON_EXCL $P5 $I4 0
```

In this example:

x	=	100
y	=	100
w	=	500
h	=	500
boxes	=	4
spacing	=	100
orientation	=	VERT
type	=	NON_EXCL
value	=	$P5 $I4
trigger	=	0

This example specifies a vertical list of 4 non-exclusive checkboxes. The upper left corner of the first box is at 100,100. The dimensions of each box are 500 x 500. The distance (in the y dimension) between successive boxes is 100 virtual pixels. This is the distance from the bottom of one box to the top of the successive box. The scratch pad area associated with this checkbox item is $P5 $I4. The possible values of this checkbox are 0 (no boxes checked) through 15 (all boxes checked). If the value = 1, the top-most box is checked; if the value = 8, the bottom-most box is checked. No trigger will be executed when the user checks or unchecks a box since trigger 0 is specified. See figure below:



Example 2

```
OL_CHECKBOX 100 100 500 500 4 100 HORZ EXCL $P1 $I0 255
```

In this example:

x	=	100
y	=	100
w	=	500
h	=	500
boxes	=	4
spacing	=	100
orientation	=	HORZ
type	=	EXCL
value	=	$P1 $I0
trigger	=	255

In this example, a horizontal list of 4 exclusive checkboxes is created. The first checkbox is at 100, 100. Each box is 500 x 500. There are 100 virtual pixels between successive boxes. The spacing begins at the right end of one checkbox, and defines where the left edge of the successive checkbox will begin. The scratch pad area used to store and read the value of the checkbox is $P1 $I0. Trigger 255 will execute when the user checks a box. Prior to executing the trigger, the new checkbox value will already have been updated in the scratch pad. The possible values for this checkbox item are 0 (left-most box checked), 1 (second box checked), 2 (third box checked), or 3 (right-most box checked).

Description

The OL_CHOICE command displays an OPEN LOOK style choice item on the diagram. The OL_CHOICE command mimics the OPEN LOOK choice item in appearance and functionality, but it is a WDPF graphics item (see applicable vendor documentation for more information on OPEN LOOK style choice items).

The user specifies the number of choices, the orientation for the choices, whether text or shape labels should be used, and whether the choices will be exclusive (only one box may be depressed at a time) or non-exclusive (multiple boxes may be depressed at the same time). An integer offset into the scratch pad area (a $P pointer and $I offset) is associated with each choice item (see Section 2 for more information on pointers and offsets). The value of the choice is read from and written to this location. The value of the ol_choice item is consistent with the OPEN LOOK choice item. If the choice is exclusive, the value is defined as the ordinal number of the depressed box beginning at 0. If the choice is non-exclusive, the value is a bit mask where each set bit represents a depressed box and each unset bit represents a raised box (the low order bit represents the first box, the second bit to the right represents the second box, and so forth).

A trigger functionality is associated with the choice item (see TRIGGER reference page for more information). When the user selects or deselects a box, the value of the item is updated in the scratch pad area, and the specified trigger is executed. Coding of the trigger is user-configurable. The value of the choice item is read each time the foreground of the diagram updates. If the value changes, the display of the choice item changes accordingly. When the user selects a box for an exclusive item, that box is depressed and the existing depressed box is raised.

When the user selects a box for a non-exclusive item, the box is depressed if the box is not currently depressed, regardless of whether or not another box is already depressed. To raise a box, the user clicks on a box that is depressed (non-exclusive items allow the user to toggle the state of the choice items).

Syntax

```
OL_CHOICE x y n orientation type value trigger label_type
label_info label1 .. labeln
```

where:

x = x coordinate of the upper left corner of first choice box.
Valid range = 0 through 16,383.

y = y coordinate of the upper left corner of first choice box.
Valid range = 0 through 16,383.

n = Number of choices.
Valid range = 1 through 64.

orientation = Orientation of choice boxes. The choices are:
- HORZ (horizontal)
- VERT (vertical)

type = Exclusive or non-exclusive flag. The choices are:
- EXCL (only one item may be depressed at a time).
- NON_EXCL (multiple items may be depressed at a time).

value = Scratch pad area associated with item ($P1 - $P99 and $I offset).

trigger = Trigger to execute when user selects or deselects a new choice.
Valid range = 0 through 255 (0 means do not execute a trigger).

label_type = Specifies text or shape labels on the individual choices. The choices are:
- TEXT_LABEL
- SHAPE_LABEL

label_info = General information which applies to all of the choice labels. This varies depending on the entry for label_type:

- TEXT_LABEL =

VECTOR char_w char_h line_width

or

BITMAP font#

where:

char_w = Width of a character.
Valid range = 3 through 16,383.

char_h = Height of a character.
Valid range = 3 through 16,383.

line_width = Line width.
Valid range = 1 through 16.
The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

font# = Bitmap font number. Valid range = 1 through 8.

- SHAPE_LABEL = box_w box_h shape_w shape_h

where:

box_w = Width of each choice box.
Valid range = 1 through 16,383.

box_h = Height of each choice box.
Valid range = 1 through 16,383.

shape_w = Width of each shape label.
Valid range = 1 through box_w - 1 (for example, if box_w = 500 then the valid range for the shape width is 500 -1 or 499).

shape_h = Height of each shape label.
Valid range = 1 through box_h - 1 (for example, if box_h = 500 then the valid range for the shape width is 500 -1 or 499).

label1... labeln = Varies according to the entry for label_type.

- TEXT_LABEL = string1 .. stringn

where:

string1 ... stringn = String (maximum of 30 characters) enclosed in single or double quotes.

- SHAPE_LABEL = shape1 rot1 inv1... shapen rotn invn

where:

shape1..shapen = ASCII shape name from Shape Library. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on the Shape Library.

rot1 ... rotn = Shape rotations. The choices are:
0, 90, 180, 270, -90, -180, -270
(Minus degrees will result in clockwise rotation. Plus degrees will result i n counterclockwise rotation.)

inv1 ... invn = Shape inversions. The choices are:
NONE (no inversion)
TTB (top to bottom)
RTL (right to left)
BOTH (TTB and RTL)

Rules

1. If a trigger of 0 is specified, then no trigger is executed when the value of the choice item changes.

2. The choice labels (shape or text) are all the same color. The user cannot vary the color of individual choice labels. The text labels all have the same font size. Text labels are either BITMAP or VECTOR text; BITMAP_OVER and VECTOR_OVER are not supported. If a multi-byte label (Chinese, Japanese, etc.) is specified, the text label font type must be set to **bitmap**. Vector font type is NOT supported for multi-byte text labels. The shape labels all have the same width and height. The dimensions of the choice boxes are all the same for shape labels. The height dimensions are the same for text labels, and the width dimensions vary according to the individual label lengths.

3. Text labels are limited to 30 characters.

4. Vertical (VERT) orientation for choices with text labels aligns the choice boxes vertically. The text is always horizontal for choice items. The user cannot have vertical text in a choice item.

5. The choice boxes are drawn in the current OL color from the COLOR command. The choice labels are drawn in the current FG color from the COLOR command. See the COLOR reference page for more information.

6. The Graphics Builder always displays ol_choice with the first box depressed (only the first box). The value of the scratch pad will be read when the diagram is initially displayed on the Operation WEStation at runtime to determine which boxes are initially depressed.

7. The possible values for an exclusive type choice item are 0 through n-1 (for example, if n = 5 then the upper limit is 5 -1 or 4). The possible values for a non-exclusive type choice are 0 through 2^n - 1 (for example, if n = 3, then the upper limit is 8 - 1 or 7).

8. The shape labels are always centered within the choice boxes. The shape origin is ignored when drawing the shape label on the choice box. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on shapes.

9. The origin of the choice item is the upper left corner of the first choice box (x, y).

10. This item is only valid in the keyboard section of the diagram.

Examples

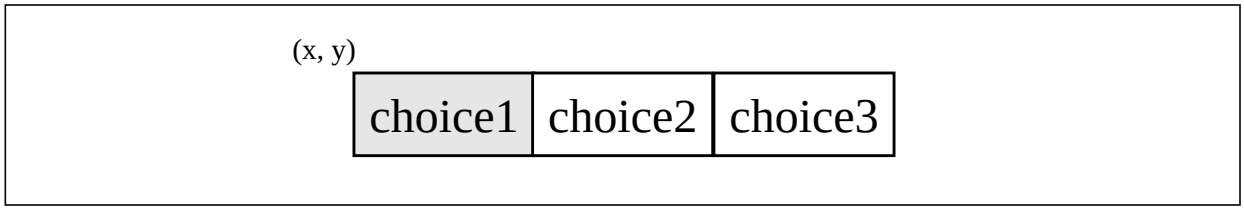
Example 1

```
OL_CHOICE 1000 1000 3 HORZ EXCL $P2 $I10 23 TEXT_LABEL VECTOR
105 211 1 “choice1” “choice2” “choice3”
```

In this example:

x	=	1000
y	=	1000
n	=	3
orientation	=	HORZ
type	=	EXCL
value	=	$P2 $I10
trigger	=	23
label	=	TEXT_LABEL
label_info	=	VECTOR 105 (char_w) 211 (char_h) 1 (line_width)
label1...label n	=	”choice1” “choice2” “choice3”

This example creates a horizontal choice item with 3 choices: choices1, choice2, and choice3. The labels are vector text. The character dimensions are 105 x 211. The line width is 1. The choice is exclusive (only one choice may be selected at a time). The upper left corner of the first choice box is at 1000, 1000. The scratch pad area associated with the choice item is $P2 $I10. Trigger 23 will be executed when the user selects one of the 3 choice boxes on the Operator WEStation at runtime. See figure below:



Example 2

```
OL_CHOICE 1000 1000 3 VERT NON_EXCL $P2 $I10 0 SHAPE_LABEL
2000 1000 500 500 valve1 0 NONE valve2 90 NONE valve3 180 NONE
```

In this example:

x	=	1000
y	=	1000
n	=	3
orientation	=	VERT
type	=	NON_EXCL
value	=	$P2 $I10
trigger	=	0
label_type	=	SHAPE_LABEL
label_info	=	2000 (box_w) 1000 (box_h) 500 (shape_w) 500 (shape_h)
label1...labeln	=	valve1 (shape_name1) 0 (rot1) NONE (inv1); valve2 (shape_name2) 90 (rot2) NONE (inv2); valve3 (shape_name3) 180 (rot3) NONE (inv3)

In this example, a vertical choice item with 3 shape labels is created. The upper left corner of the first choice box is at 1000, 1000. Each choice box is 2000 x 1000. Each shape is scaled to 500 x 500 and centered within the choice boxes. None of the shape labels are inverted. The second and third shape labels are rotated.

This choice item is non-exclusive, which means that more than one choice box may be selected at the same time. The scratch pad area associated with this choice item is $P2 $I10. No trigger will be executed when the user selects or deselects a choice since the trigger = 0.

OL_CYLINDER

Description

The OL_CYLINDER command displays a cylindrical bar graph. This command fills a vertical cylinder from the bottom up to the current value of some process point, scaled between the low and high limits defined for that point. See "Record Types User's Guide" (U0-0131) for more information on process points and record fields.

Syntax

```
OL_CYLINDER x y w h pt_name rec_fld low high
```

where:

x	=	x coordinate of the upper left corner of the cylinder's outlining rectangle. See Section 3-2 for more information on outlining rectangles. Valid range=0 through 16,383.
y	=	y coordinate of the upper left corner of the cylinder's outlining rectangle. See Section 3-2 for more information on outlining rectangles. Valid range=0 through 16,383.
w	=	Width of cylinder. Valid range = 1 through 16,383.
h	=	Height of cylinder. Valid range = 1 through 16,383.
pt_name	=	Name of point to be represented by the cylinder.
rec_fld	=	Optional. Data field within the point of which to display the value. If no record field is specified, the default record field for the given point type is assumed (see Section 2 for more information of default record fields).

low = Low limit used to scale the value.
Valid values are:
- Integers in the range -2,147,483,647 through 2,147,483,647
- Real constants
- Point names/record fields
- Pointers/offsets

high = High limit used to scale the value.
Valid values are:
- Integers in the range -2,147,483,647 through 2,147,483,647
- Real constants
- Point names/record fields
- Pointers/offsets

Rules

1. The x and y coordinates may be relative.
2. The origin of the cylinder is the upper left corner of the cylinder's outlining rectangle.
3. The cylinder fills from bottom to top. There is only one direction for the cylinder (up). The bottom of the cylinder is the low limit end; the top of the cylinder is the high limit end.
4. The empty (unfilled) portion of the cylinder, along with the top, are drawn in the FG color from the current COLOR command. The filled portion of the cylinder is drawn in the OL color from the current COLOR command. The ER color from the COLOR command is used to erase some intermediate drawing in the lower corners of the cylinder outlining rectangle. See the COLOR reference page for more information.
5. The OL_CYLINDER command may reside in the background, foreground, or trigger sections of a diagram.

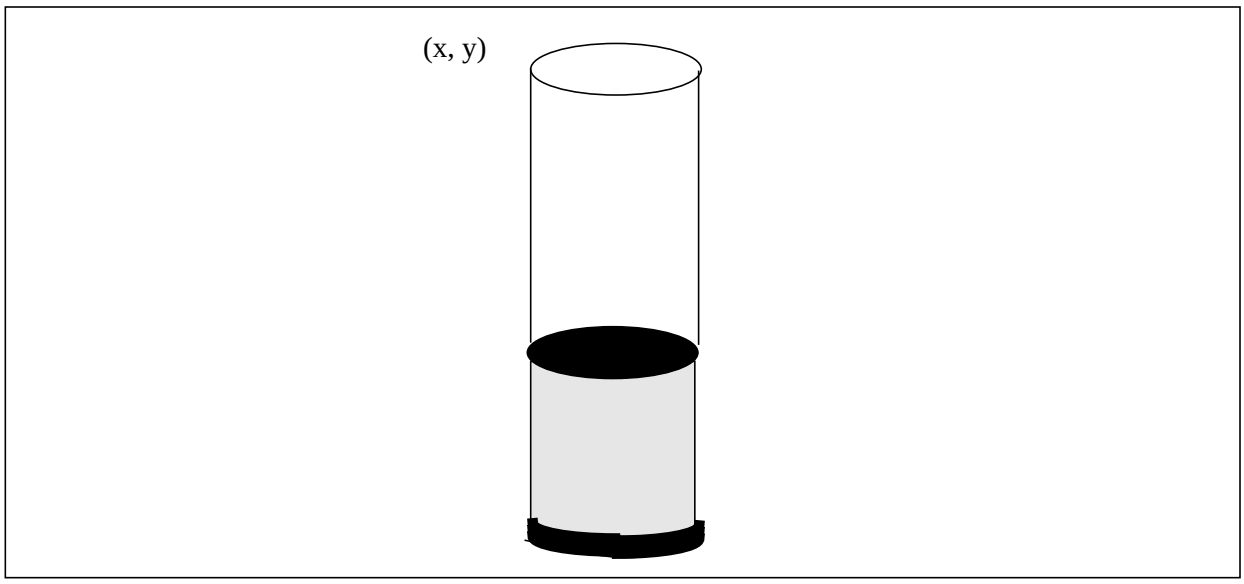
Example

```
OL_CYLINDER 5951 13305 2179 10488 A3890 AV 20 80
```

In this example:

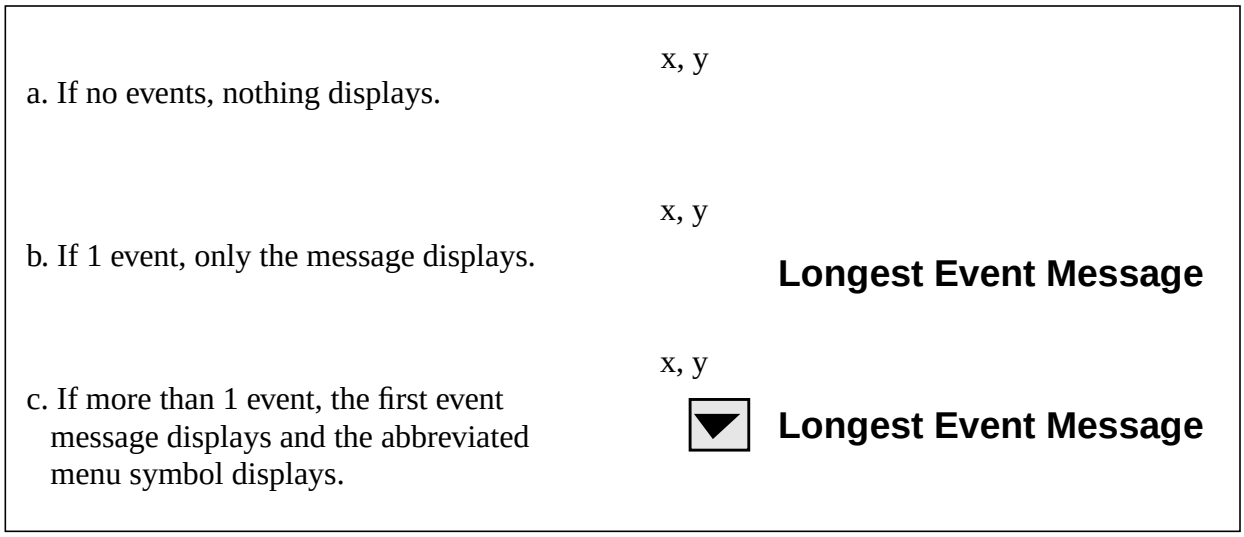
x	=	5951
y	=	13305
w	=	2179
h	=	10488
pt_name	=	A3890
rec_fld	=	AV
low	=	20
high	=	80

In this example, the OL_CYLINDER has an origin at 5951,13305. The dimensions of the cylinder are 2179 x 10488. The value represented by the cylinder is the AV field of process point A3890. This value is scaled between the low limit of 20 and the high limit of 80. See figure below:



Description

The OL_EVENT_MENU command displays an OPEN LOOK style abbreviated menu button on a diagram (but it is a WDPF graphics item). The menu may consist of up to 32 event or alarm messages. An event or alarm menu item is specified by a point name and record field, a status word, and the message to display if the status word bits match the point name record field bits. If no alarm conditions are found, nothing displays on the diagram (including the abbreviated menu symbol). If only one alarm condition is true, the associated alarm message displays, but no abbreviated menu symbol displays. If more than one alarm condition is true, the first alarm message is displayed along with the abbreviated menu symbol. See figure below.



If the user clicks on the abbreviated menu symbol when displayed on the Operator WEStation at runtime, a window resembling an OPENLOOK menu will pop up to the right of the abbreviated menu symbol. It will list all of the applicable alarm or event messages. This menu is for display purposes only. The user cannot select from this menu (that is, the user cannot acknowledge alarms from this menu). If the user clicks on the abbreviated menu symbol while the menu is being displayed, the menu will disappear.

OL_EVENT_MENU
Cont'd

Note

The Graphics Builder will always display the abbreviated menu symbol and the longest alarm or event message. The Graphics Builder will NOT support popping up the event menu when the user clicks on the abbreviated menu symbol.

Syntax

```
OL_EVENT_MENU x y n ptname_1 recfld_1 status_1 msg_1 ptname_n
recfld_n status_n msg_n font_params
```

where:

x	=	x coordinate of the upper left corner of abbreviated menu symbol. Valid range = 0 through 16,383.
y	=	y coordinate of the upper left corner of abbreviated menu symbol. Valid range = 0 through 16,383.
n	=	Number of possible event or alarm conditions in the menu. Valid range = 1 through 32.
ptname_1 ...ptname_n	=	ASCII point name from System Point Directory, dummy point name, or $ pointer. See Section 2 for more information.
recfld_1 ... recfld_n	=	Two-character ASCII record field or $ offset (used with $P, $D points). See Section 2 for more information.
status_1 ... status_n	=	Status word (see Appendix B for list of valid status words).
msg_1 ... msg_n	=	ASCII event or alarm message (enclosed in single or double quotes) to display if alarm condition is met. Message strings are limited to 50 characters.

font_params	=	BITMAP font# or VECTOR char_w char_h line_width
char_w	=	Character width. Valid range = 3 through 16, 383. See Section 3-2 for more information.
char_h	=	Character height. Valid range = 3 through 16, 383. See Section 3-2 for more information.
line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.
font#	=	Bitmap font number. Valid range = 1 through 8.

Rules

1. Vector or Bitmap text is used for the ol_event_menu. Bitmap_over and Vector_over text are NOT supported for this item. If a multi-byte string (Chinese, Japanese, etc.) is specified for any event, the font param must be set to **bitmap**. Vector font type is NOT supported for multi-byte text.

2. The event menu pop-up window will be sized in the x dimension by the longest event or alarm message string in the gcode.

3. The abbreviated menu symbol and the background of the pop-up event menu will use the OL color from the current COLOR command. The background color for the event or alarm messages will be the OL color selected for the event menu item. The FG color from the COLOR command will be used to display the event or alarm messages on the pop-up menu and also on the canvas. All messages will be displayed in the same FG color. See the COLOR reference page for more information on color.

4. Integer (32 bit) or short (16 bit) record fields must be used in conjunction with the status words in defining the alarm or event conditions.

5. Maximum event menu string length is 50 characters.
6. The origin of the event menu is the upper left corner of the event menu outlining rectangle (x, y).
7. This item may be used only in the keyboard section of a diagram.

Examples

Example 1

```
OL_EVENT_MENU 2879 7490 4 A100 AS HDWRFAIL "A100 is timed-out"
A100 AS BAD "A100 is bad" A100 AS FAIR "A100 is fair" A100 AS
POOR "A100 is poor" VECTOR 300 300 1
```

In this example:

x	=	2879
y	=	7490
n	=	4
font_params	=	VECTOR 300 300 1
pt_name1	=	A100
rec_fld_1		AS
status_1		HDWRFAIL
msg_1		"A100 is timed-out"
pt_name2	=	A100
rec_fld_2		AS
status_2		BAD
msg_2		"A100 is bad"

pt_name3	=	A100
rec_fld_3		AS
status_3		FAIR
msg_3		"A100 is fair"
pt_name4	=	A100
rec_fld_4		AS
status_4		POOR
msg_4		"A100 is poor"

In this example, an ol_event_menu item is created having the upper left corner of the abbreviated menu symbol at 2879, 7490. There are four possible event or alarm strings on the menu. The character dimensions are 300 x 300. The line width is 1.The text type is vector.

The event or alarm conditions are each based on comparing the AS record field of point A100 with some status words. The event message strings reflect the point and the status that generated the event or alarm. The Graphics Builder will display the "A100 is timed-out" string on its canvas since this is the longest string.

Example 2

```
OL_EVENT_MENU 1000 2000 3 $G1 DS SET "$G1 is set" $G2 DS RESET
"$G2 is reset" $G3 DS SET "$G3 is set" BITMAP 2
```

In this example:

x	=	1000
y	=	2000
n	=	3
pt_name1	=	$G1
rec_fld_1		DS
status_1		SET
msg_1		"$G1 is set"
pt_name2	=	$G2
rec_fld_2		DS
status_2		RESET
msg_2		"$G2 is reset"
pt_name3	=	$G3
rec_fld_3		DS
status_3		SET
msg_3		"$G3 is set"
font_ params	=	BITMAP 2

In this example, an ol_event_menu item is created having the upper left corner of the abbreviated menu symbol at 1000, 2000. There are three possible event messages to be displayed. The second bitmap size is used to display the text. As bitmap type text is used, the event menu will not resize after a zoom or window resize. Each event or alarm condition is based on checking the DS field of a digital point against the SET or RESET status words. Three unique points are used in this example. The Graphics Builder will display the "$G2 is reset" string on its canvas since this is the longest string. See figure below:

Description

The OL_GAUGE command displays an OPEN LOOK style gauge, similar in functionality to a bar graph (ol_gauge is a WDPF draw item). This command fills the gauge up to the current value of some process point, scaled between the low and high limits defined for that point. The gauge may be filled from left to right, right to left, top to bottom, bottom to top, or from the 0 value up or down (bias).

Syntax

