Graphics Language Reference Manual For Ovation 2.4

OW240_91

Revision 0

March 2005

Contents

S E C T I O N 1

INTRODUCTION TO THE OVATION GRAPHICS LANGUAGE

In This Section

1.1 WHAT IS THE GRAPHICS LANGUAGE?

The Graphics Language is the source language rules and commands used to create graphic items. 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.

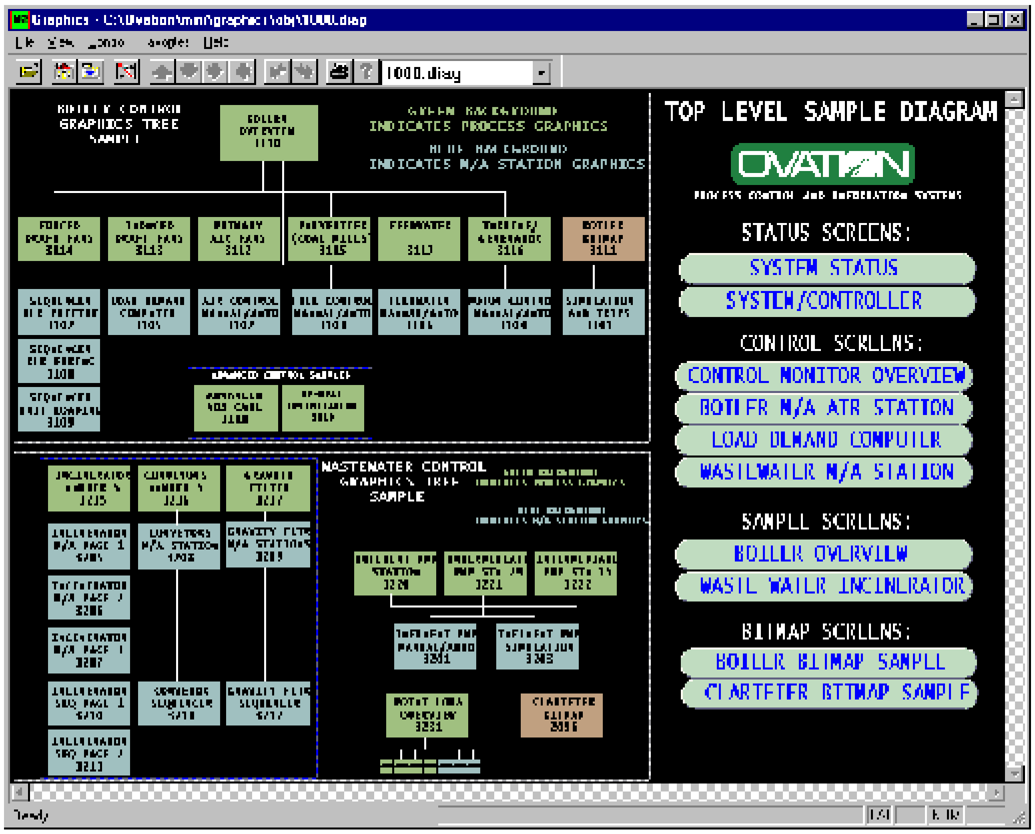


Figure 1: Process Diagram Displayed on Operator Station

SECTION 2

UNDERSTANDING PROCESS POINTS AND CONDITIONALS

In This Section

2.1 ENTERING PROCESS POINT NAMES

Point record information is required in many GBNT windows (GBNT:Bar, GBNT:Process Point, GBNT:Poke, and so forth). This section defines the valid point names and record fields that can be used. (See *Ovation Developer Studio User Guide* and *Ovation Record Types Reference Manual.*)

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

- Process point names defined in the Point Builder. (See *Ovation Developer Studio User Guide*.)
- Dummy points (unresolved points)
- $Pointer variables ($P, $G, $H, $, $D, $O, and $S)
- Record fields
- OPC points

All of the above types of points can be used in graphics commands wherever point names (or point name and record field pairs) are used.

2.1.1 WHAT IS A VALID POINT NAME?

A fully qualified point name in the Ovation system uses the following format:

“name.unit@network”

This fully qualified name consists of three parameters:

- A 16-character maximum point name.
- A six-character maximum unit name.
- An eight-character maximum network name.

In a multiple network implementation, the network name segment must be one of the following:

- actual name of the local or remote network
- any of the defined network aliases (the alias for the remote or local network)

In addition to the above parameters, the "." and "@" are required and are reserved characters. Therefore, the maximum number of characters in a fully qualified point name is 32.

Characters in the name portion may be any combination of alphabetic, numeric (0 - 9), or special punctuation characters. The only characters that are not allowed in the name are: *, , @, ., %, $, ~, \, ", ', comma, space. Note that the "." and "@" are part of a fully qualified point name, but are not allowed in the name portion of the fully qualified point name. The name portion cannot be a reserved graphic word (for example: RECTANGLE, COLOR, UP, HORZ, etc.). The unit and network portions of the fully qualified name are composed of alphanumeric characters only.

Point names are **not** case sensitive. That is, "Testai01.unit1@wdpf" is the same as "TESTAI01.UNIT1@WDPF".

Fully qualified point names **must** be delimited by backslashes (\) when used in graphics since the "." and "@" characters are not alphanumerics. Any point name that includes a non-alphanumeric character or that does not begin with an alpha (A - Z) character must be delimited by backslashes in a graphic. For example, you must enter "\Testai01.unit1@wdpf\" for a point name instead of "Testai01.unit1@wdpf". Even when not using fully qualified point names, it is a recommended practice to use backslashes.

Note: *The backslash characters are NOT included in the characters making up the point name. There is no space between the backslashes and the point name characters.*

USES OF FULLY QUALIFIED AND NON-FULLY QUALIFIED POINT NAMES

Database point names are not required to be fully qualified in a graphic. Different plant situations determine the need for fully qualified point names in a multiple network system. Consider the following:

1. In situations where you want to copy the same graphic to multiple units/networks and the points being used are defined in all the units/networks, you do NOT have to fully qualify the point name. When a point name is not fully qualified, the local network is always assumed.

   For example, you copy a graphic with point A100 from unit1/network1 to unit2/network1 and to unit1/network2. If A100 exists in all three places, then you would **not** want to fully qualify the point name, since each unit/network will read the value of A100 on its own unit/network. If A100 were fully qualified, when it was copied to the other units/networks, it would not read the value on its local network. When a point name is not fully qualified, the local network is always assumed.

2. In situations where you are building a graphic and you want to monitor a point from another network, you must fully qualify the point name. In this way, there is no ambiguity when receiving values from the point.

   For example, you build a graphic on unit1/network1 that monitors point A100 on unit1/network2. Since the local network is always assumed when a point is not fully qualified, you must specify the point as \A100.init1@network2\ so that it is being read from the proper network.

2.1.2 What are Dummy Points (Unresolved Points)?

Dummy points (also known as unresolved points) are valid point names that do not exist in the Ovation database. (See *Ovation Developer Studio User Guide*.) These may be fully qualified with a unit and network name, or they may be unqualified. If the dummy point name does not begin with an alphabetic character, or if it contains any characters other than alphanumerics, it must be delimited with backslashes. If it is a fully qualified name, it must be delimited with backslashes.

Dummy points are used to enable a diagram to be compiled and displayed before the points are added to the Ovation database. When dummy points are displayed in a diagram, they are inactive and have no value. Therefore, final testing of the diagram cannot take place until the dummy points are added to the Ovation database, and the diagram is recompiled and redisplayed. All dummy points should be resolved into database points **before** the diagram is executed at runtime.

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 Ovation database cannot be performed for dummy points. Therefore, when dummy points are added to the Ovation database 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, you 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 loaded and/or saved, GBNT may display a warning message saying that dummy points exist in the diagram. This is not done automatically. You must chose to be warned of dummy points by configuring the Unresolved Points Warning function. 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 as OFF32) would be handled as a dummy point named OFF32. Therefore, unintended dummy points in the list indicate errors in the diagram.

2.1.3 How do I Use Pointers?

The Graphics Language recognizes the following valid pointers: $P, $G, $H, $W, $O, 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 pointer variables before the diagram displays at runtime. $G pointers reference points defined externally to the diagram in the system group library file. $P and $H pointers reference a segment offset in the computer memory in the Operator Station. Pointers are identified by a dollar sign ($).

$P Pointers (Scratch Pad Pointer)

The “$P” pointers (valid range = $P1 through $P99) point to locations in memory segments. A maximum of 99 $P pointer variables can be defined in a graphic. $P pointer variables must be initialized to an existing segment and offset in memory using the POINTER command or the PTR_EQUAL command. Data may be passed between graphics using these pointer variables. For example, an area of computer memory is reserved in the Operator Station to store data received from another drop (such as general message data). Using $P pointer variables allows you to access and display this data by setting a pointer to begin at a specified segment and offset within the reserved area.

$P variables cannot be combined with the standard two-character ASCII record fields (for example: AV, RS, 1W, and so forth) in the source syntax since $P pointers do not represent actual process points in the database. Instead, $P pointer variables have an associated $offset notation. The valid $offsets for $P pointers are defined below:

- $I*n* — represents a 32-bit integer beginning at an offset of n bytes from the associated $P pointer location initialized by the POINTER or PTR_EQUAL command. The “n” must be a multiple of 4 (for example: $I0, $I4, $I8, $I12, and so forth).
- $R*n* — represents a 32-bit floating point real number beginning at an offset of n bytes from the associated $P pointer location initialized by the POINTER or PTR_EQUAL command. The “n” must be a multiple of 4 (for example: $R0, $R4, $R8, $R12, and so forth).
- $B*n* — represents a byte beginning at an offset of n bytes from the associated $P pointer location initialized by the POINTER or PTR_EQUAL command (for example, $B0,$B1,$B2, and so forth).
- $S*n* — represents a 16-bit integer beginning at an offset of n bytes from the associated $P pointer location initialized by the POINTER or PTR_EQUAL command. The “n” must be a multiple of 2 (for example: $S0, $S2, $S4, $S6, and so forth).
- $A*n*X*i* — represents an ASCII string of length *i* beginning at an offset of *n* bytes from the associated $P pointer location initialized by the POINTER or PTR_EQUAL command

where:

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

Sample $P pointer variables and associated $offsets are: $P1 $I0, $P23 $R24, $P99 $A10X32, and so forth. Note that you must be aware of the size of the segment that you initialize your $P pointer to using the POINTER or PTR_EQUAL command before/while attempting to access data within that segment. For example, if you assign $P1 to a segment that is only 256 bytes long, unpredictable results occur if you try to access $P1 $B300 because you are accessing memory beyond the total size of the segment to which you are pointing. A list of available memory segments and their associated sizes is available.