```
OL_GAUGE x y w h direction pt_name rec_fld low high
```

where:

x = x coordinate of the upper left corner of the gauge's outlining rectangle.
Valid range = 0 through 16,383.

y = y coordinate of the upper left corner of the gauge's outlining rectangle.
Valid range = 0 through 16,383.

w = Width of gauge.
Valid range = 1 through 16,383.

h = Height of gauge.
Valid range = 1 through 16,383.

direction = Direction for filling the gauge (low to high limit). The choices are:

- UP — fills vertically from bottom to top
- DOWN — fills vertically from top to bottom
- LEFT — fills horizontally from right to left
- RIGHT — fills horizontally from left to right
- BIAS — fills vertically from the 0 value position either up or down depending on the current value

pt_name = Name of point to be represented by the gauge. See "Record Types User's Guide" (U0-0131) for more information on process points and record fields.

rec_fld = Optional. Data field within the given point of which to display the value. If no record field is specified, the default record field for the given point type is assumed (see Section 2 for more information of default record fields).

low = Low limit used to scale the value. Valid values are:
- Integers in the range -2,147,483,647 through 2,147,483,647
- Real constants
- Point names/record fields
- Pointers/offsets

high = High limit used to scale the value. Valid values are:
- Integers in the range -2,147,483,647 through 2,147,483,647
- Real constants
- Point names/record fields
- Pointers/offsets

Rules

1. The x and y coordinates may be relative.
2. The origin of the gauge is the upper left corner of the gauge's outlining rectangle.
3. Unlike the BAR command, OL_GAUGE uses the upper left corner of the gauge as the origin for all directions. This means that the x, y on the source command is the upper left corner of the gauge for all directions.
4. The elliptical gauge background is drawn in the OL color from the current COLOR command. The bar portion of the gauge is drawn in the FG color from the current COLOR command. See COLOR reference page for more information on color.
5. An endcap is drawn at the low limit end of the bar portion of the gauge for all directions except the BIAS gauge.
6. The OL_GAUGE command may reside in the background, foreground, or trigger sections of the diagram.

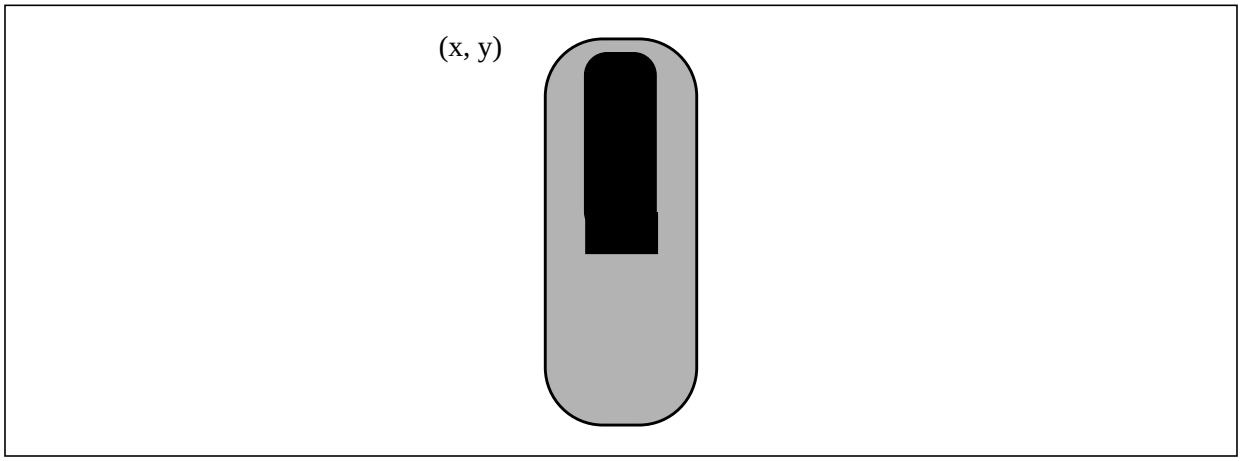
Example

```
OL_GAUGE 5951 13305 2179 10488 DOWN A3890 AV 20 80
```

In this example:

x	=	5951
y	=	13305
w	=	2179
h	=	10488
direction	=	DOWN
pt_name	=	A3890
rec_fld	=	AV
low	=	20
high	=	80

In this example, the ol_gauge has an origin at 5951, 13305. The dimensions of the gauge are 2179 x 10488. The value represented by the gauge is the AV field of process point A3890. This value is scaled between the low limit of 20 and the high limit of 80. The gauge fills from the top down. The low limit end of the gauge is at the top, and there is an endcap on the top of the bar portion of the gauge to mark this as the start point for the fill. See figure below:



Description

The OL_RECTANGLE command displays an OPEN LOOK style rectangle on the graphic (but it is a WDPF graphics item). This item is for display purposes only (it cannot be pressed or released like a button or poke field). This item may be displayed in the normal (raised) or the invoked (depressed) state. It may be used like a regular rectangle except that ol_rectangle gives a three-dimension look on a diagram.

Syntax

```
OL_RECTANGLE x y w h state
```

where:

x = x coordinate of the upper left corner of the ol_rectangle.
Valid range= 0 through 16,383.

y = y coordinate of the upper left corner of the ol_rectangle.
Valid range= 0 through 16,383.

w = Width of the rectangle.
Valid range= 1 through 16,383.

h = Height of the rectangle.
Valid range= 1 through 16,383.

state = State of the rectangle. The choices are:

- OLNORMAL (rectangle appears raised)
- INVOKED (rectangle appears pressed in)

Rules

1. The x and y coordinates may be relative.
2. The origin of this item is the upper left corner of the item (x, y).
3. This item is drawn in the OL color from the current COLOR command. See COLOR reference page for more information on color.
4. Line width, line pattern, and/or fill patterns do NOT apply to this item.
5. The OL_RECTANLGE command may reside in the background, foreground, or trigger sections of the diagram.

Example

```
OL_RECTANGLE 5000 6000 1000 3000 INVOKED
```

In this example:

x = 5000

y = 6000

w = 1000

h = 3000

state = INVOKED

In this example, an OPEN LOOK rectangle is drawn in the invoked state (that is, it appears to be pressed in). The upper left corner of the rectangle is at 5000, 6000. The dimensions of the rectangle are 1000 x 3000.

Description

The OL_SLIDER command displays an OPEN LOOK style slider on the graphic. It mimics an OPEN LOOK slider in appearance and functionality, but it is a WDPF draw item (see applicable vendor documentation for more information on OPEN LOOK style sliders).

The slider is both an input and output device. The slider control (the box the user grabs (with the mouse) that moves along the slider bar) represents the current value of the data at a specified address in the scratch pad scaled between a low and high limit. When the user moves the slider control, the data within the scratch pad is updated. Two unique triggers may be associated with the slider. One trigger is called continuously as the user is moving the slider control. The other trigger is called once when the user releases the mouse button after dragging the slider control.

Notes

- The OL_SLIDER command may be used to do digital set point entry. This is done by calling the RUN_PROGRAMS command (from the trigger executed when the user releases the slider control) to run the XPID_DIGITAL (121) application program.
- If a digital readout is desired with the slider, a PROCESS_PT command should be called from the trigger executed continuously as the slider control moves. The PROCESS_PT command should display the value of the scratch pad area associated with the slider.
- See RUN_PROGRAMS and PROCESS_PT reference pages for more information on these commands. See Section 4 for more information on application programs.

Syntax

```
OL_SLIDER x y w h direction value low high trigger1 trigger2
```

where:

x = x coordinate of the upper left corner of the slider's outlining rectangle.
Valid range = 0 through 16,383.

y = y coordinate of the upper left corner of the slider's outlining rectangle.
Valid range = 0 through 16,383.

w = Width of slider.
Valid range = 1 through 16,383.

h = Height of slider.
Valid range = 1 through 16,383.

direction = Direction for moving the slider control (defines low and high end). The choices are:

- UP — increases in value vertically from bottom to top
- DOWN — increases in value vertically from top to bottom
- LEFT — increases in value horizontally from right to left
- RIGHT — increases in value horizontally from left to right

value = Scratch pad area associated with the slider. Represents the current value of the slider ($P1 - $P99 pointers and $S, $I, $R, or $B offsets). See Section 2 for more information on pointers and offsets.

low = Low limit used to scale the value. Valid values are:

- Integers in the range -2,147,483,647 through 2,147,483,647
- Real constants
- Point names/record fields
- Pointers/offsets

high = High limit used to scale the value. Valid values are:

- Integers in the range -2,147,483,647 through 2,147,483,647
- Real constants
- Point names/record fields
- Pointers/offsets

trigger1 = Trigger executed continuously as user drags or moves slider control.
Valid range = 0 through 255 (0 = do not execute a trigger)

trigger2 = Trigger executed one time when the slider control is released.
Valid range = 0 through 255 (0 = do not execute a trigger)

Rules

1. The slider is drawn using the OL color from the current COLOR command. The ER color from the same COLOR command is used to erase the slider control as the user moves or drags it. The user should take care to set the ER color to the background color on which the slider lies. See the COLOR reference page for more information on color.

2. The value of the slider (scratch pad area) will be read each time the foreground updates on the Operator WEStation at runtime. If the value changes, the slider display will update accordingly.

3. The origin of the slider is the upper left corner of the slider's outlining rectangle (x, y).

4. The OL_SLIDER command can only be used in the keyboard section of a diagram.

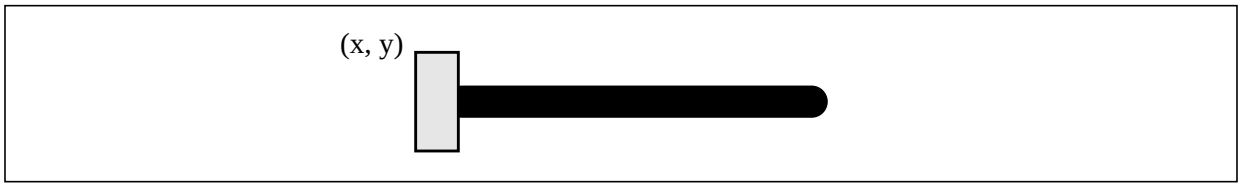
Example

```
OL_SLIDER 5000 3000 4000 1000 LEFT $P1 $R0 0 100 44 54
```

In this example:

x	=	5000
y	=	3000
w	=	4000
h	=	1000
direction	=	LEFT
value	=	$P1 $R0
low	=	0
high	=	100
trigger1	=	44
trigger2	=	54

In this example, an ol_slider is created. The upper left corner is at 5000, 3000. The dimensions of the slider are 4000 x 1000. The low limit (0) is at the right end of the horizontal slider. The high limit (100) is at the left end. The slider increases in value from right to left. The scratch pad area associated with this slider is $P1 $R0. The value read at $P1 $R0 will be displayed by the slider. As the user moves the slider control, the value at $P1 $R0 will be updated and trigger 44 will be executed. When the slider control is released, trigger 54 will be executed. See figure below:



PAGE

Description

The PAGE command defines the diagrams that are accessed by pressing the paging keys on the keyboard or paging buttons on the diagram window.

Syntax

```
PAGE page_up_diag page_right_diag page_down_diag
page_left_diag page_up_group page_right_group
page_down_group page_left_group
```

where:

page_up_diag	=	Diagram number of diagram to display when page up is selected.
page_right_diag	=	Diagram number of diagram to display when page right is selected.
page_down_diag	=	Diagram number of diagram to display when page down is selected.
page_left_diag	=	Diagram number of diagram to display when page left is selected.
page_up_group	=	Group number of group to display when page up is selected.
page_right_group	=	Group number of group to display when page right is selected.
page_down_group	=	Group number of group to display when page down is selected.
page_left_group	=	Group number of group to display when page left is selected.

Note

Groups are sets of process points that can have built-in page links. These groups are defined within the Group Builder (see "Operator WEStation User's Guide" (U0-8100) for more information on the Group Builder).

Rules

1. Only one PAGE command is allowed per diagram, and it must be in the keyboard section.
2. Valid diagram numbers are 0 through 65,535. Valid group numbers are -1 through 5,000.
3. The paging scheme for the diagram is based on the diagram number and group numbers defined in the PAGE statement as shown below:

Diagram #	Corresponding Group	Result
Positive Number	0	Displays a new diagram with the currently loaded group.
Positive Number	Positive Number	Displays a new diagram and a new group.
0	0	Displays the same diagram with a new group defined by the group paging specified for the currently loaded group.
Positive Number	-1	Displays a new diagram with a new group defined by the group paging specified for the currently loaded group.
0	Positive Number	Invalid

Example

```
PAGE 1000 0 2000 2500 0 0 -1 5000
```

This example defines the following paging:

Diagram Argument	Corresponding Group Argument	Description
1000 (UP)	0 (UP GROUP)	Paging up displays diagram 1000 and the group that is currently loaded into memory.
0 (RIGHT)	0 (RIGHT GROUP)	Paging right displays the same diagram and a new group defined by the group paging specified by the currently loaded group.
2000 (DOWN)	-1 (DOWN GROUP)	Paging down displays diagram 2000 and a new group defined by the group paging specified by the currently loaded group.
2500 (LEFT)	5000 (LEFT GROUP)	Paging left displays diagram 2500 and group 5000.

Description

The PLOT command shows the value of a process point plotted over time and scaled between a low and a high limit on the process diagram. Plots are one-dimensional (values are plotted along a straight line) as opposed to trends that are two-dimensional (a trend is an x, y graph which shows the number of values for a point and the time interval between successive values). The user specifies what field to plot for the point and how to join successive values plotted over time.

Syntax

```
PLOT x y l direction pt_name rec_fld low high plot_char
plot_char_params
```

where:

x = x coordinate for low scale value of plot line.
Valid range = 0 through 16,383.

y = y coordinate for low scale value of plot line.
Valid range = 0 through 16,383.

l = Length of plot line from low scale value to high scale.
Valid range = 1 through 16,383.

direction = Direction of plot line. The choices are:

- UP
- DOWN
- LEFT
- RIGHT

pt_name = Point name of process point to plot

rec_fld = Optional. Record field of process point to plot. If no record field is specified, the default record field for the given point type is assumed (see Section 2 for more information on default record fields).

low = Low scale value. Valid values are:

- Integers in the range -2,147,483,647 through 2,147,483,647.
- Real constants
- Point names/record fields
- Pointers/offsets

high = High scale value. Valid values are:

- Integers in the range -2,147,483,647 through 2,147,483,647.
- Real constants
- Point names/record fields
- Pointers/offsets

plot_char = Character that should be used to mark the plot values and the method of joining successive plotted values. The choices are:

- NONE — Invisible cursor position moves to plot point, but nothing visible is plotted.
- LINES — Draws a line between successive plotted values.
- SHAPE_PLOT — A specified shape from the shape library is plotted at the plot point (see "WEStation Graphics Builder User's Guide" (U0-8210) for more information on the shape library). The plot_char_params arguments are required to define the shape to display with this option.
- SQUAREWAVE — Plots a dot at each plot point and then draws horizontal and vertical line between successive plotted values (diagonal lines are not drawn).

plot_char_params = (used with plot_char = SHAPE_PLOT only)
name width height rotation inversion
(see below for description of each plot_char_param)

name = shape name from shape library

width	=	Virtual width to scale shape to. Valid range is 1-16383.
height	=	Virtual height to scale shape to. Valid range is 1-16383.
rotation	=	Degrees to rotate shape (Positive counterclockwise; Negative clockwise). Valid values are: 0,90,-90,180,-180,270,-270.
inversion	=	• NONE • TTB (top-to-bottom) • RTL (right-to-left) • BOTH (top-to-bottom & right-to-left)

Rules

1. The x and y coordinates can be relative.
2. The origin is the low limit vertex, which is different for each plot direction.
3. The process point value is always plotted along an imaginary straight line.
4. The PLOT command may be used in the background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
PLOT 5848 600 1200 DOWN A100 AV -10 10 LINES
```

In this example:

x	=	5848
y	=	600
l	=	1200
direction	=	DOWN
pt_name	=	A100
rec_fld	=	AV
low	=	-10
high	=	10
plot_char	=	LINES

This statement defines a downward line plot starting at position 5848, 600 with a display length of 1200. The value of point A100 AV is plotted over time. The scale limits are -10 to 10. Lines will be drawn joining successive plotted values over time.

Example 2

```
PLOT 5848 600 1200 DOWN A100 AV -10 10 SHAPE_PLOT ARROW 500
300 90 NONE
```

In this example:

x	=	5848
y	=	600
l	=	1200
direction	=	DOWN
pt_name	=	A100
rec_fld	=	AV
low	=	-10
high	=	10
plot_char	=	SHAPE_PLOT
plot_char _params	=	ARROW 500 300 90 NONE where: name= ARROW width=500 height=300 rotation=90 inversion= NONE

This statement defines a downward plot starting at position 5848, 600 with a display length of 1200. The value of point A100 AV is plotted over time. The scale limits are -10 to 10. The ARROW shape is used to plot the value of A100. The shape dimensions are 500x300, it is rotated 90 degrees counterclockwise, and it is not inverted.

Example 3

```
BACKGROUND
POINTER $p1 0 0
LOOP 8
PTR_VALUE $P1 $S0 0
PTR_VALUE $P1 $S2 1
PTR_MOVE $P1 ADD 4
ENDLOOP

FOREGROUND
COLOR FG RED BG DODGERBLUE2 ER DARKSEAGREEN2 OL 2
POINTER $P1 0 0
CURSOR 8424 13650
LOOP 15
PLOT [200] [0] 3838 UP $P1 $S0 0 1 SQUAREWAVE
PTR_MOVE $P1 ADD 2
ENDLOOP
```

In this example:

x	=	[200]
y	=	[0]
l	=	3838
direction	=	UP
pt_name	=	$P1
rec_fld	=	$S0
low	=	0
high	=	1
plot_char	=	SQUAREWAVE

In this example, the LOOP command in the background section stores zeros and ones in Segment 0 starting at offset 0. The LOOP command in the foreground section defines a series of upward squarewave plots which start at a position relative to the cursor command's x,y coordinates. The length of the plot is 3838, and the scale limits are from 0 to 1.

Description

The POINTER command is used to initialize the value of a pointer variable to a specific address within the scratch pad area at the Operator WEStation.

Syntax

```
POINTER pointer_name segment offset
```

where:

pointer_name	=	$P or $H names. Valid range = $P1 through $P99 and $H1 through $H99. See Section 2 for more information on pointers.
segment	=	Memory segment in the Operator WEStation to which to point. Valid range = 0 through 255 (may be relative)
offset	=	Offset into memory segment in the Operator WEStation to which to point. See Rules for valid ranges.(may be relative)

Rules

1. The user can place data into the memory location addressed by the pointer with the PTR_VALUE command, and can increment or set the pointer with the PTR_MOVE, PTR_EQUAL, and PTR_LOOP commands.

2. The POINTER command can be used in the diagram, background, foreground, and trigger sections of a diagram.

3. Segments 0, 254, and 255 can be used by the customer. All other segments are reserved for Westinghouse use only. The segment numbers, sizes and definitions are listed below.

Segment Number	Segment Size	Description
0	1024	Customer scratch pad — formatted data.
9	2048	For use with incoming data from Intel drops (for example, DPU, VAX, etc.).
20	1320	Loops and ladders GPM buffer.
21	1372	Loops and ladders GPM buffer formatted for applicable platform.
30-49	2048	TCU buffer - batch control.
200	512	Reserved for WEStation internal processing.
207	396	$G points CRT1.
209	140	Window diagram header gcode.
211	140	Main diagram header gcode.
212	140	Subwindow diagram header gcode.
213	internal	Paging Data-Main&Control windows
214	internal	Display Data-same as display data window
224	1	$G points CRT2 — (6- and 7-level software only).

225	8192	For internal use only.
226	8192	
227	8192	
228	8192	
Segment number	**Segment Size**	**Description**
234	20	WEStation Information.
237	12	Tuning ID and point name.
238	8	Control point ID's sysid1 and sysid2.
246	512	TCU buffer — batch control.
247	4	Lock — Only compatible with 6- and 7-software level Operator Station.
252	1	Westinghouse internal scratch pad CRT2 — Only compatible with 6- and 7-software level Operator Station.
253	100	Westinghouse internal scratch pad CRT1.
254	1	Customer scratch pad CRT2 — Only compatible with 6- and 7-software level Operator Station.
255	100	Customer scratch pad CRT1.

The following Westinghouse segments have specific portions that may be of use to the user. The segment numbers, offsets and descriptions are listed below.

Segment Number	Segment Offset	Format	Description
207	0	long integer	$G1 value; algorithm or point ID for control (set by poke field and CNTL_TUNE, etc.).
211	24	long integer	Main diagram number
234	0	long integer	Display drop system id

234	4	short integer	Set when window is selected
234	6	short integer	Main diagram group number
234	8	short integer	Subwindow diagram group number
234	10	short integer	Window diagram group number
237	0	long integer	Tuning point system ID (set by CNTL_TUNE).
237	4	8-byte ASCII	Tuning point name (set by CNTL_TUNE).
238	0	long integer	selected system ID #1 for control (set by CNTRLBITS).
238	4	long integer	selected system ID #2 for control (set by CNTRLBITS).

Example

```
POINTER $P2 20 99
```

In this example:

pointer_name = $P2
segment = 20
offset = 99

This statement initializes the value of $P2 to segment 20 offset 99.

Description

The POKE_FLD command defines a rectangular area (poke field) of a diagram that is activated when selected with the cursor. Various functions will be performed when a poke field is activated, depending on the type of poke field.

Syntax

```
POKE_FLD x y w h state poke_type arguments
```

where:

x = x coordinate of upper left corner of poke field.
Valid range = 0 through 16,383.

y = y coordinate of upper left corner of poke field.
Valid range 0 through 16,383.

w = Width of poke field.
Valid range = 1 through 16,383.

h = Height of poke field.
Valid range = 1 through 16,383.

state = State. The choices are:

- ON = The poke field will be active when the diagram displays.
- OFF = The poke field will not be active when the diagram displays.

poke_type = Poke type. The choices are:

- 0 = Process point
- 2 = Diagram and group
- 3 = Application program
- 6 = Ladder diagram
- 7 = Application program
- 8 = Display window
- 9 = Operating system command
- 20 = Help
- 23 = Control

arguments = List of argument(s) specific to poke_type

The applicable arguments for each poke type are shown below:

- `POKE_FLD x y w h state 0 pt_name`

where:

pt_name = Process point name

- `POKE_FLD x y w h state 2 diag_num group_num`

where:

diag_num = Diagram number.
Valid range = 0 through 65,535.

group_num = Group number.
Valid range = 0 through 5,000.
0 means do not load a group (that is, use current group).

- `POKE_FLD x y w h state` **`3`** `prog_num`

where:

prog_num = Application program number (see Section 4).

- `POKE_FLD x y w h state` **`6`** `ladder_shape_num bit_num`

where:

ladder_shape_num = Integer, process point name/record field (record field can be of type long integer byte or short integer) representing index of a ladder shape.

bit_num = Integer, process point name/record field (record field can be of type long integer byte or short integer) representing a bit within a word.

- `POKE_FLD x y w h state` **`7`** `num_of_progs prog_num1 diag_num1 num_of_args1 arg_list1 ... prog_numN diag_numN num_of_argsN arg_listN`

where:

num_of_progs = Number of application programs to run.Valid range is 1 through 65,535.

prog_num = Application program number (see Section 4).
or
$Pn $Im pointer/offset specification

diag_num = Diagram number. Valid range = 0 through 65,535.
or
$Pn $Im pointer/offset specification.

num_of_args = Number of arguments to specify for the application program chosen (see Section 4).

arg_list = Arguments for the application program. May be any of the following depending on the specific application program.

- String(130 characters or less)
- Integer or $CONSTn
- Real or $CONSTn
- Point name and record field
- Pointer/offset
- Set or $SETn
- Status word or $STATUSn

Note

The $SETn, $STATUSn, and $CONSTn variables are used to pass set variables, status words, and integer/real constants to poke7 argument lists WITHIN macro files.

- `POKE_FLD x y w h state` **`8`** `diag_num group_num pt_name1 pt_name2 ... pt_nameN`

where:

diag_num = Diagram number.
Valid range = 0 through 65,535.

group_num = Group number.
Valid range = 0 through 5,000.
0 means do not load a group (that is, use current group).

pt_name1, pt_name2, pt_nameN = Point name (without record field).

- `POKE_FLD x y w h state 9 command_line`

where:

command_line = ASCII command string including arguments, enclosed in double quotes. String is limited to 130 characters

- `POKE_FLD x y w h state 20 "file:keyname"`

where:

file = Base name of the help file. The help file is an ASCII file residing on a directory specified by $HELPPATH. The help file has a mandatory ".info" extension. This parameter does NOT include the path or the implicit extension.

keyname = Label for the applicable entry in the help file. Keyname string is limited to 130 characters.

The entire string (file:keyname) must be enclosed in double quotes as shown in the syntax. See the applicable operating system documentation for information on creating help files. See "Operator WEStation Configuration Manual" (U0-8110) for an example of WDPF help files.

Note

Help pokes are invoked by pressing the **HELP** key on the keyboard when the mouse pointer is within the poke box (as opposed to pressing the SELECT mouse key as for all other pokes). The SELECT mouse key does not invoke this type of poke.

- `POKE_FLD x y w h state` **`23`** `set_num set_val num_of_progs prog_num1 diag_num1 num_of_args1 arg_list1 ... prog_numN diag_numN num_of_argsN arg_listN`

where:

set_num	=	Set variable number from 1 through 255 or $SETn. Note that sets are global to the current main, subscreen, and window diagrams. Set1 in the main screen is the same as set1 in the subscreen, which is the same as set1 in the window.
set_val	=	Integer value to assign to variable Valid values are -32767 through 32,767.
num_of_progs	=	Number of application programs to run. Valid values are 1 through 65,535.
prog_num	=	Application program number (see Section 4)
diag_num	=	Diagram number from 0 through 65,535.
num_of_args	=	Number of arguments to specify for the application program chosen (see Section 4).
arg_list	=	Arguments for the application program specified. Valid arguments are: • String (130 characters or less) • Integer or $CONSTn • Real or $CONSTn • Point name and record field • Pointer/offset • Set or $SETn

- Status word or $STATUSn

Rules

1. The possible poke field functions are:

   - **Poke type 0** displays information on the process point specified. See "Record Types User's Guide" (U0-0131) for more information on process points and record fields.
   - **Poke type 2** displays a specified process diagram with a specified group of points. See "Operator WEStation User's Guide" (U0-8100) for information on defining point groups through the Group Builder.
   - **Poke type 3** runs an application program specified by a program number. Section 4 for usage information.
   - **Poke type 6** selects an element (contact/coil) on a ladder diagram for control. This poke field type is reserved for Westinghouse use only.
   - **Poke type 7** runs one or more application programs specified by program number and passes any required arguments. See Section 4 for usage information.
   - **Poke type 8** displays a specified window diagram with a point group and a list of points to substitute for $W points. See Section 2 for information on $W pointers.
   - **Poke type 9** runs an operating system process defined by a command line.
   - **Poke type 20** displays help on a defined item.
   - **Poke type 23** is similar to the **Poke type 7** poke field except that the set number and set value specified here must match the set number and set value that was defined in the poke field that was used to select the device.

2. A poke field can be turned on or off with the POKE_STATE command. If a poke field is off, a box is not displayed around the poke field area, and the user cannot activate the poke by selecting it with the cursor positioning device at the Operator WEStation.

3. Help pokes (poke type 20) are invoked by pressing the **HELP** key on the keyboard when the mouse pointer is within the poke box (as opposed to pressing the SELECT mouse key as for all other pokes). The SELECT mouse key does not invoke this type of poke.

4. If more than one application program is to be run with a Poke Type 7, it should be noted that the programs are run in the order given. If application program #1(CHGDIAG) is one of these programs, it should be the last one. CHGDIAG will overwrite the current diagram (including the current poke list of programs being executed), and any successive application programs will be lost.

5. The POKE_FLD command can only be used in the keyboard section.

Examples

Example 1

```
POKE_FLD 965 14447 869 1389 ON 0 A10ALM AL
```

In this example:

x	=	965
y	=	14447
w	=	869
h	=	1389
state	=	ON
poke_type	=	0
pt_name	=	A10ALM AL

This statement defines a poke field at x, y position 965, 14447. It has a width of 869 and a height of 1389. The poke field is active when the diagram displays at the Operator WEStation. When the user clicks on this poke field, (specified as type 0), the Point Information window will appear on the Operator WEStation, displaying information on A10ALM AL. See "Operator WEStation User's Guide" (U0-8100) for more information on this window.

Example 2

```
POKE_FLD 483 14400 546 904 ON 2 2450 1200
```

In this example:

x	=	483
y	=	14400
w	=	546
h	=	904
state	=	ON
poke_type	=	2
diag_num	=	2450
group_num	=	1200

This statement defines a poke field at x, y position 483, 14400. The width of the poke field is 546, and the height is 904. When the user clicks on this poke field (specified as type 2), a new diagram, number 2450, will display with the points defined in group 1200.

Example 3

```
POKE_FLD 483 14400 546 904 ON 8 2450 1200 A100 AL250 AX050
```

This statement is similar to Example 2 (above) except that poke type 8 is used for window diagrams. In addition, this poke type allows the user to substitute $W pointers in the diagram with valid point names (in this example, A100 would be substituted for $W1, AL250 would be substituted for $W2, and AX050 would be substituted for $W3).

Example 4

```
POKE_FLD 965 14465 869 1389 ON 9 "/usr/wdpf/mmi/bin
/control_panel"
```

In this example:

x	=	965
y	=	14445
w	=	869
h	=	1389
state	=	ON
poke_type	=	9
command_line	=	"/usr/wdpf/mmi/bin/control_panel"

This statement defines a poke field at x, y position 965, 14445. It has a width of 869 and a height of 1389. When the user clicks on this poke field (specified as type 9), the program, "/usr/wdpf/mmi/bin/control_panel" will execute and will display the Control Panel window. If the control panel window is already displayed, another control panel window will pop up.

Example 5

```
POKE_FLD 965 14465 869 1389 ON 20 "helpfile:helpkey"
```

This statement is similar to Example 4, except that poke type 20 is used to display a help file. When the user positions the mouse pointer on this poke field and presses the **HELP** key on the keyboard, a help window will display. The help window will display the text associated with the ":helpkey" label in the help file named: $HELPPATH/helpfile.info. If the user clicks on this poke field (as for the other poke types), nothing will happen - the **HELP** key must be pressed!

POKE_STATE

Description

The POKE_STATE command is used to enable or disable poke fields from within the diagram source code.

Note

The initial poke state is defined in the POKE_FLD command (see POKE_FLD reference page). The POKE_STATE command can be used to change the initial poke state in conditional statements.

Syntax

```
POKE_STATE x y state
```

where:

x = x coordinate within a defined poke field.
Valid range = 0 through 16,383.

y = y coordinate within a defined poke field.
Valid range = 0 through 16,383.

state = State. The choices are:

- ON (poke field will be active when the POKE_STATE command is executed).
- OFF (poke field will not be active when the POKE_STATE command is executed).

Rules

1. When a poke field is disabled, the user cannot activate it by clicking on it.
2. The POKE_STATE command can be used in the background, foreground, and trigger sections of a diagram.

Example

```
POKE_FLD 1522 1425 869 637 ON 0 A100 AV

BACKGROUND

POKE_STATE 1522 1425 OFF
```

In this example:

x = 1522

y = 1425

state = OFF

In this example, a poke field is defined at x,y position 1522, 14245. According to the POKE_FLD statement, the poke field is active when the diagram first displays. The POKE_STATE command is used to disable this poke field.

POLYGON

Description

The POLYGON command draws a polygon defined by a set of coordinate pairs (which define the vertices of the polygon).

Syntax

```
POLYGON x1 y1 x2 y2 ... xn yn line_width line_pat
<conditional expression> fill_pat <conditional expression>
```

where:

x1	=	x coordinate for start of polygon. Valid range = 0 through 16,383.
y1	=	y coordinate for start of polygon. Valid range = 0 through 16,383.
x2	=	x coordinate of next vertex in polygon.
y2	=	y coordinate of next vertex in polygon.
x*n*	=	x coordinate for end of the polygon. The maximum number of points per polygon is 255 ($2 < n <= 255$).
y*n*	=	y coordinate for end of the polygon. The maximum number of points per polygon is 255 ($2 < n <= 255$).
line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.
line_pat	=	Line pattern name.
<line_conditional_expression>	=	Optional conditional for line pattern.
fill_pat	=	Fill pattern name.
<fill_conditional_expression	=	Optional conditional for fill pattern.

Note

See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on defining line patterns and fill patterns.

Rules

1. The x and y coordinates can be relative.
2. The origin is the first vertex of the polygon.
3. The POLYGON command automatically draws a line joining the first and last coordinate pairs.
4. The user specifies the line and fill patterns to use when drawing the polygon.
5. The line and fill pattern names are conditional parameters (the user can specify conditionals to determine which line and fill patterns are used to draw the polygon).
6. The POLYGON command may be used in the background, foreground, or trigger sections of a diagram.

Examples

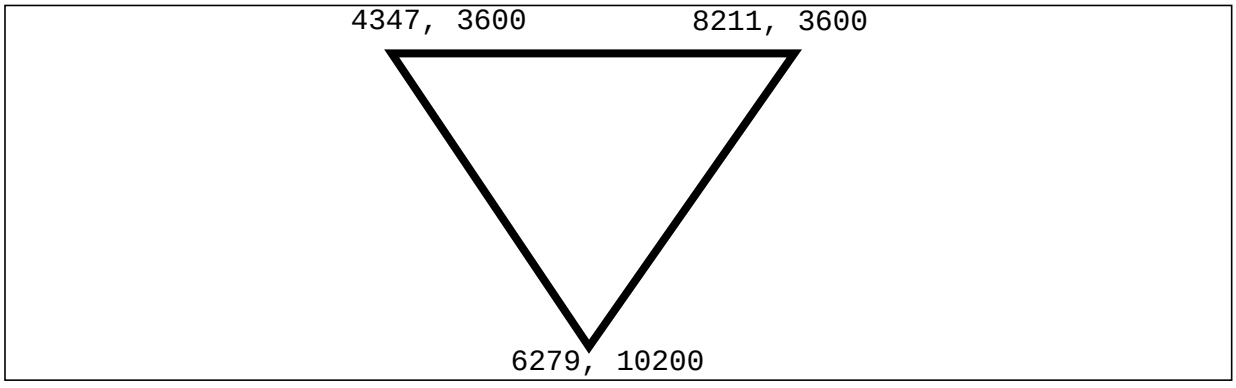
Example 1

```
POLYGON 4347 3600 8211 3600 6279 10200 4347 3600 3 solid
unfilled
```

In this example:

x1	=	4347
y1	=	3600
x2	=	8211
y2	=	3600
x3	=	6279
y3	=	10200
x4	=	4347
y4	=	3600
line_width	=	3
line_pat	=	solid
fill_pat	=	unfilled

This statement defines a polygon with a beginning point of 4347, 3600. The second vertex of the polygon is at 8211, 3600. The third vertex of the polygon is at 6279, 10200. The fourth point of the polygon is 4347, 3600. The line width is 3, the line pattern is solid, and the fill pattern is unfilled. See figure below.



Example 2 (Conditional)

```
POLYGON 4938 1724 9951 4997 7090 10070 3 solid unfilled
(QUALITY) T458 back_slash asterisks west_logo blocks
```

In this example:

x1	=	4938
y1	=	1724
x2	=	9951
y2	=	4997
x3	=	7090
y3	=	10070
line_width	=	3
line_pat	=	solid
fill_pat (good_cond_value)	=	unfilled
fill conditional expression	=	(QUALITY)
pt_name	=	T458
fair_cond_value	=	back_slash
poor_cond_value	=	asterisks
bad_cond_value	=	west_logo
timed_out_cond_value	=	blocks

The statement indicates that the fill pattern for the polygon will change based on the quality of T458 as described below:

- If T458 has good quality, the fill pattern for the polygon will be unfilled.
- If T458 has fair quality, the fill pattern for the polygon will be back_slash.
- If T458 has poor quality, the fill pattern for the polygon will be asterisks.
- If T458 has bad quality, the fill pattern for the polygon will be west_logo.
- If T458 has timed out quality, the fill pattern for the polygon will be blocks.

See Section 2 for more information on conditionals.

PROCESS_PT

Description

The PROCESS_PT command displays the contents of a process point record field (in ASCII text form) on the diagram. Optionally, this value may be displayed as an integral percentage (0 through 100) instead of as the literal value. Use the optional **low** and **high** arguments to display the value as a percentage.

Syntax

```
PROCESS_PT x y number_of_chars decimal_places format
quality orientation font_type pt_name rec_fld
<conditional_expression> <low high>
```

where:

x = The x coordinate for the baseline position of the first character of the process point display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the process point display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of the first character of the process point display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the process point display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

number_of_chars = Maximum width of process point display.
Valid range = 1 through 80

decimal_places = Number of decimal places to be displayed. (If the value in the FM record field should be used, enter -1).
Valid range = -1 through (number_of_chars - 1) (that is, if number_of_chars is 10, valid range is -1 through 9).

format = Format for display of process point value. The choices are:

- RIGHT — Right justified; left padded with blanks.
- RIGHT0 — Right justified; left padded with zeros.
- LEFT — Left justified; right padded with blanks.
- HEX — Hexadecimal format (base 16) without trailing "H".
- HEX_H—Hexadecimal format (base 16) with trailing "H" included in width.
- BINARY — Binary format (base 2).
- EXPONENTIAL — Exponential format.
- TECHNICAL — exponential format where exponent is divisible by 3.

quality = Quality. The choices are:

- ON — displays a quality character with the process point.
- OFF — does not display a quality character with the process point.

orientation = Orientation. The choices are:

- HORZ (horizontal)

- VERT (vertical)

font_type = One of the following font types:

- BITMAP (followed by bitmap_font_number)
- BITMAP_OVER (followed by bitmap_font_number)
- VECTOR (followed by w, h, line_width)
- VECTOR_OVER (followed by w, h, line_width)

See Section 3-2 for more information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number.
Valid range = 1 through 8.

w = Width of vector character cell in pixels.
Valid range = 3 through 16,383.

h = Height of vector character cell in pixels.
Valid range = 3 through 16,383.

line_width = Line width.
Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

pt_name = Point name to be displayed

rec_fld = Record field

<conditional _ expression> = Optional conditional for process point record field.

<low high> = Optional low/high percent values.
Valid range = -2,147,483,647 through 2,147,483,647.
low = minimum process point data value
high = maximum process point data value

Rules

1. The x and y coordinates can be relative.
2. The origin will be one of the following:
   - Baseline position of the first character of the text string (for bitmap).
   - Upper left corner of the first character of the text string (for vector).
3. The contents of the record field may or may not be displayed as a percentage. To display the value as a percentage, specify the low and high limits (**<low high>**). The user must know the range of possible values for the record field to use this option. The **low** limit is the minimum of the range of values possible for the given record field; the **high** limit is the maximum of the range. The formula for calculating the percentage from the actual value and the low and high limits is:
   percentage = ((actual_value - low) / (high - low)) * 100
4. To display the actual record field contents instead of the percentage, omit the **low** and **high** parameters.
5. The record field is a conditional parameter (the user may specify a conditional to determine which record field value is displayed).
6. The PROCESS_PT command may be used in the background, foreground, or trigger sections of a diagram.

Examples

Example 1

```
PROCESS_PT 9620 1794 6 2 RIGHT0 ON HORZ VECTOR 321 734 1
A300 AV -5 10
```

In this example:

x	=	9620
y	=	1794
number_of_chars	=	6
decimal_places	=	2
format	=	RIGHT0
quality	=	ON
orientation	=	HORZ
font_type	=	VECTOR
w	=	321
h	=	734
line_width	=	1
pt_name	=	A300
rec_fld	=	AV
low scale	=	-5
high scale	=	10

This command displays the value of A300 AV as a percentage(0-100) at 9620, 1794. The value is displayed in a field width of 6 with 2 decimal places. The minimum possible value for A300 AV is -5; the maximum value for A300 AV is 10. The user must know the possible range for A300 AV to display the value as a percentage. A quality character is used. Note that 1 place of the 6 character field width will be used to show quality - which leaves 5 actual characters to display the value as a percentage. The process point percentage displays horizontally and is right justified, left-padded with zeros (RIGHT0 format). Vector text is used with a width and height of 321, 734. The line width is 1.

In this example, if the current value of A300 AV = -5, the percentage displayed would be 0. If the current value of A300 AV = 10, the percentage displayed would be 100. If the current value of A300 AV = 6, the percentage displayed would be 73.33.

Example 2 (Conditional)

```
PROCESS_PT 688 7800 25 2 HEX OFF VERT BITMAP 5 PB500 ID
((PB500 LL > 50) AND (PB010 <= PB500 LL)) HL
```

In this example:

x	=	688
y	=	7800
number_of_chars	=	25
decimal_places	=	2
format	=	HEX
quality	=	OFF
orientation	=	VERT
font_type	=	BITMAP
bitmap_font_number	=	5
pt_name	=	PB500
rec_fld	=	ID
conditional_expression	=	((PB500 LL > 50) AND (PB010 <= PB500 LL)) HL

This statement displays the contents of PB500 ID located at 688, 7800. The width of the process point is 25, and it has 2 decimal places. A quality character is not used. The process point displays vertically in hexadecimal format. The hexadecimal symbol ("H") is NOT appended to the value display. Bitmap text (with a bitmap font number of 5) is used.

A conditional is used to determine which record field value is displayed. If PB500 LL > 50 AND PB010 <= PB500 LL, the record field will be HL. Otherwise, the record field will be ID.

PTR_EQUAL

Description

The PTR_EQUAL command is used to set the value of one pointer variable to that of another pointer variable. The pointer variables point to a specified address within the Operator WEStation memory. PTR_EQUAL changes location as opposed to PTR_VALUE which changes data (see PTR_VALUE reference page).

Syntax

```
PTR_EQUAL pointer_name1 pointer_name2
```

where:

pointer_name1	=	$P or $H pointer name. Valid range = $P1 through $P99 and $H1 through $H99. Section 2 for more information on pointers.
pointer_name2	=	$P or $H pointer name. Valid range = $P1 through $P99 and $H1 through $H99. See Section 2 for more information on pointers.

Rules

1. Only $P or $H pointer can be used with this command (either all $P pointers or all $H pointers or a combination of $P and $H pointers can be used).

2. The PTR_EQUAL command can be used in the background, foreground, and trigger sections of a diagram.

Example

```
POINTER $P16 9 15

PTR_EQUAL $P3 $P16
```

In this example:

pointer_name1 = $P3

pointer_name2 = $P16

The PTR_EQUAL statement in this example sets pointer $P3 to equal $P16, so that pointer $P3 will point to the same area as pointer $P16, which has been set to segment 9, offset 15.

PTR_LOOP/P_ENDLP

Description

The PTR_LOOP/P_ENDLP commands operate as a pair. These commands are used to define a set of graphic commands that are executed a given number of times. The number of times that the loop is executed is determined by the record field value of a point or by a number addressed by a pointer variable and an offset. When the user enters the PTR_LOOP command in the source editor, a corresponding P_ENDLP command is automatically displayed on the next line.

Syntax

```
PTR_LOOP pt_name rec_fld

.
.

P_ENDLP
```

where:

pt_name	=	Point name or standard pointer name
rec_fld	=	Non-ASCII record field or pointer offset representation

Rules

1. A non-ASCII record field is required for the *pt_name* argument. For pointers, this means that $In, $Bn, $Sn, and $Rn offsets are supported, but $AnXl offsets are not. It is the user's responsibility to supply the correct value for n for the given offset (for example, n must be a multiple of 4 for $R offsets). See **Record Fields** in Section 2 for the rules for defining valid pointer offsets.

2. The PTR_LOOP/P_ENDLP commands may be used in the background, foreground, or trigger sections of a diagram.

Example

```
POINTER $P1 9 14

POINTER $P2 9 0

PTR_LOOP $P1 $B0

   COLOR FG YELLOW BG WHITE

   PROCESS_PT 234 123 10 2 RIGHT ON HORZ VECTOR 321 734 1
   $P2 $S0 0 5

   PROCESS_PT 1627 387 5 0 RIGHT ON BITMAP 2 $P2 $S2 -5 5

   PROCESS_PT 1523 2532 5 0 RIGHT ON BITMAP 3 $P2 $S4 0 10

   PROCESS_PT 1672 384 10 2 RIGHT0 ON BITMAP 2 $P2 $A6X8 0 5

   POINTER $P2 0 +42

P_ENDLP
```

In this example:

pt_name = $P1

rec_fld = $B0

This example assumes that a general message has been sent to a drop and that block data has been returned in Segment 9 at Byte Offset 0.

The data has the following structure:

Segment	Byte Offset		Values
9	$P2	0	Integer value
		2	Integer value
		4	Integer value
		6	ASCII string
		8	ASCII string
		10	ASCII string
		12	ASCII string
	$P1	14	number of loops
			.
			.
			.

This loop will be executed the number of times specified in $P1 $B0. The value of the first process point is obtained from $P2 $S0 (Bytes 0 and 1 for the first loop). The value of the second process point is obtained from $P2 $S2 (Bytes 2 and 3 for the first loop). The value of the third process point is obtained from $P2 $S4 (Bytes 4 and 5 for the first loop). The value of the fourth process point is obtained from $P2 $A6X8 (Bytes 6 through 13 for the first loop). The POINTER command in the loop increments pointer $P2 to Byte Offset 42. Therefore, for the second loop, the process point values are obtained from Bytes 42 through 55.

Description

The PTR_MOVE command is used to increase or decrease the offset of a given pointer variable so that the pointer addresses a new memory location. It allows the user to change the current pointer offset by an integral value, using mathematical operations (addition, subtraction, and multiplication).

Syntax

```
PTR_MOVE pointer_name operation operand1 operand2 operand3
```

where:

pointer_name	=	$P or $H pointer names. Valid ranges = $P1 through $P99 and $H1 through $H99. See Section 2 for more information on pointers.
operation	=	Mathematical operation. Choose one of the following: • ADD (addition) • SUB (subtraction) • MULT (multiplication)
operand1 operand2 operand3	=	Operands that perform actions depending upon the selected operation. See Rules below.

Rules

1. The following actions occur, depending upon the operation selected:

   ADD = Value of the specified point and record field value, or the specified integer value, is added to the offset of the specified pointer.

   Operand1 = Point name/ record field (non-ASCII), pointer/ offset (non-ASCII), or an integer between -2,147,483,647 and 2,147,483,647.

   Operand2 = Not used

   Operand3 = Not used

   SUB = Value of the specified point name and record field, or the specified integer value, is subtracted from the offset of the specified pointer.

   Operand1 = Point name/record field (non-ASCII), pointer/ offset (non-ASCII), or an integer between -2,147,483,647 and 2,147,483,647.

   Operand2 = Not used

   Operand3 = Not used

   MULT = Value of the specified point name and record field, or specified integer, is multiplied by a user-defined multiplier and then added or subtracted from the specified pointer.

   Operand1 = Point name/ record field (non-ASCII), pointer/ offset (non-ASCII), or an integer between -2,147,483,647 and 2,147,483,647.

   Operand2 = ADD or SUB

   Operand3 = Multiplier.Valid range = 0 through 255.

2. Only $P and $H pointer types are valid for this command.

3. Non-ASCII record fields are required for all point/pointer operands. For pointer operands, this means that $In, $Bn, $Sn, and $Rn offsets are supported, but $AnXl offsets are not. It is the user's responsibility to supply the correct value for n for the given offset (for example, n must be a multiple of 4 for $R offsets). See **Record Fields** in Section 2 for the rules for defining valid pointer offsets.

4. The PTR_MOVE command can be used in the background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
POINTER $P16 7 12

PTR_VALUE $P2 $I0 2

PTR_MOVE $P16 SUB $P2 $I0
```

In this example:

pointer_name = $P16
operation = SUB
operand1 = $P2 $I0

Pointer $P16 has a value of $P2 $I0 subtracted (12 - 2 = 10) so its new location is 7, 10.

Example 2

```
PTR_MOVE $P16 MULT $P2 $I0 SUB 2
```

In this example:

pointer_name = $P16
operation = MULT
operand1 = $P2 $I0
operand2 = SUB
operand3 = 2

This statement multiplies the integer at $P2 $I0 by 2 and subtracts the value from the current offset of $P16.

Description

The PTR_VALUE command is used to change the data addressed by a pointer variable to a new value. PTR_VALUE changes data as opposed to PTR_EQUAL which changes location (see PTR_EQUAL reference page for more information).

Syntax

```
PTR_VALUE pointer_name offset value
```

where:

pointer_name = $P pointer name.
Valid range is $P1 through $P99.

offset = Non-ASCII pointer record field ($Bn, $In, $Rn, $Sn).

value = One of the following:

- Point name and non-ASCII record field.
- Pointer name and non-ASCII pointer record field.
- Integer value.
- Real value.

Rules

1. A point name/record field value causes the value of that point to be moved to the pointer record field location. A pointer/offset value causes the value pointed to by the pointer to be moved to the pointer record field location.
2. Relative values can be used to increase or decrease the current value at the pointer record field location.

3. Non-ascii record fields are required for the offset and for point/pointer value arguments. For the offset and pointer type value, this means that $In, $Bn, $Sn, and $Rn offsets are supported, but $AnXl offsets are not. It is the user's responsibility to supply the correct value for n for the given offset (for example, n must be a multiple of 4 for $R offsets). See **Record Fields** in Section 2 for the rules for defining valid pointer offsets.

4. The PTR_VALUE command can be used in the background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
PTR_VALUE $P15 $I0 A100 AS
```

In this example:

pointer_name = $P15
offset = $I0
value = A100 AS

This statement stores the value of A100 AS in the integer at $P15 $I0.

Example 2

```
PTR_VALUE $P15 $I0 [5]
```

In this example:

pointer_name = $P15
offset = $I0
value = [5]

This statement adds 5 to the current integer value at $P15 $I0.

Description

The RECTANGLE command displays a rectangle on the diagram.

Syntax

```
RECTANGLE x y w h line_width line_pat <line_conditional_
expression> fill_pat <fill_conditional_expression>
```

where:

x	=	x coordinate of upper left corner of the rectangle. Valid range = 0 through 16,383.
y	=	y coordinate of upper left corner of the rectangle. Valid range = 0 through 16,383.
w	=	Width of the rectangle. Valid range = 1 through 16,383.
h	=	Height of the rectangle. Valid range = 1 through 16,383.
line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.
line_pat	=	Line pattern name.
<line_conditional_ expression	=	Optional conditional for line pattern.
fill_pat	=	Fill pattern name.
<fill_conditional_ expression	=	Optional conditional for fill pattern.

Note

See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on defining line patterns and fill patterns.

Rules

1. The x and y coordinates can be relative.
2. The origin is the upper left corner of the rectangle.
3. The user specifies the line and fill patterns for the rectangle.
4. The line and fill pattern names are conditional parameters (the user may specify conditionals to determine which line or fill pattern to use to draw the rectangle).
5. The RECTANGLE command may be used in the background, foreground, or trigger sections of a diagram.

Examples

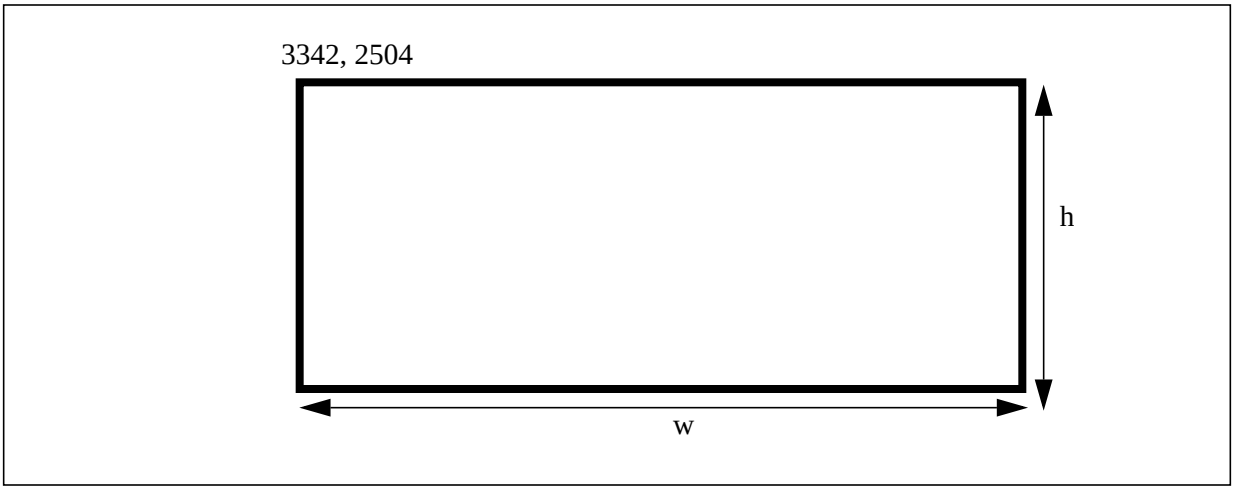
Example 1

```
RECTANGLE 3342 2504 6534 8141 3 solid unfilled
```

In this example:

x coordinate	=	3342
y coordinate	=	2504
w	=	6534
h	=	8141
line_width	=	3
line_pat	=	dot_dash
fill_pat	=	unfilled

This statement defines a rectangle located at 3342, 2504. The width and height of the rectangle is 6534, 8141. The line width is 3 and the line pattern is solid. The rectangle is unfilled.