$G Pointers (Group Pointer)

The “$G” pointers (valid range = $G1 through $G250) are used to reference the group points stored in the current point group. The Group Builder program in the Ovation Developers Studio is used to build/modify the system point groups. (See *Ovation Developer Studio User Guide*.) A given point group can have a maximum of 250 points associated with it ($G1 represents the first point in the current group, $G2 represents the second point, and so forth). The $G1 pointer is reserved for Emerson use; however, you may use $G2 - $G250 in your graphics. Since $G pointers reference actual database points, use the standard two-character ASCII point record fields (for example: AV, RS, 1W, and so forth) with them wherever a point name/record field pair is required in a graphic.

$H Pointers (Highway Pointer)

The “$H” pointers (valid range = $H1 through $H99) are used to reference database points indirectly by looking at a system ID stored in computer memory at the Operator Station. It is a precondition of using $H pointers that when you initialize the $H pointer in memory using the POINTER or PTR_EQUAL command, you must point it to a system ID stored in computer memory somewhere. You must make sure that the $H pointer points to a database point system ID; otherwise, the $H pointer will not work. Since $H pointers reference actual database points, use the standard two-character ASCII point record fields (for example: AV, RS, 1W, and so forth) with them wherever a point name/record field pair is required in a graphic.

$W Pointers (Window Pointer)

The “$W” pointers (valid range = $W1 through $W99) are used in WINDOW diagrams (diag_type parameter on the DIAGRAM command = WINDOW). The $W pointers mark generic points that will be replaced with valid process points when the window diagram is displayed. A window diagram is displayed either as a result of a DIAG_DISP command, a POKE_FLD (type 8) command, or an OL_BUTTON (poke functionality, type 8) command. Note that all of these commands take a list of database point names as a parameter in the source syntax.

When the window diagram is displayed, the first point passed in the point names list via any of the above commands is substituted for all occurrences of $W1 in the window, the second point passed is substituted for all occurrences of $W2 in the window, and so on up to $W99. Window graphics are often implemented as templates so that the same window can be called from multiple graphics with different sets of points. Since $W pointers will be replaced with actual database points, use the standard two-character ASCII point record fields (for example: AV, RS, 1W, and so forth) with them wherever a point name/record field pair is required in a window graphic. Note that when you call up a window graphic, you only pass the point name to the window (not a point name/record field pair).

The record field is coded into the window graphic and cannot be passed into the window. It is your responsibility to code the window graphic with the appropriate record field for whatever type of point will ultimately be passed to the window to replace the $W pointer. At runtime, on the Operator Station, if the point passed for any $W pointer in the window is incompatible with the coded record field for that instance of the $W pointer, an error is sent to the Error Log window, and the gcode using that $W pointer in the window is skipped (that is, not drawn at runtime).

$D Pointers (Macro Pointer)

The “$D” pointers (valid range = $D1 through $D99) are used in macro diagrams. The $D pointers mark generic points that should be replaced with valid process points when the macro is added to a graphic. Valid database points, any of the $pointer variables, dummy points, and/or OPC points can be passed to replace $D points in a macro file.

The MACRO graphics command takes a macro diagram number, and a list of up to 99 process points as parameters. $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.

Since database points, $W, $G, $H, or $P pointers can be passed to replace $D pointers in the macro, $D pointers can be paired with either the standard two-character ASCII point record fields (for example: AV, RS, 1W, and so forth) OR the $offsets (for example: $I0, $R24, $B3, $A10X16, and so forth) wherever a point name/record field pair is required. Note that when you call up a macro graphic, you only pass the point names to the macro (not the point name/record field pairs). The record fields are coded into the macro graphic and cannot be passed into the macro. It is your responsibility to code the macro graphic with the appropriate record field for whatever type of point will ultimately be passed to the macro to replace the $D pointer.

OPC points have no associated record field. When you create an item in a macro graphic using a $D pointer, even if you do not pair a record field or $offset with the $D pointer, GBNT will automatically assign a default record field of “AV” to the $D pointer for you. GBNT will automatically strip any record field paired to a $D pointer from the macro at compile time if/when you pass an OPC point to the macro, so no error will be generated when substituting an OPC point for a $D pointer/offset pair in a macro.

$O POINTERS (ONSCREEN POINTER)

The $On pointers (valid range = $O1 through $O99) are used as a programming tool to pass a two-character ASCII record field to application program SEND_CA (#80) using the POKE_FLD (type 7 or 23), RUN_PROGRAMS, FUNC_KEY, or OL_BUTTON (type 7 or 23) command.

Application program #80 performs a change attributes function on some point in the database. You must pass the two-character ASCII point record field that you want to change to this program. The $On pointer is used to do that. Valid application program arguments include strings, integers, real numbers, set variables, status identifiers, and point name/record field pairs. A stand-alone record field is not a valid argument to an applicable program; however, a point name/record pair is a valid argument.

$S POINTERS (ENTRY FIELD POINTER)

The $S pointers (valid range = $S1 through $S99) are used as placeholders for point names when only a record field is needed by an application program. They are used with FUNC_KEY, POKE_FIELD (type 7 and 23), RUN_PROGRAMS, and OL_BUTTON (poke type 7 and 23) commands. The applications programs that typically use $S pointers are SEND_GENMSG (67) and SEND_GENMSG_NETWORK (167).

For example, you define an entry field on the screen. The entry field number (buffer number) is used as the Select Pointer or type number. If the entry field was defined to use buffer number 5, then, to pass the value record field (AV) of that point in the entry field to a function key or Poke Type 7 program, you would write: **$S5 AV**.

2.1.4 Using Record Fields

Record fields are predefined in the Ovation system. These record fields vary according to the type of the process point (analog, digital, device, packed group, and packed digital), and are denoted by a two-character ASCII name (AV, DS, 1W, LL, ED, and so forth). A default record field is defined for each point type. If you do not enter a record field in a graphics command when a process point/record field is required, the default record field for that type of point is assumed.

Record fields are applicable to database points, dummy points, $W, $G, $H, $O, and $D pointers. The $offsets ($I0, $R4, $S0, $B3, $A0X16, and so forth) are only applicable to ***$P*** (see page 6) and ***$D*** (see page 7) pointer variables. OPC points do not have associated record fields or $offsets.

The following table summarizes the relationship between the type of point and the record field (or $offset) expected in the Ovation system. (See *Ovation Record Types Reference Manual*.)

Point Types with Associated Record Fields

Type of Point	**Associated Record Field (or $Offset)**	**Default Record Field ($Offset)**
Database point	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth)	Depends on the type of point.
Dummy point	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth)	None
OPC point	None	Not Applicable
$P1 - $P99	$Offset (for example: $I0, $R4, $S0, $B3, $A0X16, and so forth)	$B0
$G1 - $G250	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth)	None
$W1 - $W99	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth)	None
$D1 - $D99	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth) ***OR*** $Offset (for example: $I0, $R4, $S0, $B3, $A0X16, and so forth)	None
$O1 - $O99	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth)	None
$H1 - $H99	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth)	None
$S1 - $S99	Two-character ASCII record field (for example: AV, 1W, HL, LL, and so forth)	None

Note: *GBNT will not check for valid record types on points originating from a remote network.*

2.1.5 What are OPC Points?

OPC (**O**le for **P**rocess **C**ontrol) points are used to access data on OPC servers. This data is external to the Ovation System.

OPC point names (or tags) in graphics will be of the following format:

```
OPC$servertag$pointtag
```

where:

OPC$	The first four characters of any OPC name must be “OPC$”
servertag	Logical name of the OPC server. It is comprised of alphanumeric characters only.
$	Mandatory placeholder
pointtag	Name of the point or tag. This portion can contain any characters (including spaces and backslashes).

The entire OPC name is limited to 130 characters. The maximum source line length in a graphic is 132 characters, and OPC point names must always be delimited by backslashes (therefore, 132 -2 = 130).

The entire OPC point name is **case sensitive**. The name “OPC$server1$A100.AV” is different from “OPC$server1$a100.av”. Also, the first four characters of an OPC name must be “OPC$” (**not** “opc$”). Note that this is different than database point names, dummy point names, and $pointer variables, which are all **case insensitive**.

OPC point names must always be delimited by backslashes in graphic source commands and in entry fields on the GBNT windows that expect a point name or point name/record field pair.

OPC points do not have associated record fields (neither the two-character ASCII record fields or the $pointer offset notation). You may pass an OPC point anywhere in a graphic where a point name or point name/record field pair is expected, but do not append a record field after it (even if the entry field label or source syntax label indicates doing so). Enter the OPC point name as defined (delimited by backslashes) when specifying an OPC point name.

Since backslash characters are used to delimit OPC point names (as well as all other point names), the use of a backslash character in the pointtag portion of an OPC names is not recommended. However, if backslashes are used in the pointtag portion, two backslashes must be used. This is done so that GBNT can differentiate between the delimiting backslash that marks the end of the OPC name and the ones used in the pointtag portion. Any even number of consecutive backslashes within the pointtag portion will be considered part of the name and not the terminating backslash.

For example, if the pointtag portion exists as “A100\SYSTEM1\TEST\AV” in the OPC Server, it must be written in the graphics source file as follows:

```
\OPC$servertag$A100\\SYSTEM1\\TEST\\AV\
```

If it were written as “\OPC$servertag$A100\SYSTEM1\\TEST\AV\”, GBNT would assume that the backslash after A100 is the terminating backslash (that is, that the OPC point name is “\OPC$servertag$A100\”) and would process the rest of the name as extra data on the source line after the OPC name.

Rules for using OPC Points in Graphics

1. OPC points cannot be used with a POKE FLD (poke type = 0) command to display Ovation system point information since the OPC point is not part of the Ovation system. Similarly, you cannot poke on the display of a PROCESS PT command built with an OPC point.
2. In the PROCESS PT command, if an OPC point is specified for the point name argument, no record field (or record field conditional) will be allowed, since OPC points do not have associated record fields.
3. OPC points cannot be used in conjunction with CASE or QUALITY conditional expressions in graphics.
4. OPC point names are not stored in the database with the graphic when running GBNT within the Ovation Developer Studio. (See *Ovation Developer Studio User Guide*.) Thus, you cannot do a search or a “where used” function on OPC points as you can for database points.
5. OPC point names do not display in the list of dummy points in GBNT. There is no support for getting/printing a list of OPC points used in a graphic in GBNT.

2.2 Conditional Expressions

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*** (see page 13)
- ***Compound Expressions*** (see page 17)
- ***Case Expressions*** (see page 19)
- ***Quality Expressions*** (see page 21)
- ***Set Expressions*** (see page 23)

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, and so forth).

Specifying conditionals is always optional. GBNT does not evaluate conditionals. Conditionals are only evaluated at runtime.

In this manual, conditionals are specified in a certain way. In the source syntax for any command that can have a condition, the convention is to denote the conditional as “**[conditional]**”. The “[“ and “]” do not actually appear in the source command; they are used here to illustrate that what is within them is optional.

2.2.1 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.
CIRCLE	Line pattern and/or fill pattern.
COLOR	Color name and blink state parameter.
DYNAMIC_LINE	Line pattern.
DYNAMIC_POLYGON	Line pattern and/or fill pattern.
ELLIPSE	Line pattern and/or fill pattern.
LINE	Line pattern.
MULTI_TEXT	List of strings (same number as entered for default case)
OL_BUTTON	Shape name or label string.
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.*

2.3 Simple Expression

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

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

where:

operand1, operand2	Any of the following: ▪ Integer or real constant (123, 45.67, and so forth) ▪ Point name/record field (A100 AV, $P1 $I0, \OPC$server$pt.name\, forth) ▪ Status identifier (HDWRFAIL, OFF0, and so forth) ▪ Two-character ASCII record field (AV, TV, EQ, and so forth) ▪ Set variable (SET1, SET12, and so forth) ▪ $SETn variable ($SET1,$SET2, and so forth) ▪ $CONSTn variable ($CONST1,$CONST2, and so forth) ▪ $STATUSn variable ($STATUS1,$STATUS2, and so forth)
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)
conditional_value	Value which is used if simple_expression evaluates as True (must be of appropriate type for applicable command parameter).

Rules

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

   A100 BV = 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. 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.

5. If one operand is a status identifier or a $STATUS variable, the other operand must be a point name/record field pair, where the record field is of type integer. If no record field is specified, GBNT defaults the record field per the point type. If the default chosen is AV or A2, it is automatically changed to 1W. Specify the record field in expressions to avoid problems with GBNT defaulting the record field incorrectly. For example:

   A100 AS = ALARM
   D200 DS = RESET

6. If one operand is a two-character ASCII record field, the other operand must be a point name/record field pair, where the record field should be the 'RT' record field. Using a two-character ASCII record field as a conditional operand is specific to expressions of the following format:

   A100 RT = LA
   D200 RT = LD

7. If one operand is a set variable or a $SET variable, the other operand must be an integer or a point name/record field pair where the record field is of type integer or a byte. For example:

   Set5 > 50
   Set20 = A100 GS

8. 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

```
COLOR FG cyan (A100 AV < 50) yellow
RECTANGLE 4543 4543 6591 8343 4 solid unfilled
```

In this example:

COLOR	command
cyan	default_value

See below:
operand1 = A100 AV
rel_op = <
operand2 = 50
	simple_expression

yellow	conditional_value

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.

Example 2

```
COLOR FG red ((A100 AV > 50) AND (A100 1W = HDWRFAIL) OR (D200 =
ALARM)) blue
```

```
RECTANGLE 4543 4543 6591 8343 4 solid unfilled
```

In this example:

COLOR	command
red	default_value
See below: first operand1 = A100 AV first rel_op = > first operand2 = 50 first logical_op = AND second operand1 = A100 1W second rel_op = = second operand2 = HDWRFAIL second logical_op = OR third operand1 = D200 third rel_op = = third operand2 = ALARM	simple_expression
blue	conditional_value

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 1W = HDWRFAIL or D200 = ALARM, the foreground color for the rectangle will be blue. Otherwise, the foreground color for the rectangle will be red.

2.4 Compound Expression

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

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

where:

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.

Rules

1. The overall compound expression must be delimited by curly braces "{ }".
2. If one operand in an expression is an integer, the other operand must be a point name/record field pair, a set variable, or a $SETn variable. For example:

   A100 BV = 10
   SET2 = 20
3. 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
4. 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

5. If one operand is a status identifier or a $STATUS variable, the other operand must be a point name/record field pair, where the record field is of type integer. If no record field is specified, GBNT defaults the record field per the point type. If the default chosen is AV or A2, it is automatically changed to 1W. Specify the record field in expressions to avoid problems with GBNT defaulting the record field incorrectly. For example:

   A100 AS = ALARM
   D200 DS = RESET

6. If one operand is a two-character ASCII record field, the other operand must be a point name/record field pair, where the record field should be the 'RT' record field. Using a two-character ASCII record field as a conditional operand is specific to expressions of the following format:

   A100 RT = LA
   D200 RT = LD

7. If one operand is a set variable or a $SET variable, the other operand must be an integer or a point name/record field pair where the record field is of type integer or a byte. For example:

   Set5 > 50
   Set20 = A100 GS

8. 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.

EXAMPLE

```
COLOR FG magenta {(A100 AV > 50) blue (A100 1W = HDWRFAIL) red} BG
white

CIRCLE 7671 6349 2145 3 solid unfilled
```

In this example:

COLOR	command
magenta	default_parameter
(A100 AV > 50)	simple_expression1
blue	conditional_value1
(A100 1W = HDWRFAIL)	simple_expression2
red	conditional_value2

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 1W = HDWRFAIL, the circle will be red. If both expressions evaluate as false, the circle will be magenta.

2.5 CASE EXPRESSION

Case Expression conditionals are used as a shorthand method of implementing a compound expression in which each simple expression compares the same point name and record field pair to a range of values. The range of values is specified by an initial value and an integer increment. If the point value falls anywhere between two successive increments, then that is the conditional value that is used at runtime.

The case conditional is equivalent to the following compound conditional:

```
{((ptname & recfld >= value) AND (ptname & recfld < value + (1*inc))
) cond_value1

((ptname & recfld >= value + (1*inc)) AND (ptname & recfld < value +
(2*inc)) ) cond_value2

.
.
.

((ptname & recfld >= value + ((N-1)*inc)) AND (ptname & recfld <
value + (N*inc)) ) cond_valueN}
```

SYNTAX

```
(CASE) pt_name rec_fld initial_val increment count cond_val
```

where:

(CASE)	Keyword enclosed in parentheses.
pt_name	Process point name, dummy point name, or $pointer variable (OPC points are **not** supported here).
rec_fld	Two-character record field name (1W, AV, ID, and so forth) OR $offset identifier ($B0, $I4, $R0, and so forth). This **cannot** be an ASCII type record field.
initial_val	Point value that all increments are relative to.
increment	Integer to add to initial_value to specify next conditional value.
count	Total number of conditional values specified.
cond_value	Parameter to use at runtime in place of the default conditional parameter when the point value is within the range specified by the initial_value and the increment.

RULES

1. The CASE keyword must be enclosed in parenthesis.
2. The CASE keyword is case-insensitive. That is, you can enter “Case” or “case.”

EXAMPLE

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

In this example:

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

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.

- 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), you 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.

2.6 Quality Expression

Quality expressions allow the value of a conditional parameter to change based upon the quality of a process point variable. The default parameter is displayed when the quality of the input point is good. You must specify a conditional value for each of the following four qualities: fair, poor, bad, and timed-out. The input point must be a dummy point, a $D, $G, $H, or $W pointer variable, or an analog or digital database point. $P and OPC points are **not** supported here. Database points other than analog or digital points are not supported.

Syntax

```
(QUALITY)  pt_name  fair_quality_cond_value

                    poor_quality_cond_value

                    bad_quality_cond_value

                    timedout_quality_cond_value
```

where:

(QUALITY)	Keyword enclosed in parentheses.
pt_name	Point name or pointer variable.
fair_quality_cond_value	Conditional value which is used if point has fair quality.
poor_quality_cond_value	Conditional value which is used if point has poor quality.
bad_quality_cond_value	Conditional value which is used if point has bad quality.
timedout_quality_cond_value	Conditional value which is used if point has timed-out quality.

Rules

1. The QUALITY keyword **must** be enclosed in parenthesis.
2. The QUALITY keyword is case-insensitive. That is, you can enter “Quality” as well as “quality.”
3. You must specify all four conditional values (fair quality, poor quality, bad quality, and timed-out quality).

EXAMPLE

```
COLOR FG cyan (QUALITY) D200 green yellow red blue
RECTANGLE 4543 4543 6591 8343 3 solid unfilled
```

In this example:

COLOR	command
cyan	good_cond_value
(QUALITY)	required_keyword
D200	pt_name
green	fair_quality_cond_value
yellow	poor_quality_cond_value
red	bad_quality_cond_value
blue	timedout_quality_cond_value

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.

2.7 Set Expression

Set Expression conditionals allow the value of a parameter to change based upon the value of a set variable (SET1 through SET255). Set variable values are integers in the range of 0 through 32,767. The value of the set can be defined by graphics ***application programs*** (see page 223) or by the ***SETVAL command*** (see page 201). Note that sets are global to the current main and window diagrams. Set1 in the main screen is the same as Set1 in the window.

In this type of conditional, the value of the set variable is compared with consecutive integers starting at 0 and going up to the number of conditional values passed to this conditional. If the value of the set variable is 0 (or is greater than the number of conditional values), the default value is used. If the value of the set variable is 1, the first conditional value is used. If the value of the set variable is 2, the second conditional value is used, and so on.

Syntax

```
(SETx) N conditional_values
```

where:

(SETx)	Set variable enclosed in parenthesis ***OR*** $SET variable enclosed in parenthesis (x = 1 through 255)
N	Number of conditional values listed (N>0).
conditional_values	Parameter to use at runtime in place of the default conditional parameter when the value of SETx >=1 and the value of Setx <= N.

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 $SET specification is used to pass the set into a macro. If you want to pass the set to be evaluated in the conditional into the macro instead of hard-coding the set, you specify “($SETn)” instead of “(SETn)”.
3. The SET keyword **must** be enclosed in parenthesis.
4. The SET keyword is case-insensitive. That is, you can enter “Set” or “set.”

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:

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

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.

SECTION 3

GRAPHICS LANGUAGE RULES AND TERMS

In This Section

3.1 Definitions of Terms for Graphics Language

This document assumes that you are already familiar with certain operating system and window manager terms. Before beginning to use GBNT, 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.