Example 2 (Conditional)

```
RECTANGLE 10898 8378 4119 2471 8 dotted solid { (AI101 BB <=
50.5) asterisks (D200 DS = RESET) back_slash }
```

In this example:

x coordinate	=	10898
y coordinate	=	8378
w	=	4119
h	=	2471
line_width	=	8
line_pat	=	dotted
fill_pat	=	solid
fill_conditional_ expression	=	{(AI101 BB <= 50.5) asterisks (D200 DS = RESET) back_slash}

This statement defines a rectangle located at 10898, 8378. The width and height of the rectangle is 4119, 2471. The line width is 8 and the line pattern is dotted.

A fill pattern conditional is specified. If AI101 BB <= 50.5, the fill pattern will be asterisks. If the first conditional evaluates to false, the second conditional will be considered. If D200 DS = RESET, the fill pattern will be back_slash. If the second conditional evaluates to false, the fill pattern will be solid (default).

See Section 2 for more information on conditionals.

Description

The RUN_PROGRAMS command is used to perform a poke type 7 function from within the background, foreground, and/or trigger sections of a diagram. There is no display associated with this command. This command is used to execute one or more application programs during graphic execution without any user interface.

Notes

- See POKE_FLD reference page for more information on poke type and poke arguments.
- See Section 4 for information on application programs.

Syntax

```
RUN_PROGRAMS programs

prog_1 diag_1 #args_1 arg1 ... arguments_1

prog_2 diag_2 #args_2 arg2 ... arguments_2

...

prog_n diag_n #args_n argn ... arguments_n
```

where:

programs	=	Number (quantity) of application programs to run.
prog_1, prog_2, ...prog_n	=	Application program number, or $Pn $Im pointer. See Section 4 for a list of valid application programs.
diag_1, diag_2, ...diag_n	=	Diagram associated with application program, or $Pn $Im pointer Valid range = 0 through 65,535 (0 = no associated diagram).

#args_1,#args_2,...#args_n	=	Number of arguments passed to the associated application program. See Section 4 for more information.
arguments	=	Arguments passed to the associated application program. Valid arguments consist of the following: • String (enclosed in single or double quotes) • Integer, or $CONSTn • Real, or $CONSTn • Point name and record field • Pointer and offset • Set variable, or $SETn • Status word, or $STATUSn

Rule

1. RUN_PROGRAMS command is valid in the background, foreground, and trigger sections of the diagram.

Example

```
RUN_PROGRAMS 2 117 8104 5 0 79 27 0 0 6 0 5 A004Z004 ID
A004Z004 ID 1 9 2
```

In this example:

programs	=	2
prog_1	=	117
diag_1		8104
#args_1		5
arguments_1		0, 79, 27, 0, 0
prog_2	=	6
diag_2		0
#args_2		5
arguments_2		A004Z004 ID, A004Z004 ID, 1, 9, 2

In this example, 2 application programs are being run. The first program is 117. Diagram 8104 will be displayed for this program. There are 5 arguments passed to program 117. The second program is 6. No diagram is required for program 6. It also requires 5 arguments.

SETVAL

Description

The SETVAL command is used to set the value of the global variable setx (where x = 1 through 255). Note that sets are global to the current main, subscreen, and window diagrams. Set1 in the main screen is the same as set1 in the subscreen, which is the same as set1 in the window.

Syntax

```
SETVAL set value
```

where:

set	=	Set variable number. Valid range = 1 through 255. or $SET variable. Valid range = $SET1 through $SET255.
value	=	Integer value to assign to variable. Valid range = 0 through 32,767.

Rules

1. The SETVAL command may be used in the background, foreground, or trigger sections of a diagram.
2. The $SET arguments are used to pass set numbers into macros. If the SETVAL command resides in a macro graphic, and the user desires to pass the set number into the macro instead of hard-coding the set number, the user enters a $SET argument for the set number (the $SET arguments function like $D points in that they are only meaningful within macro files).

Example

```
SETVAL 10 4

COLOR FG magenta (SET10) 4 blue green yellow cyan BG white
```

In this example:

set = 10

value = 4

In this example, the SETVAL statement initializes the value of SET10 to 4. The SET condition in the COLOR statement specifies that the foreground color is yellow. This is determined as follows:

- If the value of SET10 = 1 or > 5, the foreground color is magenta.
- If the value of SET10 = 2, the foreground color is blue.
- If the value of SET10 = 3, the foreground color is green.
- If the value of SET10 = 4, the foreground color is yellow.
- If the value of SET10 = 5, the foreground color is cyan.

SHAPE

Description

The SHAPE command draws a predefined display item from the shape library on the diagram, scaling the shape into the dimensions of a user-specified rectangle. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on the shape library.

Syntax

```
SHAPE x y w h shape_name <conditional_expression> rotation
inversion
```

where:

x = x coordinate of position at which to place the shape's origin. See "WEStation Graphics Builder User's Guide" (U0-8210) for information on defining the shape's origin.
Valid range = 0 through 16,383.

y = y coordinate of position at which to place the shape's origin. See "WEStation Graphics Builder User's Guide" (U0-8210) for information on defining the shape's origin.
Valid range = 0 through 16,383.

w = Width of outlining rectangle into which to scale the shape. See Section 3-2 for more information on outlining rectangles.
Valid range = 1 through 16,383.

h = Height of outlining rectangle into which to scale the shape. See Section 3-2 for more information on outlining rectangles.
Valid range = 1 through 16,383.

shape_name = Shape name from shape library. See "WEStation Graphics Builder User's Guide" (U0-8210) for more information on the shape library.

<conditional_ expression>	=	Optional conditional for shape name.
rotation	=	Degrees to rotate the shape. The choices are: -270, -180, -90, 0, 90, 180, 270. (Minus degrees (-270, -180, -90) will result in clockwise rotation. Plus degrees (90, 180, 270) will result in counterclockwise rotation.
inversion	=	Inversion types. Choose one of the following: • NONE — no inversion • RTL — invert right to left • TTB — invert top to bottom • BOTH — invert shape right to left and top to bottom

Rules

1. The x and y coordinates can be relative.
2. Shapes have a defined origin. See "WEStation Graphics Builder User's Guide" (U0-8210) for information on defining a shapes' origin.
3. The shape may be rotated clockwise or counterclockwise in 90 degree increments.
4. The shape will be rotated or inverted according to the specified parameters before it is scaled into the specified rectangle.
5. The shape name is a conditional parameter (the user may specify a conditional to determine which shape to display on the screen).
6. The current erase color (ER) is used by the process diagram system to erase the existing shape when conditional shapes are used and the shape changes. It is the user's responsibility to control the background area on which the conditional shapes are displayed and the erase color such that the old shapes are erased correctly when new shapes are displayed based on some condition.
7. The SHAPE command may be used in the background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
SHAPE 5291 1794 4336 8962 VALVE 90 RTL
```

In this example:

x	=	5291
y	=	1794
w	=	4336
h	=	8962
shape name	=	VALVE
rotation	=	90
inversion	=	RTL

This statement draws the shape VALVE on the diagram. The origin of the shape is at 5291, 1794. The width and height of the shape's outlining rectangle is 4336, 8962. The shape is rotated 90 degrees counterclockwise and inverted right-to-left.

Example 2 (Conditional)

```
SHAPE 11214 4119 3071 6701 PUMP (CASE) T374H IL 1 1 2 VALVE
TANK 0 NONE
```

In this example:

x	=	11214
y	=	4119
w	=	3071
h	=	6701
shape name	=	PUMP
conditional_expression	=	(CASE) T374H IL 1 1 2 VALVE TANK
rotation	=	0
inversion	=	NONE

This statement draws the shape PUMP on the diagram. The origin of the shape is at 11214, 4119. The width and height of the shape's outlining rectangle is 3071, 6701. The shape will not be rotated or inverted.

A shape name conditional is used in this example. The shape for the diagram will change based on the value of T374H IL as described below:

- If T374H IL < 2, the shape will be PUMP.
- If T374H IL >= 2 and if T374H IL < 3, the shape will be VALVE.
- If T374H IL >= 3 and if T374H IL < 4, the shape will be TANK.
- If T374H IL >= 4, the shape will be PUMP.

See Section 2 for more information on conditionals.

SW_ALARM_COUNT

Description

The SW_ALARM_COUNT command defines an area of the diagram for displaying the number of unacknowledged alarms for a given alert list. This command is only applicable for drops that are configured to run the Alarm Management System (see "Alarm Management System User's Guide" (U0-8115) for information on this system). To determine this value, a call to an Alarm Management System Library function is performed.

Syntax

```
SW_ALARM_COUNT x y number_of_chars orientation font_type
alert_list_name
```

where:

x = The x coordinate for the baseline position of the first character of the alarm count display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the alarm count display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of the first character of the alarm count display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the alarm count display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

number_of_chars = Maximum width of alarm count display.
Valid range = 1 through 5.

orientation = Orientation. The choices are:

- HORZ (horizontal)
- VERT (vertical)

font_type = One of the following font types:

- BITMAP (followed by bitmap_font_number)
- BITMAP_OVER (followed by bitmap_font_number)
- VECTOR (followed by w, h, line_width)
- VECTOR_OVER (followed by w, h, line_width)

See Section 3-2 for more information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number.
Valid range = 1 through 8.

w = Width of vector character cell in pixels.
Valid range = 3 through 16,383.

h = Height of vector character cell in pixels.
Valid range = 3 through 16,383.

line_width = Line width.
Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

alert_list_name = ASCII string representing the alert list name, enclosed in double or single quotes. See Alarm Management documentation for more information on alert lists.
Maximum number of characters = 5 (see Rules below).

Rules

1. The x and y coordinates can be relative.
2. The origin will be one of the following:
   - Baseline position of the first character of the text string (for bitmap).
   - Upper left corner of the first character of the text string (for vector).
3. The alert list name can consist of a maximum of five characters where:
   - Character 1 can be A through Z or a dash.
   - Characters 2 through 5 can be A through Z, 0 through 9, a dash, or a blank.
4. SW_ALARM_COUNT is only used in the Alarm Management System. See "Alarm Management System User's Guide" (U0-8115) for more information.
5. The SW_ALARM_COUNT command can be used in the background, foreground, or trigger sections of a diagram.

Example

```
SW_ALARM_COUNT 2222 3333 5 HORZ VECTOR 413 2059 1 "LIST2"
```

In this example:

x	=	2222
y	=	3333
number_of_chars	=	5
orientation	=	HORZ
font_type	=	VECTOR
w	=	413
h	=	2059
line_width	=	1
alert_list_name	=	"LIST2"

In this example, the alarm count value will display at x, y location 2222, 3333. The number of characters in the alarm count is 5. It will display horizontally, in vector text. The width of the text is 413 and the height is 2059. The line width is 1. The alert list name from which the count is taken is "LIST2"

SW_ALARM_CUE

Description

The SW_ALARM_CUE command defines an area of the process diagram for displaying the priority of an alarm for a defined process point. An alarm cue symbol displays the priority and status of an alarm through colors and shapes. This command is only applicable for drops that are configured to run the Alarm Management System (see "Alarm Management System User's Guide" (U0-8115) for information on this system).

Syntax

```
SW_ALARM_CUE x y w h pt_name
```

where:

x	=	x coordinate of the upper left corner of the symbol. Valid range = 0 through 16,383.
y	=	y coordinate of the upper left corner of the symbol. Valid range = 0 through 16,383.
w	=	Width of the alarm cue field. Valid range = 1 through 16,383.
h	=	Height of the alarm cue field. Valid range = 1 through 16,383.
pt_name	=	Point name. Must be a DL, DI, BG, DC, DU, AI, AL, AV, or BN point record type.

Rules

1. The x and y coordinates can be relative.
2. The origin is the upper left corner of the symbol.
3. The alarm cue symbol represents the priority and status of an alarm through color and shapes:

   No square = normal

   ■ orange filled square = alarm acknowledge alert

   ■ orange filled square (blinking) = alarm unacknowledged alert

   ▲ red filled triangle = alarm acknowledge priority

   ▲ red filled triangle (blinking) = alarm unacknowledged alert

   S orange square with the letter "S" in it = shelved

   R orange square with the letter "R" in it = released

4. SW_ALARM_CUE is only used in Alarm Management System. See "Alarm Management System User's Guide" (U0-8115) for more information.
5. The SW_ALARM_CUE command can be used in the background, foreground, and trigger sections.

Example

```
SW_ALARM_CUE 1536 4709 1638 2401 DI100
```

In this example:

x	=	1536
y	=	4709
w	=	1638
h	=	2401
pt_name	=	DI100

This example displays an alarm cue symbol for point DI100. The symbol will be located at x, y position 1536, 4709. The alarm cue field is 1638 pixels wide and 2401 pixels high.

Description

The SW_ALERT_LIST command defines an area of the process diagram for displaying the priority of the worst alarm in the group of points that the Alert List represents. This command is only applicable for drops that are configured to run the Alarm Management System (see "Alarm Management System User's Guide" (U0-8115) for information on this system). To determine the status of an alarm, a call to an Alarm Management System library function is performed. An alarm cue symbol displays the priority and status of an alarm (through colors and shapes) for a specified alert list.

Syntax

```
SW_ALERT_LIST x y w h alert_list_name
```

where:

x	=	x coordinate of the upper left corner of the symbol. Valid range = 0 through 16,383.
y	=	y coordinate of the upper left corner of the symbol. Valid range = 0 through 16,383.
w	=	Width of the alarm cue field. Valid range = 1 through 16,383.
h	=	Height of the alarm cue field. Valid range = 1 through 16,383.
alert_list_name	=	ASCII string representing the alert list name, enclosed in double or single quotes. Maximum number of characters = 5 (see Rules below).

Rules

1. The x and y coordinates can be relative.
2. The origin is the upper left corner of the symbol.
3. The alarm cue symbol represents the priority and status of the alarm through color and shapes:

   □ orange unfilled square = normal

   ■ orange filled square = alarm acknowledge alert

   ■ orange filled square (blinking) = alarm unacknowledged alert

   ▲ red filled triangle = alarm acknowledge priority

   ▲ red filled triangle (blinking) = alarm unacknowledged alert

4. The alert list name can consist of a maximum of five characters where:
   - Character 1 can be A through Z or a dash.
   - Characters 2 through 5 can be A through Z, 0 through 9, a dash, or a blank.
5. SW_ALERT_LIST is only used in Alarm Management System. See "Alarm Management System User's Guide" (U0-8115) for more information.
6. The SW_ALERT_LIST command can be used in the background, foreground, or trigger sections of a diagram.

Example

```
SW_ALERT_LIST 13339 1247 689 3001 "LIST1"
```

In this example:

x	=	13339
y	=	1247
w	=	689
h	=	3001
alert_list_name	=	"LIST1"

This example displays an alarm cue symbol for alert list "LIST1". The symbol is located at x, y position 13339, 1247. The width of the alert list field is 689 and the height is 3001.

SW_PROCESS_PT

Description

The SW_PROCESS_PT (also known as super analog process point) command defines an area of the process diagram for displaying the contents of a process point record, along with an alarm cue, a limit transgressed marker, and a discrepancy marker (optional; a discrepancy marker is a digital point that indicates if there is a value discrepancy). This command is only applicable for drops that are configured to run the Alarm Management System (see "Alarm Management System User's Guide" (U0-8115) for information on this system).

Syntax

```
SW_PROCESS_PT x y number_of_chars decimal_places format
orientation font_type pt_name discr_marker
```

where:

x = The x coordinate for the baseline position of the first character of the process point display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the process point display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of the first character of the process point display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the process point display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

number_of_chars = Maximum width of super analog process point display.
Valid range = 1 through 30.

decimal_places = Number of decimal places to be displayed. (If the value in the FM record field should be used, enter -1.)
Valid range = -1 through (number_of_chars - 1). (for example, if number_of_chars is 10, the valid range is -1 through 9.

format = Format for display of process point value. The choices are:

- RIGHT = Right justified; left padded with blanks
- RIGHT0 = Right justified; left padded with zeros
- LEFT = Left justified; right padded with blanks
- HEX = Hexadecimal format (base 16)
- BINARY = Binary format (base 2)
- EXPONENTIAL = Exponential format
- TECHNICAL = Technical format

orientation = Orientation. The choices are:

- HORZ (horizontal)
- VERT (vertical)

font_type = One of the following font types:

- BITMAP (followed by bitmap_font_number)
- BITMAP_OVER (followed by bitmap_font_number)
- VECTOR (followed by w, h, line_width)
- VECTOR_OVER (followed by w, h, line_width)

See Section 3-2 for more information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number. Valid range = 1 through 8.

w = Width of vector character cell in pixels. Valid range = 3 through 16,383.

h = Height of vector character cell in pixels. Valid range = 3 through 16,383.

line_width = Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

pt_name = Analog point name linked to a point value field (must be an AI, AL, or AB point record type; do not enter the record field in the source editor).

discr_marker = Optional digital point name (must be DI, DL, DC, or DM point record type) indicating value discrepancy.

Rules

1. The x and y coordinates can be relative.
2. The origin can be one of the following:
   - Baseline position of the first character of the text string (for bitmap text).
   - Upper left corner of the first character of the text string (for vector text).
3. The alarm cue symbol displays the priority and status of an alarm through color and shapes:

   □ orange unfilled square = normal

   ■ orange filled square = alarm acknowledge alert

   ■ orange filled square (blinking) = alarm unacknowledged alert

   ▲ red filled triangle = alarm acknowledge priority

   ▲ red filled triangle (blinking) = alarm unacknowledged alert

   S orange square with the letter "S" in it = shelved

   R orange square with the letter "R" in it = released
4. The limit transgressed symbol indicates the severity of the exceeded limit for the specified point:

   ⩓ = high high alarm

   ^ = high alarm

   v = low alarm

   ⩔ = low low alarm
5. The optional digital point indicates if there is a value dispcrepancy.

6. The SW_PROCESS_PT command will display the items in the following order:

   <discrepancy marker> <alarm cue> <limit transgressed marker>
   < analog process point value>

   The process point string displays on the Graphics Builder interactive editor as the string "P^" or "?P^" (the string begins with a "?" only if a discrepancy marker is defined) followed by a series of "????". On the process diagram, the letter "P" is replaced by one of the alarm cue symbols, the ^ represents a limit transgressed symbol, and the "???" are replaced by the actual value in the record field.

7. SW_PROCESS_PT is only used in Alarm Management System. See "Alarm Management System User's Guide" (U0-8115) for more information.

8. The SW_PROCESS_PT command can be used in the background, foreground, and trigger sections of a diagram.

Example

```
SW_PROCESS_PT 1449 8400 10 2 RIGHT0 HORZ VECTOR_OVER 413 2059
1     A100 D100
```

In this example:

x	=	1449
y	=	8400
number_of_chars	=	10
decimal_places	=	2
format	=	RIGHT0
orientation	=	HORZ
format	=	VECTOR_OVER
w	=	413
h	=	2059
line_width	=	1
pt_name1	=	A100
pt_name2	=	D100

This statement displays the contents of analog point A100, an alarm cue symbol, and a limit transgressed symbol for A100. The discrepancy marker for point A100 is point D100. The process point display is positioned at the x, y location 1449, 8400 and is 10 characters long. Two decimal places are used in the display. The format will be RIGHT0 and will appear horizontally. The width of the process point display is 413 and the height is 2059. The line width is 1.

TEXT

Description

The TEXT command displays a text string on the diagram. The text string can be literally specified in this command, or it may be read from the active text group file. The text group file is an external ASCII file which contains up to 100 user-defined strings. See "Operator WEStation Configuration Manual (U0-8110)" for more information on the text group file.

The format of a line in this file is: <index> <string>, where index=1-100, and string is 80 characters or less.To display a string from the text group file, enter a string formatted as: "$CnXl" in the TEXT command as the string to display. Strings which begin with "$C" have special meaning to the TEXT command; they are assumed to be referencing the text group string with index n from the text group file. The number of characters that will be displayed are indicated by the "l" in the string. Strings not beginning with "$C" are interpreted literally.

Syntax

```
TEXT x y string <conditional expression> orientation
font_type
```

where:

x = The x coordinate for the baseline position of the first character of the text display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the text display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of first character of the text display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the text display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

"string" = ASCII string (80 characters or less) enclosed in double or single quotes. Use "$CnXl" format for text group strings, where n=string index in text group file, and l= number of characters of string n to display. Valid range for n=1-100. Valid range for l=1-80.

<conditional_expression> = Optional conditional for string

orientation = Orientation. The choices are:

- HORZ (horizontal)
- VERT (vertical)

font_type = Font types. The choices are:

- BITMAP (followed by bitmap_font_number).
- BITMAP_OVER (followed by bitmap_font_number).
- VECTOR (followed by w, h, line_width).
- VECTOR_OVER (followed by w, h, line_width).

See Section 3-2 for information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number. Valid range = 1 through 8.

w = Width of a vector character cell in pixels. Valid range = 3 through 16,383.

h	=	Height of a vector character cell. Valid range = 3 through 16,383.
line_width	=	Line width. Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

Rules

1. The x and y coordinates can be relative.
2. The origin will be one of the following:
   - Baseline position of the first character of the text string (for bitmap text).
   - Upper left corner of the first character of the text string (for vector text).
3. The user may select the size of the text characters used to display the string.
4. The user may select the orientation of the characters (horizontal or vertical).
5. The user may display the text in either a vector or bitmap font.
6. The user may display various text strings based on a conditional expression. The user **cannot** intermix literal strings and text group strings. If the default string is a text group string (that is, "$CnXl" format), then the conditional string must also be a text group string. If the default string is a literal string, then the conditional string will be interpreted literally also. The type of default string determines the required type of the conditional string.
7. Text strings are limited to 80 chars.
8. When conditionals are used with the TEXT command, the font_type should always be set to **vector_over** or **bitmap_over**. In addition, all of the text strings (including the default) should be padded out to the same length as the longest string.

9. If multi-byte text (Chinese, Japanese, etc.) is specified, the f**ont_type** must be set to **bitmap** or **bitmap_over**. Vector font types are not supported for multi-byte text strings. Be careful to set the font type appropriately if creating a "$Tn" string for a macro file if multi-byte text will be passed to the macro to replace "$Tn".

10. The TEXT command may be used in the background, foreground, or trigger sections of a diagram.

Examples

Example 1

```
TEXT 7912 6000 "Alert Cue" HORZ VECTOR 161 336 3
```

In this example:

x	=	7912
y	=	6000
"string"	=	"Alert Cue"
orientation	=	HORZ
font_type	=	VECTOR
w	=	161
h	=	336
line_width	=	3

This statement defines the text "Alert Cue" to be positioned at 7912, 6000. The string displays horizontally in vector text. The width and height of the text is 161, 336. The line width is 3.

Example 2 (Conditional)

```
TEXT 11063 2990 "GOOD QUALITY_____" (QUALITY) T45J1 "FAIR
QUALITY_____" "POOR QUALITY_____" "BAD QUALITY______" "TIMED
OUT QUALITY"     VERT VECTOR_OVER 990 957 2
```

In this example:

x	=	11063
y	=	2990
"string" (good_cond_value)	=	"GOOD QUALITY_____"
(QUALITY)	=	(QUALITY)
pt_name	=	T454J1
fair_cond_value	=	"FAIR QUALITY_____"
poor_cond_value	=	"POOR QUALITY_____"
bad_cond_value	=	"BAD QUALITY______"
timed_out_cond_value	=	"TIMED OUT QUALITY"
orientation	=	VERT
font_type	=	VECTOR_OVER
w	=	990
h	=	957
line_width	=	2

This statement defines the text "GOOD QUALITY_____" to be positioned at 11063, 2990. The string displays vertically in vector text, with the background of the characters also drawn. Vector_over or bitmap_over **font_type** should always be used with conditional text. The width and height of the text is 990, 957 respectively. The line width is 2. Note that the string is padded out to 17 chars with '_'. This is to illustrate that all of the text strings should be the same length (the length of the longest string). The longest string in this example is "TIMED OUT QUALITY". In a real program, the strings would be padded with blanks. Here they are padded with '_' to emphasize the padding.

The statement indicates that the text will change based on the quality of T454J1 as described below:

If T454J1 has good quality, the text will read "GOOD QUALITY_____".

If T454J1 has fair quality, the text will read "FAIR QUALITY_____".

If T454J1 has poor quality, the text will read "POOR QUALITY_____".

If T454J1 has bad quality, the text will read "BAD QUALITY_____".

If T454J1 has timed out quality, the text will read "TIMED OUT QUALITY".

See Section 2 for more information on conditionals.

Example 3 (text group)

```
TEXT 7912 6000 "$C5X20" HORZ VECTOR 161 336 3
```

The active text group file is below:

1 "1st string in the text group file"
2 "2nd string in the text group file"
3 "3rd string in the text group file"
4 "4th string in the text group file"
5 "5th string in the text group file"
6 "6th string in the text group file"

In this example:

x	=	7912
y	=	6000
"string"	=	"5th string in the text"
orientation	=	HORZ
font_type	=	VECTOR
w	=	161
h	=	336
line_width	=	3

This statement displays the first 20 characters of the fifth string index from the active text group file at 7912, 6000. The string displays horizontally in vector text. The width and height of the text is 161, 336. The line width is 3.

Description

The TIME command displays the current time on the diagram.

Syntax

```
TIME x y format font_type
```

where:

x = The x coordinate for the baseline position of the first character of the time display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The x coordinate of the upper left corner of the time display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

y = The y coordinate for the baseline position of first character of the time display (used if BITMAP or BITMAP_OVER is selected for font_type).

OR

The y coordinate of the upper left corner of the time display (used if VECTOR or VECTOR_OVER is selected for font_type).

Valid range for both font types = 0 through 16,383.

format = Format for time display.
Valid range = 0 through 2. The choices are:

- 0 - hh:mm:ss
- 1 - hh:mm:ss:t
- 2 - hh:mm

where:

- hh = hour (00 through 23)
- mm = minute (00 through 59)
- ss = second (00 through 59)
- t = tenth of a second (0 through 9)

font_type = Font types. The choices are:

- BITMAP (followed by bitmap_font_number).
- BITMAP_OVER (followed by bitmap_font_number).
- VECTOR (followed by w, h, line_width).
- VECTOR_OVER (followed by w, h, line_width).

See Section 3-2 for information on bitmap and vector text.

bitmap_font_number = Standard bitmap font number.
Valid range = 1 through 8.

w = Width of a vector character cell in pixels.
Valid range = 3 through 16,383.

h = Height of a vector character cell in pixels.
Valid range = 3 through 16,383.

line_width = Line width.
Valid range = 1 through 16. The value corresponds to an array index. See Section 3-2 for more information on the line width array index.

Rules

1. The x and y coordinates can be relative.
2. The origin can be one of the following:
   - Baseline position of the first character of the text string (for bitmap text).
   - Upper left corner of the first character of the text string (for vector text).
3. The user may select the size of the text characters used to display the time and may also select the format of the time display.
4. The TIME command may be used in the background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
TIME 1720 600 1 VECTOR 161 336 1
```

In this example:

x	=	1720
y	=	600
format	=	1
font_type	=	VECTOR
w	=	161
h	=	336
line_width	=	1

This time display uses vector text and will be in the format hh:mm:ss:t (for example, 10:35:45:8) as defined by the format number 1. The upper left corner of the time display is positioned at 1720, 600. The width and height of the time display are 161 and 336 respectively. The line width is 1.

Example 2

```
TIME 1720 1200 2 BITMAP 2
```

In this example:

x	=	1720
y	=	1200
format	=	2
font_type	=	BITMAP
bitmap_font_number	=	2

This time display uses bitmap text and will be in the format hh:mm (for example, 10:35) as defined by the format number 2. The upper left corner of the time display is positioned at 1720, 1200. The bitmap font number used is 2.

Description

The TREND command displays a graph that shows the value of a process point sampled over time. Trends are two-dimensional (a trend is an x, y graph which shows the number of values for a point and the time interval between successive values), as opposed to plots that are one-dimensional (values are plotted along a straight line).

Syntax

```
TREND x y w h orientation pt_name rec_fld low high
number_of_values interval scales
```

where:

x = x coordinate of origin of trend.

- If VERT is chosen for orientation, the x coordinate is the upper left corner.
- If HORZ is chosen for orientation, the x coordinate is the lower right corner.

Valid range = 0 through 16,383.

y = y coordinate of origin of trend.

- If VERT is chosen for orientation, the y coordinate is the upper left corner.
- If HORZ is chosen for orientation, the y coordinate is the lower right corner.

Valid range = 0 through 16,383.

w = Length of the time axis.

- Width of trend area if Orientation = HORZ.
- Height of trend area if Orientation = VERT.

Valid range = 1 through 16,383.

h = Length of the non-time axis.

- Height of trend area if Orientation = HORZ.
- Width of the trend area if Orientation = VERT.

Valid range = 1 through 16,383.

orientation = Orientation. The choices are:

- HORZ (horizontal)
- VERT (vertical)

pt_name = Process point name

rec_fld = Optional. Record field (value to be trended). If no record field is specified, the default record field for the given point type is assumed (see Section 2 for more information on default record fields).

low = Low limit used to scale sampled values.
Valid values are:

- Integers in the range -2,147,483,647 through 2,147,483,647.
- Real constants
- Point names/record fields
- Pointers/offsets

high = High limit used to scale sampled values.
Valid values are:

- Integers in the range -2,147,483,647 through 2,147,483,647.
- Real constants
- Point names/record fields
- Pointers/offsets

number_of_values = Number of sampled values to trend.
Valid range = 1 through 60.

interval = Number of seconds between successive sampled values.
Valid range = 1 through 10,000.

scales = Flag to display scale values along x, y axis. The choices are:

- NOSCALE = High and low limit scale values will not display on the trend.
- SCALE = High and low limit scale values will display on the diagram. The x, y will display in vector text 1/10 of width/height of trend respectively.

Rules

1. The x and y coordinates can be relative.
2. The origin will be one of the following:
   - Upper left corner for VERT trend.
   - Lower right corner for HORZ trend.
3. The user defines the number of values to display and the time interval between successive values.
4. The colors defined in the DEF_QUAL command for fair, poor, bad, and timed-out quality are also used to display the trend if quality is NOT good. The color defined in the active COLOR command is used to display the trend for good quality. Note that there are default colors assigned for the different qualities if a DEF_QUAL command is not found in a diagram, and these default colors will be used to display the trend for non-good quality.
5. The trend area is defined by an outlining rectangle, and the spacing between successive values is determined by dividing the length of the x or y axis by the number of values to trend.

6. The user can specify whether or not to display the scale values along the x or the y axis (vertical or horizontal orientation respectively) of the trend. If the user selects the SCALE option, the scale values will be displayed at predefined locations along the x or y axis.

   The user may independently create scales for the trend using PROCESS_PT or TEXT commands, instead of using the predefined scales.

7. The TREND command may be used in the background, foreground, or trigger sections of a diagram.

Example

```
TREND 4810 8372 3818 6483 HORZ A100 AV 10 100 5 15 SCALE
```

In this example:

x	=	4810
y	=	8372
w	=	3818
h	=	6483
orientation	=	HORZ
pt_name	=	A100
rec_fld	=	AV
low	=	10
high	=	100
number_of_values	=	5
interval	=	15
scales	=	SCALE

This statement defines a horizontal trend located at 4810, 8372. The width and height of the outlining rectangle is 3818, 6483. The point and record field to be trended is A100 AV. Five sampled values are scaled between 10 and 100 are displayed. There is a 15-second interval between each successive value. The high and low limit scale values will display on the trend.

Description

The TRIG_ON command is used to activate a trigger region of a diagram (to cause the commands in the trigger region to be executed).

Syntax

```
TRIG_ON n <conditional_expression>
```

where:

n	=	Trigger region number. Valid range = 1 through 255
<conditional_expression>	=	Optional conditional for n

Rules

1. The TRIG_ON command may include a conditional parameter (the user may activate one of several trigger regions based on the evaluation of a conditional expression).
2. The TRIG_ON command may be used in the background, foreground, or trigger sections of a diagram.

Examples

```
TRIGGER 1

COLOR FG red BG white

LINE 3545 3965 9901 3965 4355 9701 5 solid

TRIGGER 2

COLOR FG BLUE BG WHITE

LINE 636 634 7384 6373 1 solid

TRIG_ON 1 (A100 AV < 0.0) 2
```

In this example:

n	=	1
conditional_expression	=	(A100 AV < 0.0) 2

In this example, two trigger regions are defined (trigger 1 and trigger 2). If the value of A100 AV < 0.0, trigger 2 is activated, and the line defined in trigger 2 is displayed. If the value of A100 AV >= 0.0, trigger 1 is activated, and the line in trigger 1 is displayed.

Description

The XY_PLOT command is used to plot two process points (sampled over time) against one another. One process point is scaled along the x axis, the other process point is scaled along the y axis, and the point of their intersection is plotted. Optionally, a shape may be used to plot the intersection point.

Syntax

```
XY_PLOT x y w h x_pt_name x_rec_fld x_low x_high y_pt_name
y_rec_fld y_low y_high scales update_rate <SHAPE_PLOT
current_shape shapew1 shapeh1 rot1 inv1 history_shape shapew2
shapeh2 rot2 inv2 n>
```

where:

x = x coordinate for lower left corner of plot area.
Valid range = 0 through 16,383.

y = y coordinate for lower left corner of plot area.
Valid range = 0 through 16,383.

w = Width of plot area.
Valid range = 1 through 16,383.

h = Height of plot area.
Valid range = 1 through 16,383.

x_pt_name = Point name or pointer name to plot along x axis.

x_rec_fld = Optional. Record field for point on x axis. If no record field is specified, the default record field for the given point type is assumed (see Section 2 for more information on default record fields).

x_low = Low limit value to use to scale point on x axis values.

x_high = High limit value to use to scale point on x axis values.
Valid limit values are:

- Integers in the range -2,147,483,647 through 2,147,483,647.

- Real constants
- Point names/record fields
- Pointers/offsets

y_pt_name = Point name or pointer name to plot along y axis.

y_rec_fld = Optional. Record field for point on y axis. If no record field is specified, the default record field for the given point type is assumed (see Section 2 for more information on default record fields).

y_low = Low limit value to use to scale point on y axis values.

y_high = High limit value to use to scale point on y axis values.
Valid limit values are:

- Integers in the range -2,147,483,647 through 2,147,483,647.
- Real constants
- Point names/record fields
- Pointers/offsets

scales = Flag to display scale values along x, y axis:

- NOSCALE (scale values will not be displayed).
- SCALE (x, y values will display in vector text 1/10 of width/height of xy plot respectively).

update_rate = How often xy_plot will update (in seconds). This update is independent of diagram update rate.
Valid range is 1 through 32767.

current_shape = Shape name from shape library to represent current xy_plot value

history_shape = Shape name from shape library to represent past xy_plot values

shapew1, shapew2 = Virtual width to scale current/history shape to.
Valid range is 1 through 16383.

shapeh1. shapeh2 = Virtual height to scale current/history shape to. Valid range is 1 through 16383.

rot1, rot2 = Degrees to rotate current/history shape (positive = counterclockwise; negative = clockwise). Valid values are: 0,90,-90,180,-180,270,-270.

inv1,inv2 = Inversion flag for current/history shape

- NONE
- TTB (top-to-bottom)
- RTL (right-to-left)
- BOTH (top-to-bottom and right-to-left)

n = Maximum number of past values (history shapes) to display. Valid range is 1 through 255.

Rules

1. The x and y coordinates can be relative.
2. The origin is the lower left corner of the xy_plot.
3. The user specifies the low and high scale values for each axis.
4. The colors defined in the DEF_QUAL command for fair, poor, bad, and timed-out quality are also used to display the xy_plot if quality is NOT good. The color defined in the active COLOR command is used to display the xy_plot for good quality. Note that there are default colors assigned for the different qualities if a DEF_QUAL command is not found in a diagram, and these default colors will be used to display the xy_plot for non-good quality.
5. The *update_rate* parameter is only applicable if the XY_PLOT command is in the foreground diagram section; the xy plot will never update if not in the foreground. The *update_rate* must be greater than or equal to the overall diagram update rate. The xy plot can never update faster than the foreground itself updates. An error will not be reported if the *update_rate* is less than the foreground update rate, but the xy plot will not update at that rate.

6. If a SHAPE_PLOT is specified, the history shape and the current shape may be the same shape. When the number of history shapes (n) is met, the earliest drawn history shape is erased to make room for the most recent history shape. The display of the history shapes functions like a FIFO queue.

7. The user specifies whether or not to display the scale values along the x, y axis of the plot. If the user selects this option, the scale values will be displayed at predefined locations along the x, y axis. If the user does not want the scales, or if the user does not want them at these locations, the scale option should be set to NOSCALE.

   The user may create scales for the XY_PLOT using PROCESS_PT or TEXT commands, instead of using predefined scales.

8. The XY_PLOT command can be used in the background, foreground, or trigger sections of a diagram.

Examples

Example 1

```
XY_PLOT 9686 12558 4263 6743 A2008 AV -10 10 A3003 AV 0 50
SCALE 3
```

In this example:

x	=	9686
y	=	12558
w	=	4263
h	=	6743
x_pt_name	=	A2008
x_rec_fld	=	AV
x_low	=	-10
x_high	=	10
y_pt_name	=	A3003
y_rec_fld	=	AV
y_low	=	0
y_high	=	50
scales	=	SCALE
update_rate	=	3

In this example, an xy plot will be defined at x,y position 9686, 12558. The width of the xy plot is 4263 and the height is 6743. The point plotted along the x axis is A2008 AV. The low limit for A2008 AV is -10 and the high limit is 10. The point plotted along the y axis is A3003 AV. The low limit for A3003 AV is 0 and the high limit is 50. The scale values for the xy plot will be displayed on the diagram. The plot will update every 3 seconds (assuming diagram update rate <= 3 seconds).

Example 2 (shape plot)

```
XY_PLOT 9686 12558 4263 6743 A2008 AV -10 10 A3003 AV 0 50
NOSCALE 3 SHAPE_PLOT ptr 500 300 -90 RTL hollow_ptr 250 150
-90 RTL 20
```

In this example:

x	=	9686
y	=	12558
w	=	4263
h	=	6743
x_pt_name	=	A2008
x_red_fld	=	AV
x_low	=	-10
x_high	=	10
y_pt_name	=	A3003
y_rec_fld	=	AV
y_low	=	0
y_high	=	50
scales	=	NOSCALE
update_rate	=	3
current_shape	=	ptr
shapew1	=	500
shapeh1	=	300
rot1	=	-90
inv1	=	RTL
history_shape	=	hollow_ptr
shapew2	=	250

shapeh2 = 150

rot2 = -90

inv2 = RTL

n = 20

In this example, an xy plot will be defined at x,y position 9686, 12558. The width of the xy plot is 4263 and the height is 6743. The point plotted along the x axis is A2008 AV. The low limit for A2008 AV is -10 and the high limit is 10. The point plotted along the y axis is A3003 AV. The low limit for A3003 AV is 0 and the high limit is 50. The scale values for the xy plot will NOT be displayed on the diagram. The plot will update every 3 seconds (assuming diagram update rate <= 3 seconds). The ptr shape from the shape library will be used to display the current intersection point on the plot. The hollow_ptr shape will be used to display the past values. There will be a maximum of 20 past values on the plot. The current and history shapes are both rotated clockwise by 90 degrees, and are both inverted right-to-left. The current shape is twice as big as the history shapes.

Section 4. Application Programs

4-1. Section Overview

This section describes the application programs that are used in the Operator WEStation process diagrams. These application programs are internal programming routines that perform common control and interactive functions from poke fields, from the RUN_PROGRAMS command, or from the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

The following topics are included:

- Application program introduction (Section 4-2).
- Application program reference pages (Section 4-3).

4-2. Application Program Introduction

Being able to successfully use the application programs requires that the user fully understand Graphics Language commands, point and algorithm record types, and control algorithms. Application programs implement control by changing the contents of record fields in the algorithm. Often, specific record fields need to be defined in the POKE_FLD (type 7) command or the RUN_PROGRAMS command that is using the application programs (see Section 3 for information on these commands).

For these reasons, the user must have a good understanding of the control logic and drop functions being implemented in a system before building custom control diagrams. It is also recommended that the following documents be available for reference.

- "Record Types User's Guide" (U0-0131).
- "Control Algorithms" (U0-0106).

For convenience, the programs that can be used in custom diagrams are listed in Table 4-1 .

Table 4-1. Programs Used in Custom Diagrams

Program Name	Program Number	Program Name	Program Number
CHGDIAG	1	CASC_MODE	102
CNTRL_POKE	6	SEND_CA_EV	105
DIGITON / START_PC	28	WINDOW_FUNC_KEY	117
DIGITOFF / STOP_PC	29	DISP_EFDATA	119
UPSET	30	XPID_DIGITAL	121
DOWNSET	31	EXECUTE_TRIGGER	122
MANUAL	32	CNTRLBITS	124
AUTO	33	WINDOW_DELETE	125
RAISEOUT	34	CHECK_BOX_SELECT	126
LOWEROUT	35	SEND_CA_CTL_LOCK	180
DISP_EFDATA	66	TREND_GROUP	201
SEND_GENMSG	67	EXEC_PROCESS	202
SEND_CA	80	SEND_CA_EV_CTL_LOCK	205
STOP_TRAVEL	93	ACK_DROP_ALARM	210
TRIP_ACK	94	REENABLE_HWY	211
CNTL_TUNE	95	CLEAR_DROP_FAULT	212
CNTRL_PC_POKE	96	REVOTE_BUSLIST	213
DDC_MODE	100	ALARM_ACKNOWLEDGE	214
SUPV_MODE	101	XPID_DIGITAL_CTL_LOCK	221

4-3. Application Program Reference Pages

A brief description of each program is given so that the user can select the appropriate application program. If the program can be used in a custom diagram, the necessary requirements and arguments are given. If the program cannot be used in a custom diagram, it is noted with the phrase “Reserved for Westinghouse Use Only”. The reference pages are arranged in numerical order.

Notes

1. Since some of the application programs require many arguments, it is suggested that when application programs are used in a diagram, the commands should be commented in detail. Although this approach will make the source file larger, comments will not effect the size of the object file.
2. The source (.src) file format should always be loaded unless the .src format is lost or corrupted. Loading the .diag format will cause source code indenting, comments, and macro commands to be lost.

Description sample

CHGDIAG displays a new diagram in the current process diagram window set. The diagram will appear in the main, window, or subwindow depending on the number of the diagram. See Section 3 for information on the diagram number ranges. No arguments are required for this application program. However, the user must specify which diagram is to be displayed.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	Application program number. For this application program, 1 will be entered in the command line.
diag_num	=	Diagram number to be displayed (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 0 will be entered on the command line.
arguments	=	None

Example

```
POKE_FLD 800 15242 642 552 ON 7 1 1 2500 0
```

In this example:

x	=	800
y	=	15242
w	=	642
h	=	552
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	1
diag_num	=	2500
num_of_args	=	0

When the user clicks on this poke field (specified as type 7, program), program 1 (CHGDIAG) executes. CHGDIAG reads the diagram number 2500 from the POKE_FLD statement and displays the diagram. There are no additional arguments for this application program.

Note

If num_of_progs is greater than 1 and CHGDIAG is one of the programs, then it must be the **last** program in the list because when this program runs, it overwrites the existing diagram (that is, it overwrites the current poke command which contains any successive programs to run). Therefore, successive application programs after this one in a poke 7 list will never execute.

Description

CNTRL_POKE is used to define the way control selections are made from an Operator WEStation process diagram. It defines the way the system IDs and algorithm records are interfaced to the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

With CNTRL_POKE, up to 50 control items can be selected from a single diagram, which may be a main screen, window, or subwindow. By clicking on a poke field, the user initiates the actions defined by the specified TRIGGER.

Typical usage defines TRIGGER 1 to include separate calls to other triggers. Each control poke field is assigned a set number from 1 to 50. By including a value of 2 in the setval argument of the program, CNTRL_POKE clears previous trigger actions, and the new TRIGGER commands are executed.

Note

When a trigger number other than 1 is used, the old set value is not cleared. Therefore, if a trigger number other than 1 or set value number other than 2 is used, the user must reset the set value with a SETVAL command.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args point1 point2 trig_num set_num setval
```

where:

x = Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.

y =

w =

h =

state =

•
•
•
•
•

poke_type	=	Poke type (7 is required for application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	6 (program number to execute).
diag_num	=	Diagram to display (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 5 will be entered on the command line.
point1	=	Algorithm point name and ID record field for programmable key functions 1 through 4: RAISE SETPOINT, LOWER SETPOINT, START, STOP.
point2	=	Algorithm point name and ID record field for programmable key functions 5 through 8: AUTO MODE, MANUAL MODE, RAISE OUTPUT, LOWER OUTPUT.
trig_num	=	Trigger number to be executed when selected. Valid range = 1 through 254.
set_num	=	Set variable number (used in the trigger). Valid range = 1 through 255.
setval	=	Integer value to be assigned to the set variable in the trigger. Valid range = 0 through 32,767. If 2 is selected, the previous set will be reset to 1.

Note

It is recommended that trigger 1 and set value 2 be used in the CNTRL_POKE program if the previous trigger items selected are to be deselected.

Example

Example 1

```
POKE_FLD 965 14447 869 1389 ON 7 1 6 0 5 A001X011 ID
A001X021 ID 1 9 2
```

In this example:

x	=	965
y	=	14447
w	=	869
h	=	1389
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	6
diag_num	=	0
num_of_args	=	5
point1	=	A001X011 ID
point2	=	A001X021 ID
trig_num	=	1
set_num	=	9
setval	=	2

When the user clicks on the poke field (specified as type 7, program), program 6 (CNTRL_POKE) will execute. If any diagram changes are described in trigger 1, the last set executed will be cleared and Set9 will be given a value of 2. Then, trigger 1 will be executed. See Section 3 for more information on the TRIGGER keyword. The A001X011 and A001X021 algorithm points are activated for control.

Example 2

```
TRIGGER 4

COLOR FG WHITE BG BLACK BLINK FG OFF BG OFF

TEXT 7912 6000 "Main Screen Active" HORZ VECTOR 161 336 3

POKE_FLD 965 14447 869 1389 ON 7 2 6 0 5 A001X011 ID
A001X021 ID 1 9 2 122 0 2 1 4
```

In this example:

x	=	965
y	=	14447
w	=	869
h	=	1389
state	=	ON
poke_type	=	7
num_of_progs	=	2
prog_num	=	6
diag_num	=	0
num_of_args	=	5
point1	=	A001X011 ID
point2	=	A001X021 ID
trig_num	=	1
set_num	=	9
setval	=	2
prog_num	=	122
diag_num	=	0
num_of_args	=	2

si = 1

trig_num = 4

In this example, a single poke field executes two triggers. Trigger 1 is set by program 6 (CNTRL_POKE) and trigger 4 is set by program 122 (see EXEC_TRIGGER reference page for information on program 122). In this case, trigger 4 displays a message indicating that control is active in the main screen.

TUNEDSPLY (8)

Description

"Reserved for Westinghouse Use Only"

TUNEDSPLY displays the appropriate Tuning Menu for a selected algorithm. It is associated with the Tuning function on the Control Tuning menu. See "Operator WEStation User's Guide" (U0-8100) and the "MAC Application Utilities User's Guide" (U0-0136) for more information on control tuning.

Description

DIGITON / START_PC activates the START/OPEN/TRIP function defined by a particular DPU control algorithm or device point (VC point record). This program executes behind the START/OPEN/TRIP button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following control algorithms: CRTML, PKEY1TO4, PKEY5TO8, and RQAMSP.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by DIGITON.

Use the CNTRL_PC_POKE (96) program to activate the algorithm to be used by START_PC.

DIGITOFF / STOP_PC(29)

Description

DIGITOFF / STOP_PC activates the STOP/CLOSE/RESET function defined by a particular DPU control algorithm or a device point (VC point record). This program executes behind the STOP/CLOSE/RESET button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following algorithms: CRTML, PKEY1TO4, PKEY5TO8, and RQAMSP.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by DIGITOFF.

Use the CNTRL_PC_POKE (96) program to activate the algorithm to be used by STOP_PC.

Description

UPSET raises the set point of a process variable as defined by a control algorithm or a device point (VC point record). This program executes behind the Up arrow (↑) button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following control algorithms: CASMA, CRTML, MLOADER, OIMML, PKEY1TO4, PKEY5TO8, RQAMSP, XMA, XMA2, XML, XML2, XPDSPV, XPID, XPIDOVD, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by UPSET.

DOWNSET (31)

Description

DOWNSET lowers the set point of a process variable as defined by a control algorithm or a device point (VC point record). This program executes behind the Down arrow (↓) button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following control algorithms: CASMA, CRTML, MLOADER, OIMML, PKEY1TO4, PKEY5TO8, RQAMSP, XMA, XMA2, XML, XML2, XPDSPV, XPID, XPIDOVD, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by DOWNSET.

Description

MANUAL places a DPU control algorithm into MANUAL mode. This program executes behind the MANUAL button on the Control Panel window see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following control algorithms: CASMA, CRTMA, PIDMA, PKEY5TO8, QAADMS, QAMDMS, XMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm used by MANUAL.

AUTO (33)

Description

AUTO places a DPU control algorithm into AUTO mode. This program executes behind the AUTO button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following control algorithms: CRTMA, PIDMA, PKEY5TO8, QAADMS, QAMDMS, XMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by AUTO.

Description

RAISEOUT raises a DPU control algorithm's output value. This program executes behind the Output Raise button (Δ) on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following control algorithms: CRTID, CRTMA, PIDMA, PKEY5TO8, QAADMS, QAMDMS, QAMDMS, XMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by RAISEOUT.

LOWEROUT (35)

Description

LOWEROUT lowers a DPU control algorithm's output value. This program executes behind the Output Lower button (∇) on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following control algorithms: CRTID, CRTMA, PIDMA, PKEY5TO8, QAADMS, QAMDMS, XMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by LOWEROUT.

Description

DISP_EFDATA displays data in entry fields in either a main screen or subwindow. This data is read from the Engineer WEStation's memory or specified in the DISP_EPDATA command.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args main sub ef data
```

where:

POKE_FLD	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
x	=	•
y	=	•
w	=	•
h	=	•
state	=	•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	Application program number. For this program, 66 will be entered in the command line.
diag_num	=	Diagram number to be displayed after entry field data is gathered from memory. (0 is required if no other diagram is called). Valid range = 0 through 65535.
num_of_args	=	Number of arguments. For this application program, 4 will be entered on the command line.

main = Number of entry fields to write to in the main screen.

sub = Number of entry fields to write to in the subwindow.

ef = Entry field buffer number. This number should correspond to the entry field buffer number in the ENTRY_FLD command (See Section 3 for more information on this command).
Valid range = 1 through 254.

data = Data to display in the entry field. The data to be written to the field could come from another entry field on the diagram, a specific point name, a variable point name defined by the POINTER command, or a constant number or text string. A point name and record field (such as A100 AV) are considered one argument.