Bitmap text is defined by the eight font point sizes found in the **fonts.txt** file. You may change the font sizes (valid range for font sizes is 1 pt. to 200 pt. text type) as long as eight sizes are defined in the file. (See *Ovation Graphics Builder User Guide*.)

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.

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. You 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 3 through 16,383.

Another type of Vector font is the **Ovation Vector Font**. This font may be used for the following commands: TEXT, MULTI_TEXT, GTEXT, TIME, DATE, PROCESS_PT, ENTRY_FLD, OL_BUTTON (with text label), OL_EVENT_MENU, and OL_CHOICE (with text label).

Ovation Vector Font has an associated line width of 1 - 16 pixels. Note that this line width is independent from the GBNT:Line Width dialog. Also, this font cannot be displayed in bold or italic typeface. It is a fixed-width font which is implemented internally, so it will not be restricted in the way it resizes.

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.

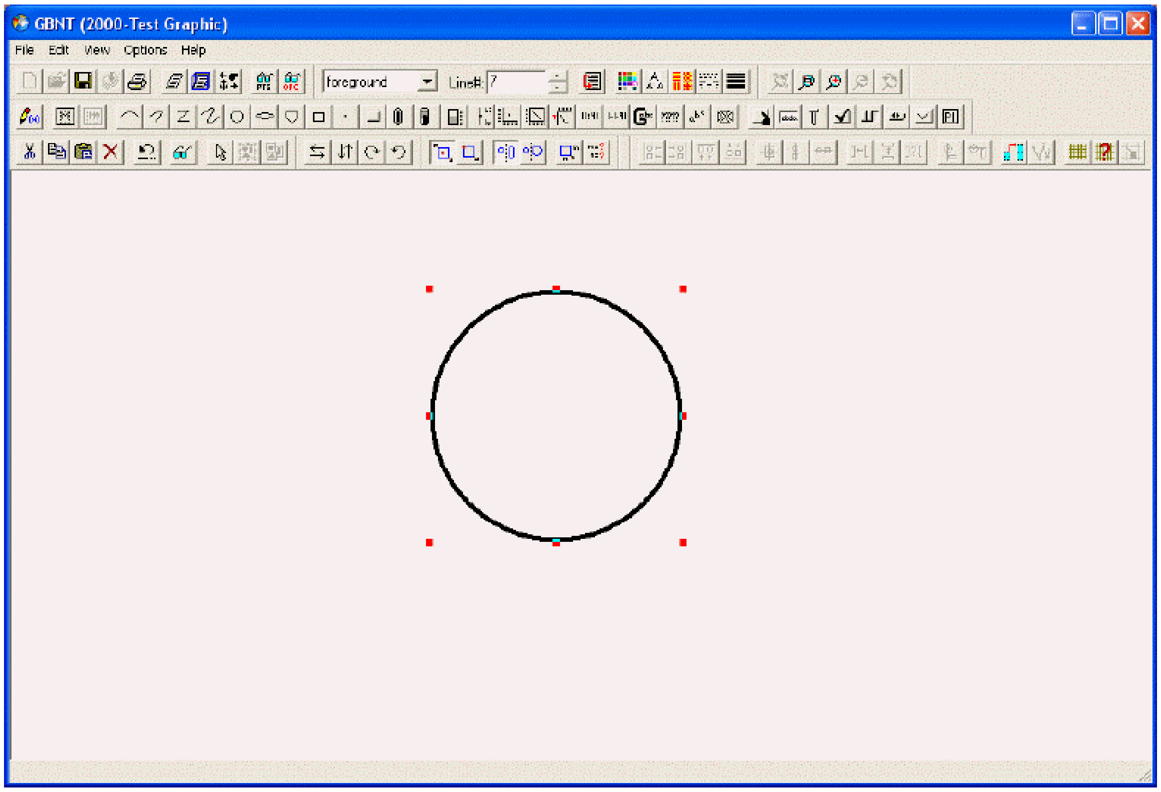
The overstrike option should always be used with text conditionals. A conditional may be written to have text change on the process diagram depending on the conditions in the plant. If the overstrike option is not used, the new text will not cover the old text (the lines for the old text will still be 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.

***Note:** When GBNT attempts to create/get a font of a particular size, it is only a request. The font mapper attempts to find the font in the requested size, and then gives the closest match if the exact font is not found. GBNT has no control over this. Sometimes the match is exact; sometimes it is not even close. The font mapper searches by height first and then by width. If the height exists, the font mapper returns that font whether it matches the width or not.*

Text sizes may change on a zoom/resize operation. This means that if you interactively resize text, the text drawn after you release the mouse pointer may not represent what the outlining rectangle handles lead you to believe the size would be while dragging for resize. GBNT may have requested a certain size font, but received a different size from the font mapper. This also means that if you draw a box around text and then resize the diagram, zoom the diagram, or group the box and text and resize the group, the text may not always be within the box after the resize. During the resize, GBNT requests a certain font size to keep the text in the box but may receive another size from the font mapper.

Note that the information above does NOT apply to Ovation Vector Font since it is an Ovation font - not a Windows font. It will size exactly per the requested size.

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 following figure shows how an outlining rectangle would appear on the GBNT interactive editor window. If a line were drawn to connect these handles, it would represent the outlining rectangle around the item.



Line Width - GBNT offers 16 line widths from which to choose. The order of the line widths corresponds to an array index that is used in the source editor to specify line widths. Neither the size of the line widths nor the line width array index is configurable. Below is a list of the line width array index numbers and their corresponding pixel line width. The left-hand column in the table is what is shown in the source syntax; the right-hand column is the number of pixels used to draw the item on the screen.

Array Index Numbers	Width of Line in Pixels
1	1
2	2
3	4
4	6
5	8
6	10
7	12
8	14
9	16
10	18
11	20
12	22
13	24
14	26
15	28
16	30

Note: *The line width described above is independent of the line width defined for Ovation Vector Font.*

3.2 Graphics Language Rules

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

1. The ***DIAGRAM command*** (see page 75) must be the first command in the graphics language program (except for comment lines and blank lines).
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 *Ovation Graphics Builder User Guide*.)
3. Comments cannot be embedded within commands.
4. Blank lines may be placed anywhere in the source file (including within a command).
5. You cannot modify a keyword that has already been entered into the source editor.

   For example, you 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 you wish to move/copy a loop, the entire loop must be blocked first. (See *Ovation Graphics Builder User Guide*.)
7. It is recommended that you choose a number as the diagram name. The valid diagram ranges are listed below:

Diagram Number	Diagram Type	Screen Type
1 - 699	System	Window
700 - 989	Customer	Window
990 - 999	System	Window
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 window. 2. Diagram 4999 is a standard blank main window.		

8. Main commands can only be used with the section labels specified in the following table.

Use of Main Commands with Section Labels

Main Commands	**Section Labels**				
	Background	**Diagram**	**Foreground**	**Keyboard**	**Trigger**
ARC	x		x		x
BAR	x		x		x
CIRCLE	x		x		x
COLOR	x		x	x	x
CURSOR	x		x		x
DATE	x		x		x
DEF_FKEY_ GROUP		x			
DEF_MACRO_ PARAMS		x			
DEF_QUAL		x			
DIAG_DISP	x		x		x
DOT	x		x		x
DYNAMIC_ LINE	x		x		x
DYNAMIC_ POLYGON	x		x		x
EF_STATE	x		x		x
ELLIPSE	x		x		x
ENTRY_FLD				x	
FKEY_STATE	x		x		x
FONT	x		x	x	x
FORCE_ UPDATE			x		
FUNC_KEY				x	
GCODE	x		x		x
GTEXT	x		x		x
IF_CHANGED/ ENDIF	x		x		x

Main Commands	**Section Labels**				
	Background	**Diagram**	**Foreground**	**Keyboard**	**Trigger**
IF/ENDIF	x		x		x
IFELSE/ELSE/ ENDIF	x		x		x
LINE	x		x		x
LOAD_FKEY_ GROUP	x		x		x
LOOP/ ENDLOOP	x		x		x
MACRO	x	x	x	x	x
MATH	x		x		x
MULTI_TEXT	x		x		x
OL_BUTTON				x	
OL_CHECKBOX				x	
OL_CHOICE				x	
OL_CYLINDER	x		x		x
OL_EVENT_ MENU				x	
OL_GAUGE	x		x		x
OL_SLIDER				x	
OL_RECTANGLE	x		x		x
PAGE				x	
PLOT	x		x		x
POINTER	x	x	x		x
POKE_FLD				x	
POKE_STATE	x		x		x
POLYGON	x		x		x
PROCESS_PT	x		x		x
PTR_EQUAL	x	x	x		x
PTR_LOOP/ P_ENDLP	x	x	x		x
PTR_MOVE	x	x	x		x

Main Commands	**Section Labels**				
	Background	**Diagram**	**Foreground**	**Keyboard**	**Trigger**
PTR_VALUE	x	x	x		x
RECTANGLE	x		x		x
RUN_PROGRAMS	x		x		x
SETVAL	x		x		x
TEXT	x		x		x
TIME	x		x		x
TREND	x		x		x
TRIG_ON	x		x		x
XY_PLOT	x		x		x

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 the following areas: MAIN, SUBWINDOW, WINDOW, or CONTROL. The valid virtual coordinate range for each type of screen is 0 through 16,383.

MAIN is the primary area of the diagram. You must enter the x, y (position) and w, h (width and height) of the diagram. Main screens can be resized at the Operator Station.

SUBWIN displays a subwindow. On the Windows platform, subwindows are displayed within the main window in a split screen format. You can configure the subwindow to be displayed at the bottom portion of the main screen. The initial width of the subwindow will correspond to the width of the main screen. The initial height will be a fixed 100 pixels. You can resize the width and height of the subwindow, but the aspect ratio of the diagram is maintained.

WINDOW can be called by poke fields to display menus, data entry fields, or other diagrams associated with the main diagram. Windows require you to enter the x, y (position) and w, h (width and height) of the diagram. Once the width and height are set, the operator cannot resize a Window at the Operator Station.

CONTROL diagrams generated by the Control Builder are of this type. You should never create a diagram of this type.

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 GBNT source editor. (See *Ovation Graphics Builder User Guide*.)

12. 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. You 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). You can build the following types of conditional statements:

    - ***Simple Conditional statement*** (see page 13).
    - ***Compound Conditional statement*** (see page 17).
    - ***Case Conditional statement*** (see page 19).
    - ***Quality Conditional statement*** (see page 21).
    - ***Set Conditional statement*** (see page 23).