If $P or $H pointer data is being sent to the diagram, these pointers must be defined in the diagram with the POINTER command (See Section 3 for more information on this command). If group ($G) pointers are used, the currently displayed group will be used.

Example

```
POKE_FLD 483 14400 546 904 ON 7 1 66 0 4 1 0 16 "ENTER ALARM
STATUS"
```

In this example:

x	=	483
y	=	14400
w	=	546
h	=	904
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	66
diag_num	=	0
num_of_args	=	4
main	=	1
sub	=	0
ef	=	16
data	=	"ENTER ALARM STATUS"

When the user clicks on the poke field, (specified as type 7, program), program 66 (DISP_EFDATA) executes. No new diagram will be displayed. Four arguments follow. The arguments indicate that data will be entered into one entry field on the main screen and no entry fields in a subwindow. The data will be written to entry field number 16 on the main screen as defined in an ENTRY_FLD command. This data is a fixed text string, "ENTER ALARM STATUS".

SEND_GENMSG (67)

Description

SEND_GENMSG reads data from a list of arguments within the POKE_FLD command, sends a general message over the Data Highway to another drop, and receives block data from that drop. A typical use of this program is to read information from a main screen or subwindow diagram and send data to another drop for processing. The user is responsible for the proper formatting of the block data. For example, there are byte order differences between the WEStation and the DPU. When sending data from the WEStation to the DPU, alternate bytes of data must be swapped for ASCII data and floating point data.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args blk total sia a sib b drop size seg tdiag
drpef si id msgtype ef data dec
```

where:

POKE_FLD	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
x	=	•
y	=	•
w	=	•
h	=	•
state	=	•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	Application program number. For this program, 67 will be entered in the command line.

diag_num = Diagram number to be displayed after general message is sent. If a new diagram is not needed, entering 0 will continue to display the current diagram.
Valid range = 0 through 65,535.

num_of_args = Number of arguments for the application program specified.

blk = When a 0 is entered into this argument, an error message will be displayed on the screen if an expected entry field is left blank; the general message will not be sent.

If a 1 is entered into this argument, a blank entry field will be automatically filled with zeros or blanks (for ASCII data). The zeros or blanks will be sent in the general message.

total = Total number of data items to be sent in the message. The number of items is limited to 255 words of gcode. If this number is incorrect, an error (INV ARG LIST) will be displayed on the screen.

sia = This program can read data from any two of the three types of diagrams. This argument identifies the type of diagram to read from first. The choices are:

1 = Main screen

2 = Subscreen

7 = Window

a = Number of entry fields to read from first diagram (defined by "sia" argument).

sib = The type of diagram to read from next. The choices are:

1 = Main screen

2 = Subscreen

7 = Window

b = Number of entry fields to read from second diagram (defined by "sib" argument).

drop = Number of drops receiving the message. This is typically a 1, but the message can be sent to multiple drops.

size = Maximum size of expected returned data in 1K blocks. This must not exceed the site of seg.

seg = An area of drop memory assigned to store data blocks to and from other drops on the highway.

tdiag = Trigger diagram number. A trigger can be built into the diagram source file to execute when program 67 returns the block data. For example, a message to the user might be displayed. If this is not needed, the value of this argument should be 0. If a trigger is desired, then the tdiag argument must match the number of the diagram to be displayed (diag_num) or the current diagram (main, subscreen, or window).

drpef =

The number of the entry field to contain receiving drop number. If the drop number will not be read from an entry field (general message will always be sent to a drop specified by the "id" parameter), this entry should equal 0.

si = Screen index for entry field to contain receiving drop number. This identifies which diagram contains the drop entry field, If the drop number will not be read from an entry field, enter a 0 for this argument:

0 = Not used

1 = Main screen

2 = Subscreen

7 = Window

id = System ID of receiving drop. This is the number of the drop that is to receive the general message. This number is either read from an entry field that has been completed by the operator, or is specified by the programmer.

If the name of the drop is to be read from an entry field, this argument should equal zero (0).

If the number of the drop is to be read from an entry field, this argument should equal one (1).

If a specific drop number (1 through 254) is specified in this argument, program 67 will always send the general message to that drop.

Note

The drpef, si, and id arguments are used together, as shown in the following table.

Entry field number (drpef argument)	Screen Index (si argument)	System ID (id argument)
0	0	Point name (for example, A100 CN or $P1 $I0) or constant value (for example, 185) representing the actual drop number.
Entry field number (N)	Screen index (S) to location of entry field: 1 = Main 2 = Subscreen 3 = Window	0 = Entry field N on screen S is a point name. 1 = Entry field N on screen S is an integer drop ID.

msgtype = General message type. This must be the first piece of data in the general message. It is used by the receiving drop in order to correctly decipher the message and to initiate the correct action.

Message type information is a special case of the combination of ef and data arguments, described below. It is always defined by two integers (ef = 0, data = message type number).

For some applications, there may be multiple message type numbers included in the general message using ef = 0.

ef = Number of an entry field to read, or 0 for data to be specified in the data argument.

data = If an entry field number is specified for ef, data = type of data in entry field:
0 = ASCII
1 = integer
2 = real
3 = byte
If ef = 0, specify actual data to be sent

dec = Number of decimal places. This argument is required for real numbers.

Notes

1. For every item of data to be included in the general message, there must be "ef" and "data" arguments. The "dec" argument is used only for real numbers.
2. The type of information to be included in a general message depends on the application program receiving the data.

Example

```
POKE_FLD 483 14400 546 904 ON 7 1 67 0 20 0 3 1 1 2 0 1 1 0
3000 0 0 165 0 1 0 30 2 2 2
```

In this example:

x	=	483
y	=	14400
w	=	546
h	=	904
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	67
diag_num	=	0
num_of_args	=	20
blk	=	0
total	=	3
sia	=	1
a	=	1
sib	=	2
b	=	0
drop	=	1
size	=	1
seg	=	0
tdiag	=	3000
drpef	=	0
si	=	0

```
        id  =  165
   msgtype  =  0    1    *message type (bid a prgm)
               0    30   *prgm 30
        ef  =  2
      data  =  2
       dec  =  2   *send real number
```

In this example, a poke field (546 wide and 904 high) is defined at coordinates 483, 14400. A poke type of 7 is required to call a program. One program is called (number 67). A new diagram will not be displayed after the message is sent. Twenty arguments are associated with this command.

The blk argument is 0, so that an error message will be displayed if the required data is not entered before the poke is selected. Three data items will be sent (the message type is included as a data item). One entry field will be read from the main screen (sia = 1, a = 1), and none will be read from the subscreen (sib = 2, b = 0). One drop will receive this block of data.

Note

When data is to be read from an entry field (for example, the operator enters the name or number of the drop to receive the message), the entry field(s) must be built into the diagram (using the ENTRY_FLD command).

A 1K block of memory will be reserved for the response message or data from the target drop. The return data will be written to Segment 0. Main screen number 3000 will be triggered when the data is received at the drop.

The drop number to which the message is sent will not be entered by the operator (drpef = 0, si = 0). The drop number to which the message will always be sent is 165.

The msgtype arguments precede the data. This information is required by the receiving drop to interpret the incoming data correctly. In this case, the message type is a 1 and the program to receive the data is number 30. The data to be sent (a real number with two decimal places) will be read from the second entry field in the main screen.

This example shows the general format for program 67. The num_of_args, total, sia, a, sib, and b arguments would change according to the main and subscreen diagrams being used to send the data.

In general, the entry field number is obtained for each item. If it is non-zero, the data is read from the entry field. If the entry field number is zero, the following data is read. A select pointer type may also be specified, as shown below.

Once all the data is in the message, a receive area is set up for the target drop, the prode number is placed in the message, and the message is sent over the Data Highway. The diagram (if any) from the argument list is displayed. When the data is received from the remote drop, the specified trigger(s) will be executed.

Select pointers ($S)

The select pointer type ($S) may be used in the data argument to indicate that a point name has been entered in an entry field. The pointer name is followed by the record field which is to be included in the message.

When using the select data type ($Sx), the entry field number must be zero. However, this item should be included in the count for the number of entry fields on the applicable screen (the "a" argument for screen "sia" or the "b" argument for screen "sib"). The order in which the entry fields are read is important, as shown in the following example:

sia	a		sib	b		
1	1		2	2	*	read 1 main screen, two subscreen entry fields.
		•				
		•				
		•				
ef #		data		dec		
1		1			*	read an integer from main screen entry field 1.
2		2		2	*	read a real from subscreen entry field 2 (two decimal places
0		$S3		ID	*	read point name from subscreen entry field 3 and send its ID field in the message.

If the $S entry field was specified first, it would be assumed to be a main screen entry field (sia = 1 indicates the main screen is read first).

SEND_CA (80)

Description

SEND_CA is used to send a change attribute message to a point from a diagram. The data to be sent is obtained from entry fields or constants. This program may be used from a main screen, subscreen, or window diagram.

It is recommended that only users who are experienced with graphic diagrams and very familiar with algorithms, record fields, and their functions use this program.

Notes

The SEND_CA program can be locked out by the **WDPF.Tuning.Enabled** parameter in the lockout.dat file.

Another program SEND_CA_CTL_LOCK (180) has the same functions and syntax as SEND_CA but is locked out by the **WDPF.Control.Enabled** parameter in the lockout.dat file.

For more information on the lockout.dat file, see "Operator WEStation Configuration Manual" (U0-8110).

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args blank tot sia a sib b sysef si sysid cmd
bit bitval ef off/type const
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field.
prog_num	=	Application program number. For this program, 80 is entered on the command line.
diag_num	=	Diagram to be displayed after the change attribute message is sent. If a new diagram is not needed, entering "0" will continue to display the current diagram. Valid range = 0 through 65,535.
num_of_args	=	Number of arguments for the specified application program.
blank	=	Error flag. When a 0 is entered into this argument, an error message will be displayed on the screen if an expected entry field is left blank; the change attribute message will not be sent. If a "1" is entered into this argument, a blank entry field will be ignored and not processed by the program.
tot	=	Total number of data items to be sent in the message. The number of items is limited to 255 words of gcode. If this number is incorrect, an error (INV ARG LIST) will be displayed on the screen.

sia = Screen Index A. This program can read data from any two of the three types of diagrams. This argument identifies the type of diagram to read from first. The choices are:

1 = Main screen

2 = Subwindow

7 = Window

a = Number of entry fields to read from first diagram (defined by "sia" argument).

sib = Screen Index B. The type of diagram to read from next. The choices are:

1 = Main screen

2 = Subwindow

7 = Window

b = Number of entry fields to read from second diagram (defined by "sib" argument).

sysef = Entry field buffer number containing the point name or ID of the point to be changed, or zero (0) if sending a constant.

si = Screen index containing the point name ID of the point to be changed. Valid entries are:

0 = Decode the system ID rather than reading it from an entry field.

1 = Entry field in main screen.

2 = Entry field in subscreen.

7 = Entry field in window.

sysid = Description of the type of data found in the entry field. This description can be any of three types depending on the entry field number (sysef) and the screen index (si). These types are constants, variable point names, or variable drop numbers.

Note

The sysef, si, and sysid arguments are used together, as shown in the following table.

Entry field number (sysef argument)	Screen Index (si argument)	System ID (sysid argument)
0	0	Point name (for example, A100 CN or $P1 $I0) or constant value (for example, 185) representing the actual drop number.
Entry field number (N)	Screen index (S) to location of entry field: 1 = Main 2 = Subscreen 7 = Window	0 = Entry field N on screen S is a point name. 1 = Entry field N on screen S is an integer drop ID.

cmd = Type of change attributes being sent, where:

- 0 = change sent in command word — check status for acknowledge (use "bit" and "bitval" arguments).
- 1 = change sent to data record field — check record field for acknowledge.
- 2 = Change sent to data record field — no acknowledge.

bit = Bit number (used only when cmd = 0 only).
Valid range = 0 through 47. This number defines the status record field to be used (and the bit within the field), as follows:

- 0 through 15 = bits 0 through 15 of AS or DS record field.
- 16 through 31 = bits 0 through 15 of A2 record field.
- 32 through 47 = bits 0 through 15 of A3 record field.
- If cmd = 1 or 2, leave this argument blank. DO NOT enter zero.

bitval	=	Value "bit" should be changed to (0 - 1) (when cmd = 0 only). If cmd does not equal zero, leave this argument blank. DO NOT enter zero.
ef	=	Entry field buffer number containing data. When a constant is to be sent, enter 0.
off/typ	=	Field to be changed in data record in the form $On tt, where: • $On = offset pointer • tt = point record field (for example, EV, TB, BB)
const	=	Constant data. When a constant is not sent, leave this argument blank (do not enter a zero instead of a blank).

Although many record fields for a point can be changed by a single command, each record field change is sent as a separate message. This is in contrast to general messages sent to other drops that send all data in the argument list as a single message.

The applicable arguments (cmd, bit, bitval, ef, off/typ, and const) should be defined for every entry field that is being read, unless the field should be left blank as defined in the argument.

Every diagram using program 80 should also include these commands:

POINTER	=	Point to system ID of point to be changed, unless an entry field is used to enter the point name.
ENTRY_FLD	=	Contains data for each change to the point record.

When changing values in record fields, see "Record Types User's Guide" (U0-0131) to determine which record fields can be changed and the proper values to enter into these fields.

In some cases, algorithm points can receive change attribute messages. This occurs during algorithm tuning. To determine which record fields need to be set to accomplish a specific tuning task, see "Control Algorithms" (U0-0106).

Examples

Example 1

```
POKE_FLD 2452 14465 869 898 ON 7 1 80 0 19 1 3 1 2 2 0 0 0
$P90 $I0
1 1 $O1 B0
1 2 $O1 B1
2 0 $O1 CG 0
```

In this example:

x	=	2452
y	=	14465
w	=	869
h	=	898
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	80
diag_num	=	0
num_of_args	=	19
blk	=	1
tot	=	3
sia	=	1
a	=	2
sib	=	2
b	=	0
sysef	=	0
si	=	0
sysid	=	$P90 $I0

SEND_CA (80)
Cont'd

First data item	cmd	=	1
	bit	=	blank (cmd = 1)
	bitval	=	blank (cmd = 1)
	ef	=	1
	off/typ	=	$O1 B0
	const	=	blank (not sending a constant)
Second data item	cmd	=	1
	bit	=	blank (cmd = 1)
	bitval	=	blank (cmd = 1)
	ef	=	2
	off/typ	=	$O1 B1
	const	=	blank (not sending a constant)
Third data item	cmd	=	2
	bit	=	blank (cmd = 2)
	bitval	=	blank (cmd = 2)
	ef	=	0 (sending a constant)
	off/typ	=	$O1 CG
	const	=	0 (sending a constant)

When the user clicks on the poke field (specified as type 7, program) program number 80 (SEND_CA) is executed. A new diagram will not be displayed after the message is sent. Nineteen arguments are associated with this command.

This example indicates that 3 data items will be sent in the message (as specified by tot). Two entry fields will be read from the main screen (sia = 1, a = 2) and zero entry fields will be read from the subwindow (sib = 2, b = 0). In addition, a constant will be sent (ef = 0).

The system entry field buffer number (sysef) and screen index (si) arguments are both 0 so the system ID (sysid) is $P90 $I0.

Each of the three data items is described below:

In the First data item, the cmd argument = 1 which indicates a change sent to the data record field (check record field for acknowledge). The bit number and bit value are left blank because cmd = 1 (see Syntax). The entry field number (ef) is 1. The field to be changed in the data record (off/typ) is $O1 B0. The const argument is left blank because a constant was not sent.

The Second data item is similar to the first data item except that the entry field number (ef) is 2. Also, the field to be changed in the data record (off/typ) is $O1 B1.

In the Third data item, the cmd argument = 2 which indicates a change sent to the data record field (no acknowledge). The bit number and bit value are left blank because cmd = 2 (see Syntax). The entry field number is 0 since a constant is being sent. The field to be changed in the data record (off/typ) is $O1 CG. Zero is the constant (const).

Example 2

```
POKE_FLD 2540 3597 1873 547 ON 7 1
80 0 28 1 4 1 1 2 0 0 0 \AI212S01\ ID
0 11 1 0 $O1 CM 2
1 1 $O1 EV
0 10 1 0 $O1 CM 9
2 0 $O1 CM 1
```

In this example,

x	=	2540
y	=	3597
w	=	1873
h	=	547
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	80 (SEND_CA_EV)
diag_num	=	0
num_of_args	=	28
blk	=	1 - ignore blank entry field
tot	=	4 - total # of items to be sent in message
sia	=	1 - screen index - main screen
a	=	1 - number of entry fields on main screen to read
sib	=	2 - screen index - subwindow
b	=	0 - number of entry fields on subwindow to read
sysef	=	0 - sending a constant
si	=	0 - decode the system id rather than reading it from an entry field
sysid	=	\AI212S01\ ID

First data item (takes point off scan)	cmd	=	0 - change sent in command word - wait for acknowledge
	bit	=	11 - bit 11 from AS record field - point is removed form scan
	bitval	=	1 - wait for bit 11 to equal a "1"
	ef	=	0 - entry field #
	off/typ	=	$O1 CM - offset pointer / CM record field
	const	=	2 - set off scan
Second data item (reads entry field and sets EV record field)	cmd	=	1 - change sent to data record field
	bit	=	Blank (cmd = 1)
	bitval	=	Blank (cmd = 1)
	ef	=	1 - entry field #1
	off/typ	=	$O1 EV - offset pointer - EV record field
	const	=	Blank (not sending a constant)
Third data item (reads EV field and sets AV field)	cmd	=	0 - change sent in command word - wait for acknowledge
	bit	=	10 - bit 1- from AS record field - current value = entered value
	bitval	=	1 - wait for bit 10 to equal a "1"
	ef	=	0 - entry field #
	off/typ	=	$O1 CM - offset pointer / CM record field
	const	=	9 - read entered value
Fourth data item (puts point back on scan)	cmd	=	2 - change sent to data record field - no acknowledge
	bit	=	blank (cmd = 2)
	bitval	=	blank (cmd = 2)
	ef	=	0 - entry field #
	off/typ	=	$O1 CM - offset pointer / CM record field
	const	=	1 - set on scan

When the user clicks on the poke field (specified as type 7), program number 80 (SEND_CA) executes. A new diagram will not be displayed after the message is sent. Twenty-eight arguments are associated with this command.

This example indicates that 4 data items will be sent in the message. The First data item removes the point AI212S01 from scan, the Second data item reads entry field #1 and sets the EV record field to the entered value, the Third data item sets the AV record field equal to the EV record field, and finally the Fourth data item puts the point back on scan.

Description

"Reserved for Westinghouse Use Only"

FORCE_VAL runs behind the Force Open and Force Close buttons from the Force Value Function window (diagram 44). It is used to force ladder contacts and coils open and closed. See "Operator WEStation User's Guide" (U0-8100) for a description of this function.

CLEAR_FORCE (88)

Description

"Reserved for Westinghouse Use Only"

CLEAR_FORCE runs behind the Clear Force button from the Force Value Function window (diagram 44). It is used to return specific forced ladder contacts and coils to their natural (unforced) state. See "Operator WEStation User's Guide" (U0-8100) for a description of this function.

Description

"Reserved for Westinghouse Use Only"

CLEAR_ALL runs behind the Clear All Ladder button from the Force Value Function window (diagram 44). It is used to return all forced contacts and coils in a specific ladder to their natural state. See "Operator WEStation User's Guide" (U0-8100) for a description of this function.

STOP_TRAVEL (93)

Description

STOP_TRAVEL runs behind the STOP TRAVEL button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window). It should only be used with device point (VC) record types.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	93 (program number to execute).
diag_num	=	Diagram to display (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 0 will be entered on the command line.

Example

Example 1

```
POKE_FLD 965 14447 869 1389 ON 7 1 93 0 0
```

In this example:

x	=	965
y	=	14447
w	=	869
h	=	1389
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	93
diag_num	=	0
num_of_args	=	0

TRIP_ACK (94)

Description

TRIP_ACK runs behind the TRIP ACK button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window). It should only be used with device point (VC) record types.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	94 (program number to execute).
diag_num	=	Diagram to display (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 0 will be entered on the command line.

Example

Example 1

```
POKE_FLD 965 14447 869 1389 ON 7 1 94 0 0
```

In this example:

x	=	965
y	=	14447
w	=	869
h	=	1389
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	94
diag_num	=	0
num_of_args	=	0

CNTL_TUNE (95)

Description

CNTL_TUNE allows tuning of M/A station diagrams or other diagrams that include standard Westinghouse tuning diagrams. The user first selects an M/A station by clicking on a defined poke field. When the CNTL_TUNE program is executed, a standard tuning menu will be displayed depending upon the type of algorithm selected.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args tune_id
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field.
prog_num	=	Application program number (for this program, 95 is entered on the command line).
diag_num	=	This argument must always equal 1700 for correct operation. Program 95 calculates the correct diagram to display based on this constant.
num_of_args	=	Number of arguments specified for this application program. For this program, 1 is required.

tune_id = Algorithm point name (sysid) to be tuned. The algorithm name must be in the form, point_name ID, as shown in the following examples: A001X011 ID, $G5 ID, $H90 ID, or $P90 $I0.

With this program, the user can tune an algorithm that has been selected for control. The CNTRL_POKE program automatically defines which algorithm is to be tuned and records that point name in Segment 238.

Example

```
POKE_FLD 965 14465 869 1389 ON 7 1 95 1700 1 A001X011 ID
```

In this example:

x = 965
y = 14465
w = 869
h = 1389
state = ON
poke_type = 7
num_of_progs = 1
prog_num = 95
diag_num = 1700
num_of_args = 1
tune_id = A001X011 ID

In this example, the poke field calls one application program, number 95 (CNTL_TUNE). A001X011 ID will be tuned. Based on the algorithm selected for tuning (A001X011 ID in this example), CNTL_TUNE displays the appropriate tuning screen(s).

CNTRL_PC_POKE (96)

Description

This program is a special version of CNTRL_POKE (program 6). CNTRL_PC_POKE is used only to define control pokes using programmable controllers through either a Programmable Controller Interface (PCI) or Universal Programmable Controller Interface (UPCI) drop.

CNTRL_PC_POKE is used to define more than the eight standard control selections made with the Operator Station Control Keys 1 through 8.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args drop# subdrop# register bit trig# set#
setval
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	7 (required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field.
prog_num	=	Application program number. For this program, 96 is entered on the command line.
diag_num	=	Diagram number to be displayed. If a new diagram is not required, enter a “0”.
num_of_args	=	Number of arguments for the specified application program. This program requires seven arguments.

drop# = Drop number of PCI or UPCI to receive message.Valid range is 1 through 254.

subdrop# = Port number on PCI or UPCI to receive message. Valid range is 0 through 8.

register = Register that contains START/STOP instruction or buffer number required by the specific programmable controller.
IMPORTANT: Refer to the "UPCI User's Guide"(U0-1960) for a list of valid values for this argument. The value depends on the type of programmable controller that is being used.

bit = Bit of the register or other information required by programmable controller.
IMPORTANT: Refer to the "UPCI User's Guide"(U0-1960) for a list of valid values for this argument. The value depends on the type of programmable controller that is being used.

trig# = Trigger number to be executed when selected. Valid range is 1 through 254. If trig# is set to 1, CNTRL_PC_POKE will automatically reset the previous trigger. If any other trigger is used, the previous trigger must be manually reset using the SETVAL Graphics POL command.

set# = Number of the set to be used in the trigger. Valid range is 1 through 255.

setval = Value to be assigned to the set in the trigger. Valid range is 0 through 32767. If a value of "2" is selected, the previous set will be reset to a value of "1".

Rules

1. Trigger 1(trig#=1) and set value 2(setval=2) **must** always be used in the CNTRL_PC_POKE program if the previous trigger items selected are deselected.

2. When using CONTROL_PC commands and the CNTRL_PC_POKE program in the same diagram, do not use SET1 through SET8. Refer to the CONTROL_PC statement.

Example

```
POKE_FLD 1000 2000 3500 3500 ON 7 1 96 0 7 15 0 10 3 1 5 2
```

In this example:

x	=	1000
y	=	2000
w	=	3500
h	=	3500
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	96
diag_num	=	0
num_of_args	=	7
drop#	=	15
subdrop#	=	0
register	=	10
bit	=	3
trig#	=	1
set#	=	5
setval	=	2

Description

DDC_MODE places a DPU control algorithm into DDC mode. This program runs behind the DDC button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following algorithms: PIDMA, XPIDOVD.

Use the CNTRL_POKE (6) program to activate the algorithm used by DDC_MODE. See CNTRLBITS(124) for usage information.

SUPV_MODE (101)

Description

SUPV_MODE places a DPU control algorithm into Supervisory mode. This program runs behind the SUPV button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following algorithms: CASMA, XPDSPV, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm used by SUPV_MODE. See CNTRLBITS(124) for usage information.

Description

CASC_MODE places a DPU control algorithm into Cascade mode. This program runs behind the CASC button on the Control Panel window (see "Operator WEStation User's Guide" (U0-8100) for more information on this window).

This program can be used with the following algorithms: CASMA, XPDSPV, XPID, XPIDOVD, XPIDSLI, XPIDSPV.

Use the CNTRL_POKE (6) program to activate the algorithm used by CASC_MODE. See CNTRLBITS(124) for usage information.

SEND_CA_EV (105)

Description

SEND_CA_EV is used to send a change attribute message for a point from a control diagram. It also creates an operator Event Message that is output to the printer or stored at the Event Logger (see "Historian WEStation User's Guide" (U0-8500) for more information).

Notes

The SEND_CA_EV program can be locked out by the **WDPF.Tuning.Enabled** parameter in the lockout.dat file.

Another program SEND_CA_EV_CTL_LOCK (205) has the same functions and syntax as SEND_CA_EV but is locked out by the **WDPF.Control.Enabled** parameter in the lockout.dat file.

For more information on the lockout.dat file, see " Operator WEStation Configuration Manual" (U0-8110).

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args blank tot sia a sib b sysef si sysid pnid
cmd bit bitval evflag evdesc ef off/typ const
```

where:

x = Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.

y =

w =

h =

state =

poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field.
prog_num	=	Application program number. For this program, 105 is entered on the command line.
diag_num	=	Diagram to be displayed after the change attribute message is sent (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments.
blank	=	Error flag. When a 0 is entered into this argument, an error message will be displayed on the screen if an expected entry field is left blank. Thus, the general message will not be sent. If a 1 is entered into this argument, a blank entry field will be ignored and not processed by the program.
tot	=	Total number of data items to be sent in the message.
sia	=	Screen Index A. This program can read data from any two of the three types of diagrams. This argument identifies the type of diagram to read from first. The choices are: 1 = Main screen 2 = Subwindow 7 = Window
a	=	Number of entry fields to read from first diagram (defined by sia argument).

sib = Screen Index B. The type of diagram to read from next. The choices are:

1 = Main screen

2 = Subwindow

7 = Window

b = Number of entry fields to read from second diagram (defined by sib argument).

sysef = Entry field buffer number containing the point name or ID of the point to be changed, or zero (0) if sending a constant.

si = Screen index containing the point name ID of the point to be changed. Valid entries are:

0 = Not used.