14. For the commands that use conditional expressions in their syntax, anything enclosed in "[ ]" characters is optional. The "[ ]" do not actually appear in the source command. They are used here only to illustrate that what is within them is optional.

15. Valid point names in the graphics language consist of process point names defined in the Ovation database and the pointer names.

16. Valid record field names in the graphics language consist of standard record fields for more information) and the $offset pointers used with the $P pointer type.

17. 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 and MACRO strings; 50 characters for the OL_EVENT_MENU strings; 30 characters for the MULTI_TEXT, OL_CHOICE, OL_BUTTON, and FONT name strings; and 130 characters for GCODE, POKE_FLD, and PTR_VALUE strings.

3.3 Coordinates

GBNT 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 used to define the location/size of the diagram windows displayed on the Ovation Operator Station monitor screen (DIAGRAM and DIAG_DISP commands). They are also used to specify the spacing between strings for the MULTI_TEXT command. Screen coordinates are classic pixels.

Screen coordinates are defined by the pixel resolution of the CRT monitor screen. These coordinates are used in conjunction with the DIAGRAM, DIAG_DISP, and MULTI_TEXT commands only. (See *Ovation Graphics Language Reference Manual.*)

Virtual Coordinates — Virtual coordinates define the drawing surface of a diagram. The drawing surface is defined to be 16,384 x 16,384 pixels. Therefore, the virtual coordinate range is 0 through 16,383 and 0 through 16,383 where 0, 0 is the upper left corner of the diagram, and 16,383, 16,383 is the lower right corner of the diagram. All of the graphics commands which draw items on the diagram (ARC, CIRCLE, BAR, LINE, and so forth), specify x, y, w, h parameters in terms of virtual coordinates. (See *Ovation Graphics Language Reference Manual.*)

Virtual coordinates are required to implement software pan and zoom. For example, since it is not possible to display a diagram that is 16,384 x 16,384 pixels on a CRT screen which is only 1,152 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 you zoom 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** specify a virtual offset from the previous display item (circle, bar, text, rectangle, and so forth), or from 0, 0 if there is no previous item in the diagram section. Absolute virtual coordinates are specified as 0 through 16,383. Relative virtual coordinates are specified as -16,383 through 16,383. Note that relative coordinates must be enclosed in brackets.

Refer to the following examples:

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 example, the second rectangle would be relative to the first rectangle. The second rectangle would be 1,000 pixels left of the first, and 3,000 pixels below the first rectangle. If you interactively move the first rectangle, the position of the second automatically shifts also. If you delete the first rectangle, the second rectangle becomes relative to the line. If you then delete the line, the second rectangle becomes relative to 0, 0. If you then add a new item after the color command, the rectangle becomes relative to the new item.

For those commands 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).

3.3.1 RULES FOR COORDINATES

The following rules apply to relative coordinates:

1. Items can only be defined with relative coordinates through the source editors (integrated source editor or generic source editor). You cannot interactively draw items with relative coordinates. You 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 main GBNT window.
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). You may specify x and y coordinates independently (x may be absolute and y may be relative). Commands with multiple x, y pairs (LINE and POLYGON) may specify any number of coordinates as relative; only the x, y position of the DYNAMIC_LINE and DYNAMIC_POLYGON may be relative (not the individual x, y coordinates).
4. Relative coordinates may only be used to specify the position of the following commands: ARC, BAR, CIRCLE, DATE, DOT, DYNAMIC_LINE, DYNAMIC_POLYGON, ELLIPSE, GTEXT, LINE, MULTI_TEXT, OL_CYLINDER, OL_GAUGE, OL_RECTANGLE, PLOT, POLYGON, PROCESS PT, RECTANGLE, SHAPE, 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

DYNAMIC_POLYGON [1000] [4000] 3770 2412 \A100\ LL \A100\ HL \A200\ LL
\A200\ HL 5 4 12.45 \A200\ AV \A100\ HL 0 20 \A200\ AV 100 \A200\ HL
3 dashed unfilled

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 [1000] [-2000] 1000 4000 A100 AV A100 ll a100 HL

OL_GAUGE [-3000] [0] 2000 550 LEFT A100 AV 0 100

OL_RECTANGLE [1234] [5678] 2500 2500 INVOKED

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

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** (see page 37).

3.3.2 ORIGINS OF DRAW ITEMS

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 the following table.

Origin by Draw Item

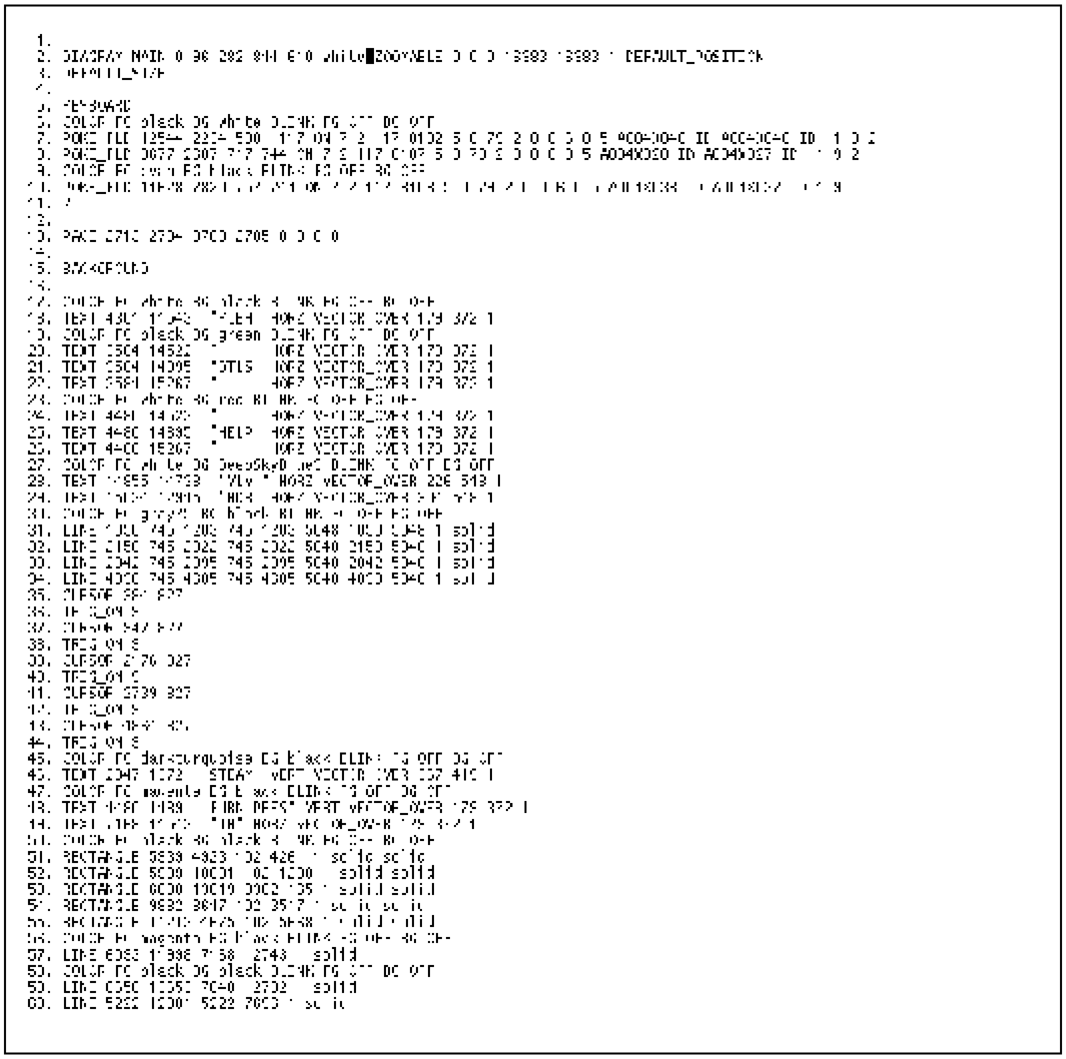
Draw Item	Relative Origin
Arc	Starting point of the arc.
Bar	Upper left corner (down, right or bias).
	Lower left corner (up).
	Upper right corner (left).
Circle	Center.
Date	Baseline position of the first character of the text string (for bitmap and bitmap_over text).
	Upper left corner of the outlining rectangle of the text string (for vector and vector_over text).
Dot	x, y position.
Dynamic_line	Lower left corner of the outlining rectangle.
Dynamic_polygon	Lower left corner of the outlining rectangle.
Ellipse	Center.
Entry Field	Baseline position of the first character of the text string (for bitmap and bitmap_over text).
	Upper left corner of the outlining rectangle of the text string (for vector and vector_over text).
Gtext	Baseline position of the first character of the text string (for bitmap and bitmap_over text).
	Upper left corner of the outlining rectangle of the text string (for vector and vector_over text).
Line	First x, y coordinate of the line.
Macro	Upper left corner of macro outlining rectangle.
Multi_text	Baseline position of the first character of the first text string (for bitmap and bitmap_over text).
	Upper left corner of the outlining rectangle of the entire item (for vector and vector_over text).
OL Button	Upper left corner of button outlining rectangle (x, y from command syntax).
OL Checkbox	Upper left corner of checkbox outlining rectangle (x, y from command syntax).
OL Choice	Upper left corner of choice (x, y from command syntax).
OL Cylinder	Upper left corner of cylinder outlining rectangle (x, y from command syntax).
OL Event Menu	Upper left corner of event menu (x, y from command syntax).

Draw Item	Relative Origin
OL Gauge	Upper left corner of gauge outlining rectangle (x, y from command syntax).
OL Rectangle	Upper left corner (x, y from command syntax).
OL Slider	Upper left corner of slider outlining rectangle (x, y from command syntax).
Plot	Top of plot line (down)
	Bottom of plot line (up)
	Left end of plot line (right)
	Right end of plot line (left)
Poke Field	Upper left corner of poke field.
Polygon	First x, y coordinate of the polygon.
Process Point	Baseline position of the first character of the text string (for bitmap and bitmap_over text).
	Upper left corner of the outlining rectangle of the text string (for vector and vector_over text).
Rectangle	Upper left corner.
Shape	Defined origin.
Text	Baseline position of the first character of the text string (for bitmap and bitmap_over text).
	Upper left corner of the outlining rectangle of the text string (for vector and vector_over).
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).
XY Plot	Lower left corner.