1 = Entry field in main screen.

2 = Entry field in subscreen.

7 = Entry field in window.

sysid = Description of the type of data found in the entry field. This description can be any of three types depending on the entry field number (sysef) and the screen index (si). These types are constants, variable point names, or variable drop numbers.

Note

The sysef, si, and sysid arguments are used together, as shown in the following table.

Entry field number (sysef argument)	Screen Index (si argument)	System ID (sysid argument)
0	0	Point name (for example, A100 CN or $P1 $I0) or constant value (for example, 185) representing the actual drop number.
Entry field number (N)	Screen index (S) to location of entry field: 1 = Main 2 = Subscreen 7 = Window	0 = Entry field N on screen S is a point name. 1 = Entry field N on screen S is an integer drop ID.

pnid = Point name or pointer for location of ASCII point name. Use this format: $Pn $AnXn or "xxxxxxxx"
where "xxxxxxxx" = the ASCII point name.

cmd = Type of change attributes being sent, where:

- 0 = change sent in command word — check status for acknowledge (use "bit" and "bitval" arguments).
- 1 = change sent to data record field — check record field for acknowledge.
- 2 = Change sent to data record field — no acknowledge.

bit = Bit number (used only when cmd = 0 only).
Valid range = 0 through 47. This number defines the status record field to be used (and the bit within the field), as follows:

- 0 through 15 = bits 0 through 15 of AS or DS record field.
- 16 through 31 = bits 0 through 15 of A2 record field.

- 32 through 47 = bits 0 through 15 of A3 record field.
- If cmd = 1 or 2, leave this argument blank. DO NOT enter zero.

bitval = Value "bit" should be changed to (0 - 1) (when cmd = 0 only). If cmd does not equal zero, leave this argument blank. DO NOT enter zero.

evflag = Indicates whether or not an event message is to be sent to the printer at the Event Logger (see "Historian WEStation User's Guide" (U0-8500)).

- Zero (0) indicates that no event message or description should be sent.
- One (1) indicates that an event message and a description (see evdesc) should be sent.

evdesc = Maximum of 8 alphanumeric characters to describe the event. This description is determined by the user. If evflag = 0, leave this argument blank. DO NOT enter a zero. DO NOT use this when cmd = 2.

ef = Entry field buffer number containing data. When a constant is to be sent, enter 0.

off/typ = Field to be changed in data record in the form $On tt, where:

- $On = offset pointer
- tt = point record field (for example, EV, TB, BB)

const = Constant data. When a constant is not sent, leave this argument blank (do not enter a zero instead of a blank).

Although many record fields for a point can be changed by a single command, each record field change is sent as a separate message. This is in contrast to general messages sent to other drops that send all data in the argument list as a single message.

The applicable arguments (cmd, bit, bitval, evflag, evdesc, ef, off/typ, and/or const) should be defined for every entry field that is being read.

Example

```
POKE_FLD 2452 14465 869 898 ON 7 1 105 0 24 1 3 1 2 2 0 0 0
$P90 $S0 $P93 $A0X8
```

1	1	'B0 FLD'	1	$01 B0	
1	1	'B1 FLD'	2	$01 B1	
2			0	$01 CG	0

In this example:

x	=	2452
y	=	14465
w	=	869
h	=	898
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	105
diag_num	=	0
num_of_args	=	24
blk	=	1
tot	=	3
sia	=	1
a	=	2
sib	=	2
b	=	0
sysef	=	0
si	=	0
sysid	=	$P90 $S0
pnid	=	$P93 $A0X8

SEND_CA_EV (105)
Cont'd

First data item	cmd	=	1
	bit	=	blank (cmd = 1)
	bitval	=	blank (cmd = 1)
	evflag	=	1 (send event and description)
	evdesc	=	'B0 FLD'
	ef	=	1
	off/typ	=	$O1 B0
	const	=	blank (not sending a constant)
Second data item	cmd	=	1
	bit	=	blank (cmd = 1)
	bitval	=	blank (cmd = 1)
	evflag	=	1 (send event and description)
	evdesc	=	'B1 FLD'
	ef	=	2
	off/typ	=	$O1 B1
	const	=	blank (not sending a constant)
Third data item	cmd	=	2
	bit	=	blank (cmd = 2)
	bitval	=	blank (cmd = 2)
	evflag	=	blank (cmd = 2)
	evdesc	=	blank (cmd = 2)
	ef	=	0 (sending a constant)
	off/typ	=	$O1 CG
	const	=	0 (sending a constant)

When the user clicks on the poke field (specified as type 7, program) program number 105 (SEND_CA_EV) is executed. A new diagram will not be displayed after the message is sent. Twenty-four arguments are associated with this command.

This example indicates that 3 data items will be sent in the message (as specified by tot). Two entry fields will be read from the main screen (sia = 1, a = 2) and zero entry fields will be read from the subwindow (sib = 2, b = 0). In addition, a constant will be sent (ef = 0).

The system entry field buffer number (sysef) and screen index (si) arguments are both 0 so the system ID (sysid) is $P90 $S0. The point name (pnid) is $P93 $A0X8.

Each of the three data items are described below:

In the First data item, the cmd argument = 1 which indicates a change sent to the data record field (check record field for acknowledge). The bit number and bit value are left blank because cmd = 1 (see Syntax). The event flag (evflag) is set to 1 which indicates that an event and a description should be sent. The event description (evdesc) is 'B0 FLD'. The entry field number (ef) is 1. The field to be changed in the data record (off/typ) is $O1 B0. The const argument is left blank because a constant was not sent.

The Second data item is similar to the first data item except that the event description (evdesc) is 'B1 FLD' and the entry field number (ef) is 2. Also, the field to be changed in the data record (off/typ) is $O1 B1.

In the Third data item, the cmd argument = 2 which indicates a change sent to the data record field (no acknowledge). The bit number and bit value are left blank because cmd = 2 (see Syntax). The event flag (evflag) and the event description (evdesc) are left blank because cmd = 2 (see Syntax). The entry field number is 0 since a constant is being sent. The field to be changed in the data record (off/typ) is $O1 CG. Zero is the constant (const).

WINDOW_FUNC_KEY (117)

Description

WINDOW_FUNC_KEY displays a window when a poke field is selected. To access this program, use the POKE_FLD command as shown below (see Section 3 for more information on this command).

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args group dispx dispy type num_of _points
point_list
```

where:

x y w h state	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information. • • • • •
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field. Otherwise, the number of application programs being executed is entered).
prog_num	=	Application program number. For this program, 117 is entered on the command line.
diag_num	=	Window diagram number to be displayed. Valid range = 7000 through 8999.
num_of_args	=	Number of arguments for this application program.

group	=	Group to display. The choices are: 0 = No group -1 = Same group as in the main screen 1 through 5000 = Window type group
dispx	=	Top left x CRT screen coordinate to begin window. Not currently supported.
dispy	=	Top left y CRT screen coordinate to begin window. Not currently supported.
type	=	0 (no other types supported).
num_of_points	=	The number of $W pointer parameters. That is, the number of points to be passed to the window. If points are not passed, enter a 0 (zero).
point_list	=	List of points to pass into the window. The ID record field must be passed with each point. Passing any other record field will result in a run-time error. If no points are being passed, leave this argument blank. This is applicable if displaying a window that only contains text information, or if point names are coded into the window, or if groups are used.

Example

```
POKE_FLD 2958 13603 1010 1437 ON 7 1 117 7001 7 5000 60 22 0
2 A100 ID A101 ID
```

In this example:

x	=	2958
y	=	13603
w	=	1010
h	=	1437
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	117
diag_num	=	7001
num_of_args	=	7
group	=	5000
dispx	=	60
dispy	=	22
type	=	0
num_of_points	=	2
point_list	=	A100 ID, A101 ID

In this example, poke type 7 (program) calls one application program, number 117 (WINDOW_FUNC_KEY). When the poke field is selected, window number 7001 will display. Seven arguments will be used in this command. Points from group 5000 will be used. The upper left corner of the window will appear at the coordinates 60, 22. Two points, A100 and A101, were passed from the main screen to replace $W pointers in this window diagram.

Description

DISP_EFDATA displays data in entry fields in either a window or subwindow. This data is read from the Engineer WEStation's memory or specified in the DISP_EPDATA command.

This command provides the same functionality and has the same arguments as DISP_EFDATA (66). The only difference is that DISP_EFDATA 119 has an argument called "window" instead of "main" (DISP_EFDATA 66 displays data in entry fields in either a main screen or subwindow, as opposed to a window or subwindow).

For information on the syntax for these two programs, see the DISP_EFDATA (66) reference page.

XPID_DIGITAL (121)

Description

XPID_DIGITAL allows the set point and output fields of an XPID-type algorithm to be changed to user-specified values. This is similar to the standard DES/DEO tuning functions. This function supports the 6- and 7-level Operator Station functionality and enhanced functionality.

The 6- and 7-level Operator Station format uses a single parameter defined as a function number. The function number indicates both the value to be changed and the slew rate (immediate or 20 seconds). The new set value is read from an entry field on the subwindow.

The enhanced format uses 5 parameters. The function number indicates either a set point or an output change. Both the new set value and the slew rate can be entered from entry fields, or the data can be specified in an additional parameter. A screen index is used to define the window from which the entry fields are read. The value and slew rate parameters can be a constant, a point name and record field, or be defined using pointers ($P, $H etc.).

Notes

The XPID_DIGITAL program can be locked out by the **WDPF.Tuning.Enabled** parameter in the lockout.dat file.

Another program XPID_DIGITAL_CTL_LOCK (221) has the same functions and syntax as SPID_DIGITAL, but is locked out by the **WDPF.Control.Enabled** parameter in the lockout.dat file.

For more information on the lockout.dat file, see "Operator WEStation Configuration Manual" (U0-8110).

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args function [ef_value si_value|value
ef_slew si_slew|slew]
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field. Otherwise, the number of application programs being executed is entered).
prog_num	=	Application program number. For this program, 121 is entered on the command line.
diag_num	=	Diagram number to be displayed.
num_of_args	=	Number of arguments for this application program. Valid values are 1 or 5.
function		If num_of_args = 1valid values are:
		1 = change set point with no slew.
		2 = change output with no slew.
		3 = change setpoint with 20 second slew rate.
		4 = change output with 20 second slew rate.
		If num_of_args = 5 valid values are:
		1 = change setpoint
		2 = change output

ef_value	=	Entry field number for value (enter 0 if using parameter list)
si_value	=	Entry field screen index for value (enter if ef_value = 0)
value	=	New set value (enter if ef_value > 0)
ef_slew	=	Entry field number for slew rate (enter 0 if use parameter list)
si_slew	=	Entry field screen index for slew rate (enter if ef_slew > 0)
slew	=	Slew rate in seconds Valid range = 1 through 255 (enter if ef_slew = 0)

Examples

Example 1

```
POKE_FLD 274 2911 989 1081 ON 7 2
6 0 5 A004X046 ID A004X046 ID 1 9 2
121 0 1 1
```

where:

x	=	274
y	=	2911
w	=	989
h	=	1081
state	=	ON
poke_type	=	7
num_of_progs	=	2
prog_num	=	6
diag_num	=	0
num_of_args	=	5
point1	=	`A004X046` ID
point2	=	`A004X046` ID
trig_num	=	1
set_num	=	9
setval	=	2
prog_num	=	121
diag_num	=	0
num_of_args	=	1
function	=	1 = change setpoint with no slew.

In Example 1, the set point of A004X046 changes with no slew. When the poke field is selected, application program number 6, CNTRL_POKE, is executed to select point A004X046 for control. Then XPID_DIGITAL is executed to perform the control action. The control action is specified by the function parameter which indicates change set point with no slew. The new set point is read from entry field number one of the subscreen, and the necessary point attributes are changed by the program to change the set point.

Example 2

```
POKE_FLD 274 2911 989 1081 ON 7 2
6 0 5 A004X046 ID A004X046 ID 1 9 2
121 0 5 1 0 $P3 $S0 0 30
```

where:

x	=	274
y	=	2911
w	=	989
h	=	1081
state	=	ON
poke_type	=	7
num_of_progs	=	2
prog_num	=	6
diag_num	=	0
num_of_args	=	5
point1	=	`A004X046` ID
point2	=	`A004X046` ID
trig_num	=	1
set_num	=	9
setval	=	2

prog_num	=	121
diag_num	=	0
num_of_args	=	5
function		1 = change setpoint
ef_value	=	0
value	=	new set value = $P3 $S0
ef_slew	=	0
slew	=	slew rate = 5 seconds

In Example 2, the set point of A004X046 changes using a slew rate. The program CNTRL_POKE is used to select the point for control as in the first example. The entry field number for the set point is zero which means that the new set point is defined in the next parameter. The set point value is obtained from the memory location defined by the $P3 $S0 parameter. The entry field for the slew rate is also zero so the slew rate is defined by the next parameter which is five seconds.

EXEC_TRIGGER (122)

Description

EXEC_TRIGGER executes a diagram trigger from a poke field.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args si trigger
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	Application program number. For this program, 122 will be entered in the command line.
diag_num	=	Not used. Enter a 0 (zero).
num_of_args	=	Number of arguments. For this application program, 2 will be entered on the command line.
si	=	Identifies the type of diagram containing the trigger. The choices are: 1 = Main screen 2 = Subwindow 7 = Window
trigger	=	Number of the diagram trigger to be executed. Valid trigger numbers are 1 through 255.

Example

Executing a Main screen trigger from a Window

Main Screen	TRIGGER 100 COLOR FG black BG white BLINK FG OFF BG OFF LINE 3848 5382 5291 5382 3848 5980 1 solid
Window	POKE_FLD 962 14352 963 1197 ON 7 1 122 0 2 1 100

In this example, the main screen contains a routine (trigger 100). This routine will be executed by the poke field (specified as type 122) in the window. This technique avoids recoding a trigger that is already available.

CNTRLBITS (124)

Description

CNTRLBITS activates the Operator WEStation P-Key functions to interface with certain DPU control algorithms or device points (VC point record type).

The algorithm(s) to be affected must be selected using CNTRL_POKE (program 6). This program allows the operator to select a control algorithm or device point by pressing the poke field.

When the poke field is pressed, CNTRLBITS determines the correct System ID and P-Key function for the selected algorithm or device point.

The operation associated with each P-Key depends on the type of algorithm (or device point). For example, when using the PKEY1TO4 algorithm, if the poke field associated with the P-Key 1 function is pressed, the digital point associated with the PK1 algorithm parameter will be set. If the poke field associated with the P-Key 1 function is pressed when used with a device point, the STOP TRAVEL function will be implemented. If the P-Key function is not defined in the algorithm, pressing the poke fields that have P-Key functions will not affect the algorithm. See "Control Algorithms" (U0-0106) for more information on PKEY algorithms and "Record Types User's Guide" (U0-0131) for more information on device points.

Note that the first algorithm or point name listed in the CNTRL_POKE program is assigned to P-Key functions 1 through 4 and the second algorithm or point name listed is assigned to P-Key functions 5 through 8.

Notes

- To use all 8 P-Key functions with the same point, enter the same point name into both arguments of the CNTRL_POKE program.
- To use only P1 through P4 (for example, with PKEY1TO4), enter the same point name twice, because the CNTRL_POKE command requires two system IDs.

CNTRLBITS operates on the CN and CP fields of the algorithm/device record. The CN field contains the control word which defines the operation(s) requested by the poke field. The CP field is a copy of the control word (CN field), used to determine when the control word has changed (see "Record Types User's Guide" (U0-0131) for more information on the CN and CP record fields).

At the Operator WEStation, the CP field is read into a temporary variable. Each P-Key function then corresponds to a bit within this variable (the bit is toggled when the poke field is pressed). This temporary variable is written to the CN field of the specified algorithm/device record.

The DPU algorithm/device will read the CN field into a temporary variable, and compare it to the CP field (typically, using an exclusive OR). The temporary variable will be written to the CP field if any bits are found to be different.

Note

The Operator WEStation always reads from the CP field and writes to the CN field. The DPU reads from the CN field, and may both read and write the CP field.

Because this program is implemented using the CN and CP fields of the algorithm record, it can be used only with control algorithms of record types MA, MP, MV, KF, and KV which include these fields. The most commonly used algorithms of these types are listed below:

Text Algorithms	Graphic Algorithms
PKEY1TO4	XMA2
PKEY5TO8	XML2
PKEY1TO8	XPID
SM2XMTRS	XPIDOVD
	XPIDSPV
	XPDSVP
	XPIDSLI
	XPIDV

For a complete list of applicable algorithms, see "Control Algorithms" (U0-0106).

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args Pkey
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field. Otherwise, the number of application programs being executed is entered).
prog_num	=	Application program number. For this program, 124 is entered on the command line.
diag_num	=	Diagram number
num_of_args	=	Number of arguments for the application program specified
Pkey	=	Number that corresponds to the Pkey functions. Valid range = 1 through 8.

Example

```
POKE_FLD 2452 14465 869 1389 ON 7 2 6 0 5 A001Z001 ID
A001Z002 ID 1 9 2 124 0 1 1
```

In this example:

x	=	2452
y	=	14465
w	=	869
h	=	1389
state	=	ON
poke_type	=	7
num_of_progs	=	2
prog_num	=	6 (CNTRL_POKE)
diag_num	=	0
num_of_args	=	5
point1	=	A001Z001 ID
point2	=	A001Z002 ID
trig_num	=	1
set_num	=	9
setval	=	2
prog_num	=	124 (CNTRLBITS)
diag_num	=	0
num_of_args	=	1
Pkey	=	1

In this example, a poke field calls two application programs: program 6 (CNTRL_POKE), which selects the algorithms and program 124 (CNTRLBITS), which assigns the control operation (P-Key functions) to this poke field. When the poke field is selected, both application programs are executed. Neither program uses diagram numbers, therefore, a new diagram will not be displayed.

Five arguments are needed for the CNTRL_POKE program. The first two arguments define which two algorithms will be affected by the P-Key function. The next three arguments work together to execute trigger 1 in the diagram where the cursor is currently displayed (see the description of CNTRL_POKE in this section for more information on this program).

When CNTRL_POKE is complete, CNTRLBITS is executed. Since the poke field is associated with P-Key function 1, the action defined by the PK1 algorithm parameter will be taken.

Description

WINDOW_DELETE removes the control window from the CRT screen.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	7
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	125
diag_num	=	Diagram number to be displayed (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 0 will be entered on the command line.

Example

```
POKE_FLD 12636 2730 2809 2185 ON 7 1 125 0 0
```

In this example:

x	=	12636
y	=	2730
w	=	2809
h	=	2185
state	=	ON
poke_type	=	7
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	125
diag_num	=	Diagram number to be displayed (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 0 will be entered on the command line.

When the user clicks on the poke field (specified as type 7) program 125 (WINDOW_DELETE) executes. The control window (if one is displayed with this frame) will be removed.

Description

CHECK_BOX_SELECT will set (that is, check) the SELECT check box on the process diagram window in which this program is invoked from. It also resets (that is, unchecks) the currently selected process diagram window (if applicable).

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	7
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	126
diag_num	=	Diagram number to be displayed (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 0 will be entered on the command line.

Example

```
POKE_FLD 1872 3822 1405 1639 ON 7 1 126 0 0
```

In this example:

x	=	1872
y	=	3822
w	=	1405
h	=	1639
state	=	ON
poke_type	=	7
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	126
diag_num	=	Diagram number to be displayed (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 0 will be entered on the command line.

When the user clicks on the poke field (specified as type 7), program 126 (CHECK_BOX_SELECT) executes. The SELECT check box will be set for the main frame in which this poke resides. If another process diagram window is currently selected (that is, checked), it will be deselected (that is, unchecked).

Description

The SEND_CA_CTL_LOCK program is identical to the SEND_CA (80) program except for the lock-out procedure.

It is recommended that only users who are experienced with graphic diagrams and very familiar with algorithms, record fields, and their functions use this program.

Notes

The SEND_CA_CTL_LOCK program can be locked out by the **WDPF.Control.Enabled** parameter in the lockout.dat file.

SEND_CA can be locked out by the **WDPF.Tuning.Enabled** parameter in the lockout.dat file.

For more information on the lockout.dat file, see "Operator WEStation Configuration Manual" (U0-8110).

Syntax

The syntax for the SEND_CA_CTL_LOCK program is exactly the same as for the SEND_CA (80) program. See the SEND_CA (80) section for the syntax.

TREND_GROUP (201)

Description

TREND_GROUP sends a request to the Trend System to display a live or historical trend group.

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num
diag_num num_of_args type trend_num
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (7 is required for this application program).
num_of_progs	=	Number of programs to be executed (usually 1, unless additional application programs are linked to this poke field).
prog_num	=	Application program number. For this application program, 201 will be entered on the command line.
diag_num	=	Diagram number to be displayed (0 is required if no other diagram is called). Valid range = 0 through 65,535.
num_of_args	=	Number of arguments. For this application program, 2 will be entered on the command line.
type	=	Type of trend group. The choices are: 1 = Live trend group 2 = Historical trend group
trend_num	=	Trend group number. Valid range = 1 through 600.

Example

```
POKE_FLD 965 1447 869 1389 ON 7 1 201 0 2 1 350
```

In this example:

x	=	965
y	=	1447
w	=	869
h	=	1389
state	=	ON
poke_type	=	7
num_of_progs	=	1
prog_num	=	201
diag_num	=	0
num_of_args	=	2
type	=	1
trend_num	=	350

When the user clicks on the poke field (specified as type 7, program) program 201 (TREND_GROUP) will execute. A live trend window will be displayed with trend group 350. See "Operator WEStation User's Guide" (U0-8100) for more information on trends.

EXECUTE_PROCESS(202)

Description

EXECUTE_PROCESS mirrors a UNIX command line with parameters. It is run from a poke type 7 command. It is also used to implement the poke type 9 command. The user may want to run this program from a poke type 7 if additional application programs need to be run when this program is run.

Syntax

```
POKE_FLD x y w h state poke_type progs prog_num diag_num args
process
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	7
progs	=	Number of programs to execute (usually 1, unless additional application programs are linked to this poke field)
prog_num	=	202
diag_num	=	Diagram number to be displayed (0 is required if no other diagram is to be displayed). Valid range is 0 through 65635.
args	=	1 (number of arguments)
process	=	The complete path name plus parameters of the UNIX process to be executed (string must be in quotes).