Note: *The origin of a group is defined as the upper left corner of the group’s outlining rectangle.*

3.4 SAMPLE FILE

The following illustrates a partial source program. The required syntax and definition for each graphics language command are defined in the reference pages.



```

DIAGRAM MAIN 0 36 282 844 610 white ZOOMABLE 0 0 0 13983 13983 1 DEFAULT_POSITION
DEFAULT_SIZE

KEYBOARD
COLOR FG black BG white BLINK FG OFF BG OFF
POKE_FLD 12544 2234 500 117 ON 7 2 17 0102 5 0 70 2 0 0 0 0 5 AC0400A0 ID AC0400A0 ID 1 0 2
POKE_FLD 3077 2307 717 744 ON 7 2 117 0107 5 0 70 2 0 0 0 0 5 A004X020 ID AC04X027 ID 1 9 2
COLOR FG cyan BG black BLINK FG OFF BG OFF
POKE_FLD 11878 282 [illegible]
/

PAGE 2710 2704 3700 2705 0 0 0 0

BACKGROUND

COLOR FG white BG black BLINK FG OFF BG OFF
TEXT 4301 11540 "FILE" HORZ VECTOR_OVER 179 372 1
COLOR FG black BG green BLINK FG OFF BG OFF
TEXT 2504 14522 "      HORZ VECTOR_OVER 179 372 1
TEXT 2504 14095 "UTILS  HORZ VECTOR_OVER 179 372 1
TEXT 2581 15267 "      HORZ VECTOR_OVER 179 372 1
COLOR FG white BG red BLINK FG OFF BG OFF
TEXT 4481 14522 "      HORZ VECTOR_OVER 179 372 1
TEXT 4480 14395 "HELP  HORZ VECTOR_OVER 179 372 1
TEXT 4400 15267 "      HORZ VECTOR_OVER 179 372 1
COLOR FG white BG DeepSkyBlue2 BLINK FG OFF BG OFF
TEXT 14855 14733 "Yly " HORZ VECTOR_OVER 226 513 1
TEXT [illegible] "HCR  HORZ VECTOR_OVER [illegible] 1
COLOR FG gray71 BG black BLINK FG OFF BG OFF
LINE 1000 745 1205 745 1205 5048 1000 5048 1 solid
LINE 2150 745 2022 745 2022 5040 2150 5040 1 solid
LINE 2042 745 2095 745 2095 5040 2042 5040 1 solid
LINE 4000 745 4305 745 4305 5040 4000 5040 1 solid
CURSOR 884 827
TRIG_ON 5
CURSOR 947 827
TRIG_ON 5
CURSOR 2176 827
TRIG_ON 5
CURSOR 2739 827
TRIG_ON 5
CURSOR [illegible] 827
TRIG_ON 5
COLOR FG darkturquoise BG black BLINK FG OFF BG OFF
TEXT 2047 1272 "STEAM" VERT VECTOR_OVER 237 410 1
COLOR FG magenta BG black BLINK FG OFF BG OFF
TEXT 4180 1139 "FIRE PRESS" VERT VECTOR_OVER 179 372 1
TEXT [illegible] "IN" HORZ VECTOR_OVER [illegible] 1
COLOR FG black BG black BLINK FG OFF BG OFF
RECTANGLE 5539 4923 102 426 1 solid solid
RECTANGLE 5000 10001 102 1200 1 solid solid
RECTANGLE 6000 10019 1002 135 1 solid solid
RECTANGLE 9632 3617 102 3517 1 solid solid
RECTANGLE [illegible] 1 solid solid
COLOR FG magenta BG black BLINK FG OFF BG OFF
LINE 6050 11398 7158 2743 1 solid
COLOR FG black BG black BLINK FG OFF BG OFF
LINE 6050 11051 7040 2702 1 solid
LINE 5222 12001 5222 7000 1 solid
```

SECTION 4

GRAPHICS LANGUAGE COMMANDS REFERENCE

In This Section

4.1 COMMAND INTRODUCTION

The Graphics Language consists of two types of commands:

- **Section labels** define the major portions of a diagram. Section labels correspond to “Place” in GBNT. (See *Ovation Graphics Builder User Guide*.)
- **Main commands** define the displayable and non-displayable items and their properties. 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.

Note: *Some commands use conditional expressions in their syntax. In the syntax, anything enclosed in “[ ]” characters is optional. The “[ ]” do not actually appear in the source command. They are used here only to illustrate that what they contain is optional.*

The following table lists all of the graphic commands in alphabetical order. The are alphabetically arranged left to right.

Graphic Commands Showing Type and Section Number

Command	Type and Where Found	Command	Type and Where Found
ARC	Main Command	BACKGROUND	Section Label
BAR	Main Command	CIRCLE	Main Command
COLOR	Main Command	CURSOR	Main Command
DATE	Main Command	DEF_FKEY_GROUP	Main Command
DEF_MACRO_PARAMS	Main Command	DEF_QUAL	Main Command
DIAG_DISP	Main Command	DIAGRAM	Section Label

Command	**Type and Where Found**	**Command**	**Type and Where Found**
DOT	Main Command	DYNAMIC_LINE	Main Command
DYNAMIC_POLYGON	Main Command	EF_STATE	Main Command
ELLIPSE	Main Command	ENTRY_FLD	Main Command
FKEY_STATE	Main Command	FONT	Main Command
FORCE_UPDATE	Main Command	FOREGROUND	Section Label
FUNC_KEY	Main Command	GCODE	Main Command
GTEXT	Main Command	IF_CHANGED/ENDIF	Main Command
IF/ENDIF	Main Command	IFELSE/ELSE/ENDIF	Main Command
KEYBOARD	Section Label	LINE	Main Command
LOAD_FKEY_GROUP	Main Command	LOOP/ENDLOOP	Main Command
MACRO	Main Command	MATH	Main Command
MULTI_TEXT	Main Command	OL_BUTTON	Main Command
OL_CHECKBOX	Main Command	OL_CHOICE	Main Command
OL_CYLINDER	Main Command	OL_EVENT_MENU	Main Command
OL_GAUGE	Main Command	OL_RECTANGLE	Main Command
OL_SLIDER	Main Command	PAGE	Main Command
PLOT	Main Command	POINTER	Main Command
POKE_FLD	Main Command	POKE_STATE	Main Command
POLYGON	Main Command	PROCESS_PT	Main Command

Command	**Type and Where Found**	**Command**	**Type and Where Found**
PTR_EQUAL	Main Command	PTR_LOOP/P_ENDLP	Main Command
PTR_MOVE	Main Command	PTR_VALUE	Main Command
RECTANGLE	Main Command	RUN_PROGRAMS	Main Command
SETVAL	Main Command	SHAPE	Main Command
TEXT	Main Command	TIME	Main Command
TREND	Main Command	TRIGGER	Section Label
TRIG_ON	Main Command	XY_PLOT	Main Command

4.2 ARC

The ARC command draws a filled or unfilled circular or elliptical arc. The arc is defined by the dimensions of the outlining rectangle of the circle/ellipse the arc resides on and 2 angles. One angle defines the start point of the arc on the circle/ellipse, and the other angle defines the length/direction of the arc. Negative angles imply counterclockwise direction; positive angles imply clockwise direction. Both angles are defined in degrees, and are relative to the 3 o'clock position from the center of the circle/ellipse. Note that the outlining rectangle around the circle/ellipse the arc lies on is **not** the same as the outlining rectangle around the arc itself.

SYNTAX

```
ARC x y w h angle1 angle2 line_width line_pat [conditional] fill_pat
[conditional]
```

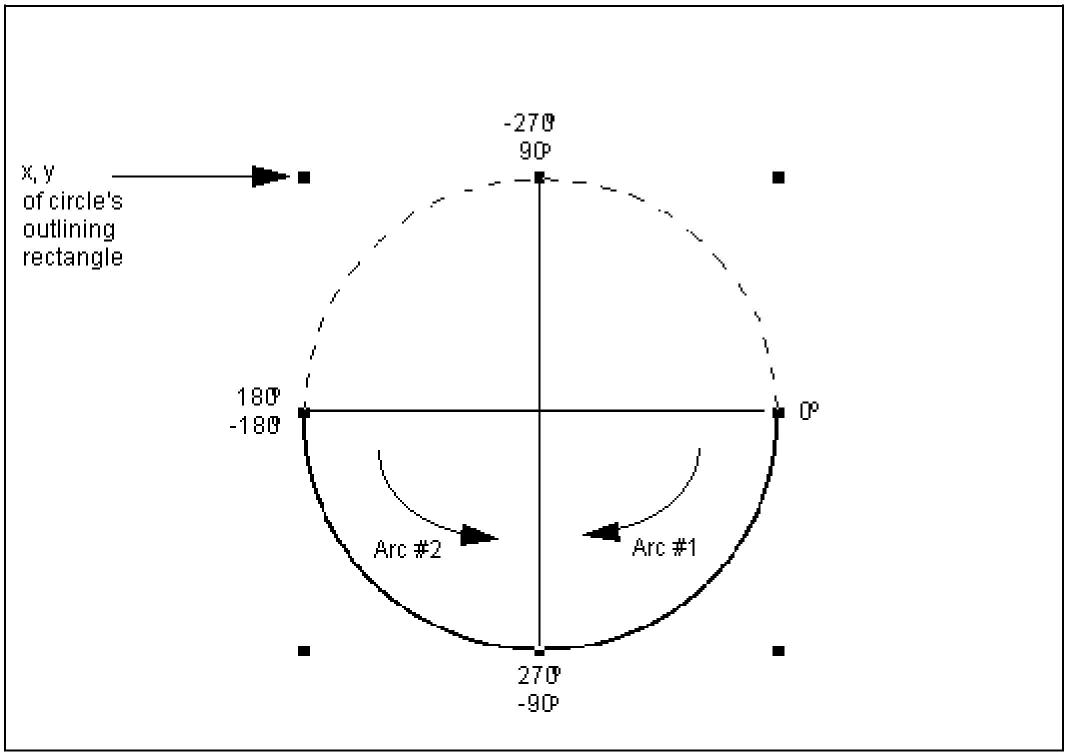
where:

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 = -32,767 through 32,767 (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 = -32,767 through 32,767 (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 65,535.
h	Height of outlining rectangle (see definition for x above). Valid range = 1 through 65,535.
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 the line width array index.
line_pat	Line pattern name.
[conditional]	Optional conditional for line pattern.
fill_pat	Fill pattern name.
[conditional]	Optional conditional for fill pattern.