Example

```
POKE_FLD 2013 3560 1422 1373 ON 7 1 202 0 1
"/usr/openwin/bin/shelltool -I pwd"
```

where:

x	=	2013
y	=	3560
w	=	1422
h	=	1373
state	=	ON
poke_type	=	7
progs	=	1
prog_num	=	202
diag_num	=	0 - no diagram to be displayed
args	=	1
process	=	"/usr/openwin/bin/shelltool -I pwd"

When the user clicks on the poke field (specified as type 7), program 202(EXECUTE_PROCESS) runs. The program "/usr/openwin/bin/shelltool" will be executed and the parameters "-I pwd" will be passed to the shell. This UNIX command brings up a new shelltool window and prints the current working directory (pwd) after the command line prompt.

SEND_CA_EV_CTL_LOCK (205)

Description

The SEND_CA_EV_CTL_LOCK program is identical to the SEND_CA_EV (105) program except for the lock-out procedure.

Notes

The SEND_CA_EV_CTL_LOCK program can be locked out by the **WDPF.Control.Enabled** parameter in the lockout.dat file.

SEND_CA_EV can be locked out by the **WDPF.Tuning.Enabled** parameter in the lockout.dat file.

For more information on the lockout.dat file, see " Operator WEStation Configuration Manual" (U0-8110).

Syntax

The syntax for the SEND_CA_EV_CTL_LOCK program is exactly the same as for the SEND_CA_EV (105) program. See the SEND_CA_EV (105) section for the syntax.

Description

ACK_DROP_ALARM acknowledges a Drop Alarm for the selected drop, by issuing the appropriate command through the Drop Status Record for that drop. This program is intended for use on the System Status Display diagrams (see "Self-Test Diagnostics" (M0-0003) for more information on the System Status Display). The program is invoked by first selecting a Drop Status Record (point record of type DV) and then selecting the poke to which program 210 is attached (see "Record Types User's Guide" (U0-0131) for more information on Drop Status Records).

Syntax

```
POKE_FLD x y w h state poke_type prog_num
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (3 is required for this application program).
prog_num	=	Application program number. For this application program, 210 will be entered on the command line.

Example

```
POKE_FLD 8288 14697 445 415 ON 3 210
```

In this example:

x	=	8288
y	=	14697
w	=	445
h	=	415
state	=	ON
poke_type	=	3
prog_num	=	210

When the user clicks on the poke field (specified as type 3), program 210 (ACK_DROP_ALARM) will execute on the Drop Status record that the user selected.

Description

REENABLE_HWY attempts to re-enable a failed highway channel for the selected drop by issuing the appropriate drop command through the Drop Status Record for that drop (see "Record Types User's Guide" (U0-0131) for more information on the Drop Status records). This program is intended for use on the System Status Display (see "Self-Test Diagnostics" (M0-0003) for more information on the System Status Display).

Syntax

```
POKE_FLD x y w h state poke_type prog_num
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (3 is required for this application program).
prog_num	=	Application program number. For this application program, 211 will be entered on the command line.

Example

```
POKE_FLD 5180 15318 445 415 ON 3 211
```

In this example:

x	=	5180
y	=	15318
w	=	445
h	=	415
state	=	ON
poke_type	=	3
prog_num	=	211

When the user clicks on the poke field (specified as type 3), program 211 (REENABLE_HWY) will execute on the Drop Status Record that user selected. See "Record Types User's Guide" (U0-0131) for more information on Drop Status Records.

Description

CLEAR_DROP_FAULT attempts to clear the drop alarm and fault conditions for the selected drop, by issuing the appropriate drop command through the Drop Status record of that drop (see "Record Types User's Guide" (U0-0131) for more information on Drop Status records). This program is intended for use on the System Status Display (see "Self-Test Diagnostics" (M0-0003) for more information on the System Status Display).

Syntax

```
POKE_FLD x y w h state poke_type prog_num
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (3 is required for this application program).
prog_num	=	Application program number. For this application program, 212 will be entered on the command line.

Example

```
POKE_FLD 8288 15318 445 415 ON 3 212
```

In this example:

x	=	8288
y	=	15318
w	=	445
h	=	415
state	=	ON
poke_type	=	3
prog_num	=	212

When the user clicks on the poke field (specified as type 3), program 212 (CLEAR_DROP_FAULT) will execute on the Drop Status record that the operator selected.

Description

REVOTE_BUSLIST causes the Bus Allocation List to be revoted at the selected highway (which is implied by the selected Drop Status Record). For more information on the Revote Bus List function, see "WDPF System Planning and Highway Installation Manual" (M0-8000).

Syntax

```
POKE_FLD x y w h state poke_type prog_num
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	Poke type (3 is required for this application program).
prog_num	=	Application program number. For this application program, 213 will be entered on the command line.

Example

```
POKE_FLD 8288 15318 445 415 ON 3 213
```

In this example:

x	=	8288
y	=	15318
w	=	445
h	=	415
state	=	ON
poke_type	=	3
prog_num	=	213

When the user clicks on the poke field (specified as type 3), program 213 (REVOTE_BUSLIST) will execute on the Drop Status record that the operator selected.

Description

The ALARM_ACKNOWLEDGE function acknowledges a group of alarms that are defined with this poke field. Up to a maximum of 100 points are allowed in one list, but only 40 will be acknowledged with each poke selection.

Syntax

```
POKE_FLD x y w h state poke_type progs prog_num diag_num
num_pts points
```

where:

x	=	Standard POKE_FLD parameters. See POKE_FLD reference pages for more information.
y	=	•
w	=	•
h	=	•
state	=	•
		•
poke_type	=	7
progs	=	Number of application programs to run (usually 1 unless additional programs are to be invoked by this poke)
prog_num	=	Application program number. For this application program, 214 will be entered on the command line.
diag_num	=	Number of diagram to be displayed (0 if no new diagram is required).Valid range is 0 through 65535.
num_pts	=	Number of points to acknowledge alarms for.Valid range is 1 through 100.
points	=	list of num_pts point names (with mandatory ID record fields).

Example

```
POKE_FLD 8892 4368 2341 2185 ON 7 1 214 0 3 \AI212S01\ ID
\AI212S02\ ID \AI212S03\ ID
```

In this example:

x	=	8892
y	=	4368
w	=	2341
h	=	2185
state	=	ON
poke_type	=	7
progs	=	1
prog_num	=	214
diag_num	=	0 - no new diagram will be displayed
num_pts	=	3
points	=	\AI212S01\ ID, \AI212S02\ ID, \AI212S03\ ID

When the user clicks on the poke field (specified as type 7), program 214 (ALARM_ACKNOWLEDGE) will execute. The points AI212S01, AI212S02, and AI212S03 will be acknowledged if they are in alarm.

Note

If the n*um_pts* parameter is greater than 40, then the first 40 points in alarm will be acknowledged when the poke is selected. The user will then have to select the poke again to acknowledge the next set of 40, and so on. Since there can be a maximum of 100 points defined, the user would have to select the poke a maximum of three times to acknowledge all alarms, assuming all 100 points were in alarm.

Description

The XPID_DIGITAL_CTL_LOCK program is identical to the XPID_DIGITAL (121) program except for the lock-out procedure.

Notes

The XPID_DIGITAL_CTL_LOCK program can be locked out by the **WDPF.Control.Enabled** parameter in the lockout.dat file.

XPID_DIGITAL can be locked out by the **WDPF.Tuning.Enabled** parameter in the lockout.dat file.

For more information on the lockout.dat file, see " Operator WEStation Configuration Manual" (U0-8110).

Syntax

The syntax for the XPID_DIGITAL_CTL_LOCK program is exactly the same as for the XPID_DIGITAL (121) program. See the XPID_DIGITAL (121) section for the syntax.

START_PROGRAM (254)

Description

"Reserved for Westinghouse Use Only"

START_PROGRAM establishes a shared memory segment for interprocess communication.

Description

"Reserved for Westinghouse Use Only"

RETURN_KEY returns an integer value (representing a function key number) to a "child" process.

Appendix A. Reserved Words

A-1. Overview

This appendix lists the words that are reserved for use by the compiler. In addition to the words listed on the following pages, the items below are also considered reserved:

- Point names.
- Pointer names ($P, $H, $D, $G, and $W) and offsets ($In, $Rn, $Bn, $Sn, $AnXl).
- Record type and record field names. For additional information, see "Record Types User's Guide" (U0-0131).
- Fill pattern names.
- Line pattern names.
- Color names.
- Shape names.

Note

These terms are case insensitive (that is, not sensitive to upper and lower case letters). The exception is for shape names. They must be used as defined.

A-2. List of Reserved Words

The following words are reserved for use by the compiler:

AB	CIRCLE	EXEC_TRIG
AC	COLOR	EXIT
ADD	CURSOR	EXPONENTIAL
AI	CUTOUT	FAIR
AL	DATE	FALSE
ALARM	DC	FG
ALARMACK	DI	FITTED
ALARMOFF	DIAG_DISP	FOREGROUND
AM	DIAGRAM	FORMAT
ARC	DISABLE	FUNC_KEY
ASCII	DL	GCODE
AX	DM	GOOD
BACKGROUND	DOT	GP
BAD	DOWN	GTEXT
BAR	DROPALM	GX
BETTER	DROPCLEAR	HDWRFAIL
BG	DROPFAULT	HEX
BIAS	DX	HEX_H
B INARY	ELLIPSE	HIGHALARM
BITMAP	ELSE	HORZ
BITMAP_OVER	ENDIF	IF
BLINK	ENDLOOP	IFELSE
BN	ENTERVALUE	INT
BOTH	ENTRY_FLD	INVOKED
BX	EXCL	KEYBOARD
BYTE	EXEC_POKE	LARGE

LEFT
LIMITOFF
LINE
LOOP
LOWALARM
MACRO
MAIN
MATH
MCB0OFFLIN
MCB1OFFLIN
MEDIUM
MULT
NONE
NON_EXCL
NORMAL
NOSCALE
NOT_FITTED
OFF
OFF0
OFF1
OFF2
OFF3
OFF4
OFF5
OFF6
0FF7
OFF8
OFF9
OFF10
OFF11
OFF12
OFF13
OFF14
OFF15
OL
OLNORMAL
OL_BUTTON
OL_CHECKBOX
OL_CHOICE
OL_CYLINDER
OL_EVENT_MENU
OL_GAUGE
OL_RECTANGLE
OL_SLIDER
ON
ON0
ON1
ON2
ON3
ON4
ON5
ON6
ON7
ON8
ON9
ON10
ON11
ON12
ON13
ON14
ON15
OPATTN
PB
P_ENDLOOP
PAGE
PLOT
POINTER
POKE_FLD
POKE_STATE
POLYGON
POOR
PRESET0
PRESET1
PRESET2
PRESET3
PRESET4
PRESET5
PRESET6
PRESET7
PRESET8
PRESET9
PRESET10
PRESET11
PRESET12
PRESET13
PRESET14
PRESET15
PRESET16
PRESET17
PRESET18

PRESET19	PSET16	RUN_PROGRAMS
PRESET20	PSET17	SCALE
PRESET21	PSET18	SCANOFF
PRESET22	PSET19	SENSORALM
PRESET23	PSET20	SENSORMODE
PRESET24	PSET21	SET
PRESET25	PSET22	SETALM
PRESET26	PSET23	SETVAL
PRESET27	PSET24	SHAPE
PRESET28	PSET25	SHAPE_LABEL
PRESET29	PSET26	SHAPE_PLOT
PRESET30	PSET27	SMALL
PRESET31	PSET28	SQUARED
PROCESS_PT	PSET29	SUB
PSET0	PSET30	SUBWIN
PSET1	PSET31	SW_ALARM_COUNT
PSET2	PTR_EQUAL	SW_ALARM_CUE
PSET3	PTR_LOOP	SW_ALERT_LIST
PSET4	PTR_MOVE	SW_PROCESS_PT
PSET5	PTR_VALUE	TEXT
PSET6	PX	TEXT_LABEL
PSET7	REAL	TIME
PSET8	RECTANGLE	TIMED_OUT
PSET9	RESET	TITLE
PSET10	RESETALM	TOGGLE
PSET11	RIGHT	TREND
PSET12	RIGHT0	TRIGGER
PSET13	ROTATION	TRIG_ON
PSET14	ROUNDED	TRUE
PSET15	RTL	TTB

UP	VECTOR	VX
UPDATETIME	VECTOR_OVER	WIN
VC	VERT	WORSE
		XY_PLOT

Appendix B. Status Words

B-1. Overview

This appendix lists the Graphics Language status words. The general purpose words (ONx, OFFx, etc.) enable the user to check a bit within any integer record field of a point record. Other status words are designed to check for specific conditions, based on the value of specific point record fields. (See "Record Types User's Guide" (U0-0131) for the structures of the various point records.) Multiple bits may be checked by using a compound conditional (that is, using a logical operator (AND or OR) to combine the simple expressions that check each bit.

- Mask = Bits that are checked.
- Pattern = Value of the bits that are checked.

For example, the status word NORMAL has mask = 1000 0000 1000 0000, indicating that bits 7 and 15 are checked.

The pattern for NORMAL is 0000 0000 0000 0000, indicating that the bits checked should equal zero. For example, the conditional {A100 = NORMAL} will evaluate as true if bits 7 and 15 in the AS field of analog point A100 are reset (equal to zero).

B-2. List of Status Words

Table B-1 provides the following information on the Graphics Language status words: description, mask, pattern, and record type (default record fields).

Table B-1. Graphics Language Status Words

Status Words	Description	Mask 15 0	Pattern 15 0	Record Type (Record Field)
ALARM	Point in Alarm	1000 0000 1000 0000	0000 0000 1000 0000	Analog (AS) Digital (DS) Device (DS) Drop Status (FA)
ALARMACK	Unacknowledged Alarm	1000 0000 0010 0000	0000 0000 0010 0000	Analog (AS Digital (DS) Device (DS) Drop Status (FA)
ALARMOFF	Alarm Check Off	1010 0000 0000 0000	0010 0000 0000 0000	Analog (AS) Digital (DS) Device (DS)
BAD	Quality = Bad	1000 0011 0000 0000	0000 0011 0000 0000	Analog (AS) Digital (DS) Device (DS)
BETTER	Better	1000 0000 0000 0011	0000 0000 0000 0001	Analog (AS)
CUTOUT	Cutout	1000 0000 0100 0000	0000 0000 0100 0000	Analog (AS) Digital (DS) Device (DS)
DROPALM	Drop in Alarm	0000 0000 1000 0000	0000 0000 1000 0000	Drop Status (FA)
DROPCLEAR	Drop not in Alarm	0000 0000 1000 0000	0000 0000 0000 0000	Drop Status (FA)
DROPFAULT	Drop Fault	0000 0000 0100 0000	0000 0000 0100 0000	Drop Status (FA)
ENTERVALUE	Value was Entered	1000 0100 0000 0000	0000 0100 0000 0000	Analog (AS) Digital (DS)
FAIR	Quality = Fair	1000 0011 0000 0000	0000 0001 0000 0000	Analog (AS) Digital (DS) Device (DS)
GOOD	Quality = Good	1000 0011 0000 0000	0000 0000 0000 0000	Analog (AS) Digital (DS) Device (DS)

Table B-1. Graphics Language Status Words (Cont'd)

Status Words	Description	Mask 15 0	Pattern 15 0	Record Type (Record Field)
HDWRFAIL	Hardware Fail (timed-out)	1000 0000 0000 0000	1000 0000 0000 0000	Analog (AS) Digital (DS) Device (DS) Drop Status (FA) Packed Digital (AS) Packed Group (DS)
HIGHALARM	High Alarm	1000 0000 1000 1100	0000 0000 1000 1000	Analog (AS)
LIMITOFF	Limit Check Off	1001 0000 0000 0000	0001 0000 0000 0000	Analog (AS)
LOWALARM	Low Alarm	1000 0000 1000 1100	0000 0000 1000 0100	Analog (AS)
MCB0OFFLIN	MBC 0 Inactive	0000 0001 0000 0000	0000 0001 0000 0000	Drop Status (FA)
MCB1OFFLIN	MBC 1 Inactive	0000 0010 0000 0000	0000 0010 0000 0000	Drop Status (FA)
NORMAL	Point not in Alarm	1000 0000 1000 0000	0000 0000 0000 0000	Analog (AS) Digital (DS) Drop Status (FA) Device (DS)
OFF0	Specified bit is OFF	0000 0000 0000 0001	0000 0000 0000 0000	OFF0 through OFF15 are applicable to any integer record field of any point type.
OFF1		0000 0000 0000 0010	0000 0000 0000 0000	
OFF2		0000 0000 0000 0100	0000 0000 0000 0000	
OFF3		0000 0000 0000 1000	0000 0000 0000 0000	
OFF4		0000 0000 0001 0000	0000 0000 0000 0000	
OFF5		0000 0000 0010 0000	0000 0000 0000 0000	
OFF6		0000 0000 0100 0000	0000 0000 0000 0000	
OFF7		0000 0000 1000 0000	0000 0000 000 0000	
OFF8		0000 0001 0000 0000	0000 0000 0000 0000	
OFF9		0000 0010 0000 0000	0000 0000 0000 0000	
OFF10		0000 0100 0000 0000	0000 0000 0000 0000	
OFF11		0000 1000 0000 0000	0000 0000 0000 0000	
OFF12		0001 0000 0000 0000	0000 0000 0000 0000	
OFF13		0010 0000 0000 0000	0000 0000 0000 0000	
OFF14		0100 0000 0000 0000	0000 0000 0000 0000	

Table B-1. Graphics Language Status Words (Cont'd)

Status Words	Description	Mask 15 0	Pattern 15 0	Record Type (Record Field)
OFF15	Specified bit is OFF	1000 0000 0000 0000	0000 0000 0000 0000	OFF0 through OFF15 are applicable to any integer record field of any point type.
ON0	Specified bit is ON	0000 0000 0000 0001	0000 0000 0000 0001	ON0 through ON15 are applicable to any integer record field of any point type.
ON1		0000 0000 0000 0010	0000 0000 0000 0010	
ON2		0000 0000 0000 0100	0000 0000 0000 0100	
ON3		0000 0000 0000 1000	0000 0000 0000 1000	
ON4		0000 0000 0001 0000	0000 0000 0001 0000	
ON5		0000 0000 0010 0000	0000 0000 0010 0000	
ON6		0000 0000 0100 0000	0000 0000 0100 0000	
ON7		0000 0000 1000 0000	0000 0000 1000 0000	
ON8		0000 0001 0000 0000	0000 0001 0000 0000	
ON9		0000 0010 0000 0000	0000 0010 0000 0000	
ON10		0000 0100 0000 0000	0000 0100 0000 0000	
ON11		0000 1000 0000 0000	0000 1000 0000 0000	
ON12		0001 0000 0000 0000	0001 0000 0000 0000	
ON13		0010 0000 0000 0000	0010 0000 0000 0000	
ON14		0100 0000 0000 0000	0100 0000 0000 0000	
ON15		1000 0000 0000 0000	1000 0000 0000 0000	
OPATTN	Operator Attention Required	0000 0000 0001 0000	0000 0000 0001 0000	Drop Status (FA)
POOR	Quality = Poor	1000 0011 0000 0000	0000 0010 0000 0000	Analog (AS) Digital (DS) Device (DS)
PRESET0	Specified bit is reset (= 0)	0000 0000 0000 0001	0000 0000 0000 0000	Packed Digital (AV)
PRESET1		0000 0000 0000 0010	0000 0000 0000 0000	
PRESET2		0000 0000 0000 0100	0000 0000 0000 0000	
PRESET3		0000 0000 0000 1000	0000 0000 0000 0000	

Table B-1. Graphics Language Status Words (Cont'd)

Status Words	Description	Mask 15 0	Pattern 15 0	Record Type (Record Field)
PRESET4	Specified bit is reset (= 0)	0000 0000 0001 0000	0000 0000 0000 0000	Packed Digital (AV)
PRESET5		0000 0000 0010 0000	0000 0000 0000 0000	
PRESET6		0000 0000 0100 0000	0000 0000 0000 0000	
PRESET7		0000 0000 1000 0000	0000 0000 0000 0000	
PRESET8		0000 0001 0000 0000	0000 0000 0000 0000	
PRESET9		0000 0010 0000 0000	0000 0000 0000 0000	
PRESET10		0000 0100 0000 0000	0000 0000 0000 0000	
PRESET11		0000 1000 0000 0000	0000 0000 0000 0000	
PRESET 12		0001 0000 0000 0000	0000 0000 0000 0000	
PRESET13		0010 0000 0000 0000	0000 0000 0000 0000	
PRESET14		0100 0000 0000 0000	0000 0000 0000 0000	
PRESET15		1000 0000 0000 0000	0000 0000 0000 0000	
PRESET16		0000 0000 0000 0001	0000 0000 0000 0000	
PRESET17		0000 0000 0000 0010	0000 0000 0000 0000	
PRESET18		0000 0000 0000 0100	0000 0000 0000 0000	
PRESET19		0000 0000 0000 1000	0000 0000 0000 0000	
PRESET20		0000 0000 0001 0000	0000 0000 0000 0000	
PRESET21		0000 0000 0010 0000	0000 0000 0000 0000	
PRESET22		0000 0000 0100 0000	0000 0000 0000 0000	
PRESET23		0000 0000 1000 0000	0000 0000 0000 0000	
PRESET24		0000 0001 0000 0000	0000 0000 0000 0000	
PRESET25		0000 0010 0000 0000	0000 0000 0000 0000	
PRESET26		0000 0100 0000 0000	0000 0000 0000 0000	
PRESET27		0000 1000 0000 0000	0000 0000 0000 0000	
PRESET28		0001 0000 0000 0000	0000 0000 0000 0000	
PRESET29		0010 0000 0000 0000	0000 0000 0000 0000	
PRESET30		0100 0000 0000 0000	0000 0000 0000 0000	
PRESET31		1000 0000 0000 0000	0000 0000 0000 0000	

Table B-1. Graphics Language Status Words (Cont'd)

Status Words	Description	Mask 15 0	Pattern 15 0	Record Type (Record Field)
PSET0	Specified bit is set (= 1)	0000 0000 0000 0001	0000 0000 0000 0001	Packed Digital (AV)
PSET1		0000 0000 0000 0010	0000 0000 0000 0010	
PSET2		0000 0000 0000 0100	0000 0000 0000 0100	
PSET3		0000 0000 0000 1000	0000 0000 0000 1000	
PSET4		0000 0000 0001 0000	0000 0000 0001 0000	
PSET5		0000 0000 0010 0000	0000 0000 0010 0000	
PSET6		0000 0000 0100 0000	0000 0000 0100 0000	
PSET7		0000 0000 1000 0000	0000 0000 1000 0000	
PSET8		0000 0001 0000 0000	0000 0001 0000 0000	
PSET9		0000 0010 0000 0000	0000 0010 0000 0000	
PSET10		0000 0100 0000 0000	0000 0100 0000 0000	
PSET11		0000 1000 0000 0000	0000 1000 0000 0000	
PSET12		0001 0000 0000 0000	0001 0000 0000 0000	
PSET13		0010 0000 0000 0000	0010 0000 0000 0000	
PSET14		0100 0000 0000 0000	0100 0000 0000 0000	
PSET15		1000 0000 0000 0000	1000 0000 0000 0000	
PSET16		0000 0000 0000 0001	0000 0000 0000 0001	
PSET17		0000 0000 0000 0010	0000 0000 0000 0010	
PSET18		0000 0000 0000 0100	0000 0000 0000 0100	
PSET19		0000 0000 0000 1000	0000 0000 0000 1000	
PSET20		0000 0000 0001 0000	0000 0000 0001 0000	
PSET21		0000 0000 0010 0000	0000 0000 0010 0000	
PSET22		0000 0000 0100 0000	0000 0000 0100 0000	
PSET23		0000 0000 1000 0000	0000 0000 1000 0000	
PSET24		0000 0001 0000 0000	0000 0001 0000 0000	
PSET25		0000 0010 0000 0000	0000 0010 0000 0000	
PSET26		0000 0100 0000 0000	0000 0100 0000 0000	
PSET27		0000 1000 0000 0000	0000 1000 0000 0000	

Table B-1. Graphics Language Status Words (Cont'd)

Status Words	Description	Mask 15 0	Pattern 15 0	Record Type (Record Field)
PSET28	Specified bit is set (= 1)	0001 0000 0000 0000	0001 0000 0000 0000	Packed Digital (AV)
PSET29		0010 0000 0000 0000	0010 0000 0000 0000	
PSET30		0100 0000 0000 0000	0100 0000 0000 0000	
PSET31		1000 0000 0000 0000	1000 0000 0000 0000	
RESET	Reset	1000 0000 0000 0001	0000 0000 0000 0000	Digital (DS) Device (DS)
RESETALM	Reset Alarm State	1000 0000 0000 0010	0000 0000 0000 0000	Digital (DS)
SCANOFF	Point not Scanned	1000 1000 0000 0000	0000 1000 0000 0000	Analog (AS) Digital (DS) Device (DS)
SENSORALM	Sensor Alarm	1000 0000 1000 1100	0000 0000 1000 1100	Analog (AS)
SENSORMODE	Sensor Mode	1000 0000 0001 0000	0000 0000 0001 0000	Analog (AS)
SET	Set	1000 0000 0000 0001	0000 0000 0000 0001	Digital (DS) Device (DS)
SETALM	Set Alarm State	1000 0000 0000 0010	0000 0000 0000 0010	Digital (DS)
TOGGLE	Toggle	1100 0000 0000 0000	0100 0000 0000 0000	Analog (AS) Digital (DS) Device (DS) Drop Status (FA)
UPDATETIME	Update Clock	0000 0100 0000 0000	0000 0100 0000 0000	Drop Status (FA)
WORSE	Worse	1000 0000 0000 0011	0000 0000 0000 0010	Analog (AS)

A

B

C

D

T

U

V

W

X

Z