RULES

1. The x and y coordinates can be relative.
2. The arc's origin is the starting point of the arc. If the arc is defined with relative coordinates, the origin of the arc is relative to the origin of the previous item drawn.
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 90o 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. The line pattern and fill pattern names may be followed by conditional expressions which will determine which line/fill pattern to display at runtime. The conditional expressions are optional.
5. The ARC command may be used in the background, foreground, or trigger sections.
6. For large, nearly horizontal/vertical arcs, it is possible that the circle/ellipse the arc lies on well exceeds the virtual canvas, even though the entire arc may be on the virtual canvas. Therefore, the coordinates defining the arc are not limited to the virtual canvas as for the other commands. A negative x or y coordinate for an arc indicates that the upper left corner of the outlining rectangle of the circle/ellipse the arc lies on is off the screen. It is also possible that some arcs are so large that the x, y, w, h required to draw them overflows the memory reserved for these fields in the object file. In this case, GBNT returns an error and does not allow arcs that large to be created.

EXAMPLES

Example 1

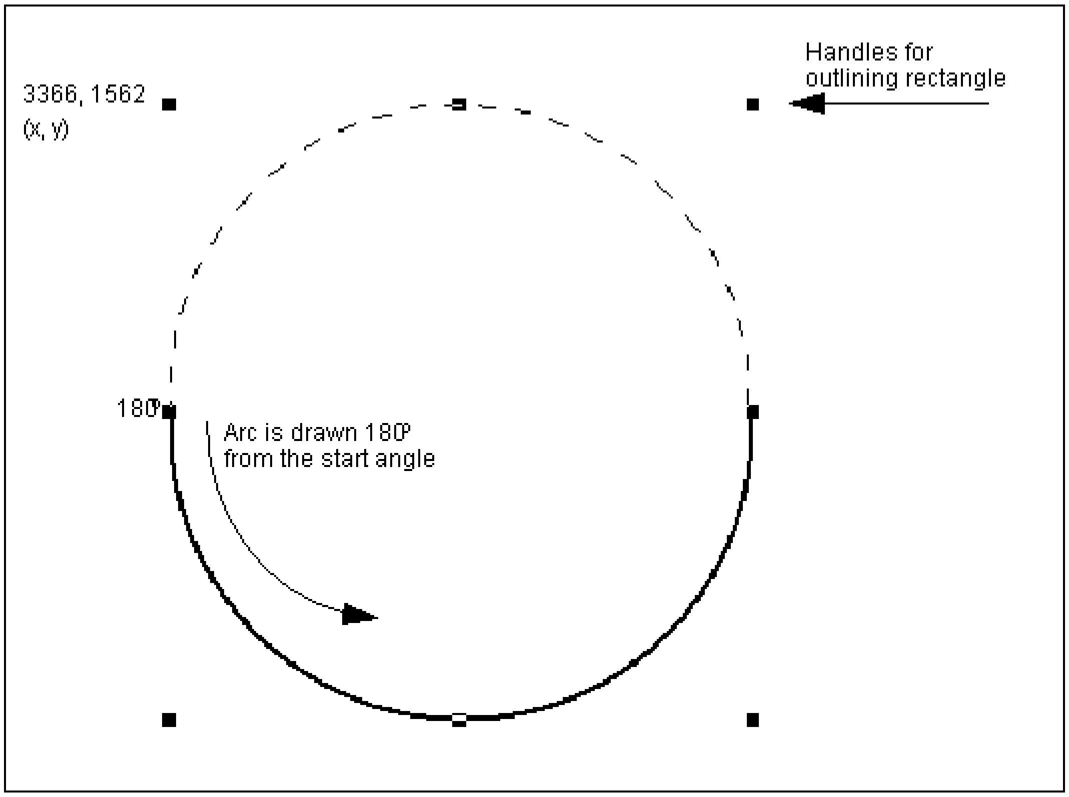
```
ARC 3366 1562 6262 10683 180 180 2 solid unfilled
```

In this example:

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

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 following figure:


Handles for outlining rectangle
3366, 1562
(x, y)
180°
Arc is drawn 180° from the start angle

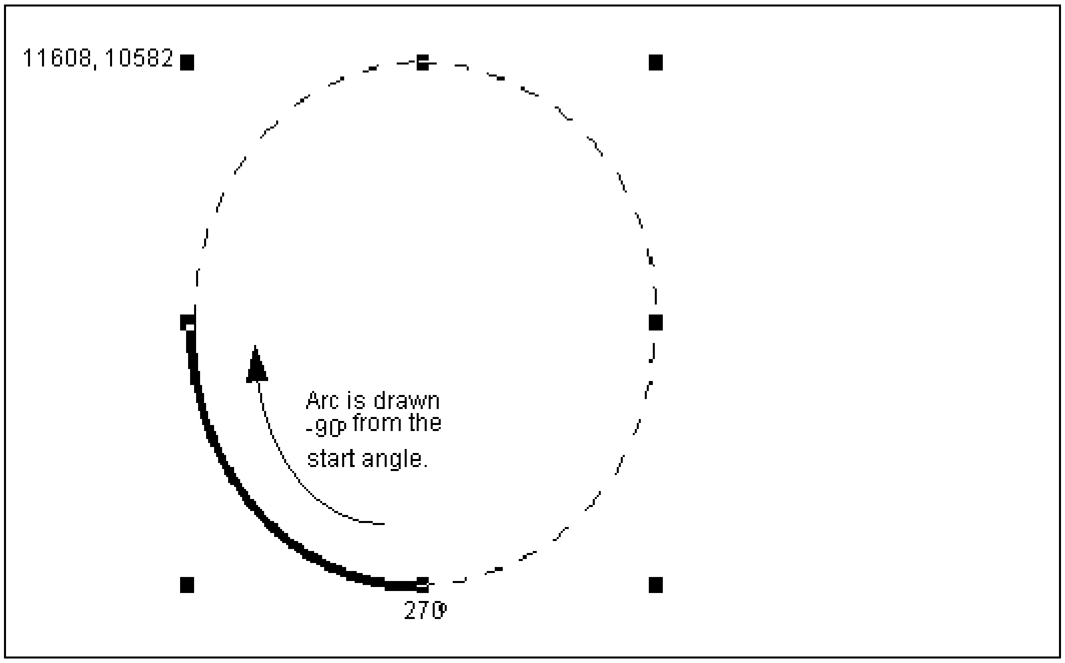
Example 2 (Conditional)

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

In this example:

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

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 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.

4.3 BACKGROUND

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 each time an exposure event is generated by the window manager. The window manager generates exposure events when the graphic window is resized and when other windows are on top of the graphic and then moved away. 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, or KEYBOARD command is encountered or when the end of the file is reached.

4.4 BAR

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_limit high_limit
```

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. Valid values are process point names, dummy point names, $pointers ($P, $G, $D, $W, $H, or $O) or OPC point names
rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth).
low_limit	Low limit used to scale value. Valid values are ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $Pointer identifier and $Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)

high_limit	High limit used to scale value. Valid values are ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $Pointer identifier and $Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)

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 at runtime. That is, if a UP type bar has a height of 100, and the scaled bar value is 25, then 1/4 of the bar will be filled with the current foreground color (FG) and the remaining 3/4 of the bar will be filled with the current erase color at run-time. The erase color should be the same as the background color over which the bar is displayed. GBNT 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
```

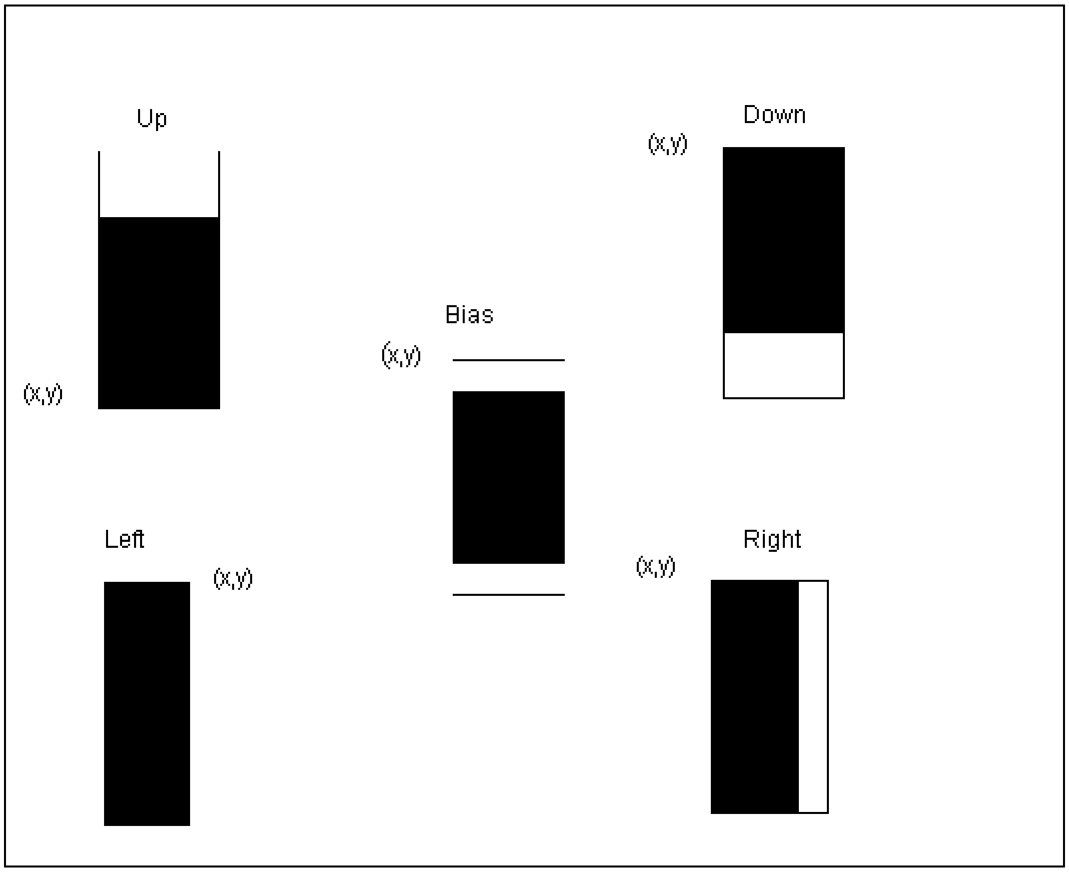
In this example:

5951	x coordinate
13305	y coordinate
2179	w (width)
10488	h (height)

UP	direction
A3890	pt_name
AV	rec_fld
20	low_limit
80	high_limit

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 GBNT interactive editor. (See *Ovation Graphics Builder User Guide*.) Notice that an arbitrary point value is assumed for bars in GBNT. 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.



4.5 CIRCLE

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

SYNTAX

```
CIRCLE cx cy r line_width line_pat [conditional] fill_pat
[conditional]
```

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; r<=cx; cx+r<=16383; cx-r>=0; r<=cy; cy+r<=16383; cy-r>=0)
line_width	Line width. Valid range = 1 through 16. The value corresponds to an array index.
line_pat	Line pattern name.
[conditional]	Optional conditional for line pattern.
fill_pat	Fill pattern name.
[conditional]	Optional conditional for fill pattern.

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 may be followed by conditional expressions which will determine which line/fill pattern to display at runtime. The conditional expressions are optional.
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. 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 you are 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 you are 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 you 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:

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

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.

Example 2 (Conditional)

```
CIRCLE 10582 2392 1132 2 solid unfilled (CASE) \A050\ LL 1 1 2
asterisks solid
```

In this example:

10582	cx
2392	cy
1132	r
2	line_width
solid	line_pat
unfilled	fill_pat
(CASE) \A050\ LL 1 1 2 asterisks solid	[conditional]

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 \A050\ LL as described below:

- If A050 LL < 2, the fill pattern will be unfilled (default parameter).
- If A050 LL >= 2 and if A050 LL < 3, the fill pattern will be asterisks.
- If A050 LL >= 3 and if A050 LL < 4, the fill pattern will be solid.
- If A050 LL >= 4, the fill pattern will be unfilled (default parameter).

4.6 COLOR

The COLOR command sets the foreground, background, erase, and OL drawing colors for subsequent display items in a graphic until the next COLOR command. Also sets the blink state for all subsequent display items in the current section if the optional blink attributes are specified.

Note: *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

```
COLOR FG color [conditional] BG color [conditional] ER color
[conditional] OL olcolor_index [conditional] BLINK FG blink_state
[conditional] BG blink_state [conditional]
```

where:

FG	Flag for foreground. Color (and optional conditional) that follows will be for the foreground.
color	Color name from the **colors.txt** file OR a $COLORn variable (1<=n<=100). It may be followed by an optional color conditional. The colors.txt file allows you to define custom colors in addition to the eight standard colors (black, blue, green, cyan, red, magenta, yellow, and white). (See *Ovation Graphics Builder User Guide*.)
[conditional]	Optional conditional for foreground color.
BG	Flag for background. Color (and optional conditional) that follows will be for the background.
color	Color name from the colors.txt file OR a $COLORn variable (1<=n<=100). It may be followed by an optional color conditional.
[conditional]	Optional conditional for background color.
ER	Flag for erase color. Color (and optional conditional) that follows will be used for the erase color.
color	Color name from the colors.txt file OR a $COLORn variable (1<=n<=100). It may be followed by an optional color conditional.
[conditional]	Optional conditional for erase color.
OL	Flag for OL color. Number from color index (and optional conditional) that follows will be used for the OL color.
olcolor_index	OL color index (0 through n-1) from the colors.txt file ***OR*** a $OL_COLORn variable (1<=n<=100) followed by an optional OL color index conditional. (See *Ovation Graphics Builder User Guide*.)

[conditional]	Optional conditional for OL color.
BLINK	Command for blink.
FG	Flag for foreground blink. State (and optional conditional) that follows will be for the foreground.
blink_state	Blink state (ON or OFF).
[conditional]	Optional conditional for foreground blink.
BG	Flag for background blink. State (and optional conditional) that follows will be for the background.
blink_state	Blink state (ON or OFF).
[conditional]	Optional conditional for background blink.

RULES

1. The foreground (FG) color is the color used to draw solid lines, text characters, bars, and so forth. The background (BG) color is only applicable when a line/fill pattern is used. Line/fill patterns are implemented as monochrome patterns, where the FG color replaces set bits and the BG color replaces clear bits in the pattern.
2. The erase (ER) color is used only by the runtime Ovation Graphics program when displaying bars, shapes, OL checkboxes, OL sliders, and OL cylinders. It is used to draw the unfilled portion of bars at runtime. It is used to erase the existing shape at runtime when a shape changes as a result of evaluating a shape conditional. It is used to erase the portion of a checkmark which extends outside the checkbox in an OL_CHECKBOX item when a box is checked/unchecked at runtime. It is used to erase the slider control when you drag the control on an OL_SLIDER item at runtime. It is used for some intermediate drawing of the OL cylinder. GBNT does not use ER color when displaying these items.
3. The OL color is used to draw any of the OL display items: OL_BUTTON, OL_RECTANGLE, OL_CHECKBOX, OL_CHOICE, OL_SLIDER, OL_EVENT_MENU, and OL_GAUGE. The OL color is the three-dimensional color.
4. $COLORn variables and $OL_COLORn variables are applicable to macro graphics only. Use these color variables to tagout colors which will be input parameters to the macro. $COLORn variables are used to input FG, BG, or ER colors. $OL_COLORn variables are used to input OL color indices. $COLORn and $OL_COLORn variables are intended to be used in macros only. If you use them in a non-macro graphic, undefined results will occur at runtime. GBNT allows you to configure the display color that GBNT uses to draw items built with each possible $COLORn and $OL_COLORn variable in the macro file while you are editing the macro file. (See *Ovation Graphics Builder User Guide*.)
5. Items blink only at runtime. GBNT does not blink items.
6. Blink is only practical in the foreground section of a graphic. However, using blink may adversely affect the diagram update rate as blink is implemented in software as opposed to hardware. Usage of blink should be minimized in a graphic.
7. The COLOR command is valid in the background, foreground, trigger, and keyboard sections. Blink is not valid in the keyboard section. Therefore, a COLOR command in the keyboard section of a graphic that specifies the BLINK attribute with either the FG or BG state set to "ON" will generate an error.

8. “FG”, “BG”, “ER”, “OL”, and “BLINK” tokens in the syntax are optional flags. You can specify all of these flags, a subset of these flags, or even none of these flags in the COLOR command. For any flag that is not specified in the COLOR command, the previous value from a preceding COLOR command in the same section of the graphic is used. Defaults are defined for each flag if it has not been previously specified in the graphic. The default BG, FG, and ER color is black, and the default OL color index is 0. The default FG and BG blink state is OFF.

Note: *BLINK is strictly an attribute of the COLOR command. Blink is not a graphics command.*

9. The FG, BG, ER, and OL colors may be followed by conditional expressions. You may specify conditionals to determine which foreground, background, erase, and OL color to use to display successive items in the graphic. Similarly, the FG and BG blink states may be followed by conditional expressions. You may specify conditionals to determine whether successive items should blink or not. Conditional expressions are optional.

EXAMPLES

Example 1

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

In this example:

red	foreground color
white	background color
yellow	erase color
Color associated with OL color 0 in the colors.txt file. (See *Ovation Graphics Builder User Guide*.)	ol color
off	blink foreground state
off	blink background state

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

Example 2 (Conditional)

```
COLOR FG cyan BG magenta (QUALITY) D200 green yellow red blue ER
black OL 1 BLINK FG OFF {(D007 LL = 101.5) ON (A403 AV >= A210 AV)
ON} BG OFF
```

In this example:

cyan	foreground color
magenta	background color for good_cond_value
(QUALITY)	required _keyword
D200	pt_name
green	background color for fair_cond_value
yellow	background color for poor_cond_value
red	background color for bad_cond_value
blue	background color for timed_out_cond_value
black	erase color
Color associated with OL color 1 in the colors.txt file. (See *Ovation Graphics Builder User Guide*.)	ol color
off	blink foreground state
{(D007 LL = 101.5) ON (A403 AV >= A210 AV) ON}	[conditional (FG blink)]
off	blink background state

The statement indicates that the foreground color is cyan, the erase color is black, and the OL 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 background color for subsequent items will be magenta.
- If D200 has fair quality, the background color for subsequent items will be green.
- If D200 has poor quality, the background color for subsequent items will be yellow.
- If D200 has bad quality, the background color for subsequent items will be red.
- If D200 has timed out quality, the background color for subsequent items will be blue.

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 A403 AV >= A210 AV, 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).

4.7 CURSOR

The CURSOR command places the drawing cursor at a specific location on the graphic. The CURSOR command is used in conjunction with display items built with relative coordinates to initialize the drawing position. Subsequent drawings on the graphic 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.
2. The x and y coordinates may be relative.

EXAMPLE

```
BACKGROUND
COLOR FG RED
CURSOR 0 1000
LOOP 14
   LINE [+1000] 1000 [+0] 15383 1 solid
ENDLOOP
CURSOR 1000 0
LOOP 14
   LINE 1000 [+1000] 15383 [+0] 1 solid
ENDLOOP
```

The example draws a red grid on the graphic. The grid lines are spaced every 1000 virtual pixels. There is a border of 1000 virtual pixels around the grid. The first CURSOR command initializes the drawing position for the vertical grid lines. The second CURSOR command initializes the drawing position for the horizontal grid lines. Note that the LINE commands utilize relative coordinates and the LOOP commands execute 14 times to draw 14 vertical or horizontal lines making up the grid.

4.8 DATE

The DATE command displays the current date on a graphic in a specified format.

SYNTAX

```
DATE x y format font_type font_size
```

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 text outlining rectangle (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 date display (used if BITMAP or BITMAP_OVER is selected for font_type). ***OR*** The y coordinate of the upper left corner of the text outlining rectangle (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 digits of the year are implied to be 20). ▪ mmm = month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC)

font_type	Font types. The choices are: ▪ BITMAP - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ BITMAP_OVER - non-scalable; character and background drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**. ▪ VECTOR_OVER - scalable; character and background drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. **bitmap_font_number** - Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through 8. **charw** - Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.

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 and bitmap_over).
   - Upper left corner of the first character of the text outlining rectangle (for vector and vector_over).
3. The DATE command may be used in the background, foreground, and trigger sections of a diagram. However, it should be used in the foreground so that it updates if the date changes while the diagram is running.

EXAMPLES

Example 1

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

In this example:

4128	x
3000	y
0	format
VECTOR	font_type

161 (charw)	font_size
336 (charh)	
1 (lw)	

This date uses vector text and will be in the format mm/dd/yy (for example, 11/25/00) 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:

2752	x
3397	y
2	format
BITMAP	font_type
3 (bitmap_font_n)	font_size

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 size will be the 3rd point size specified in the fonts.txt file.

4.9 DEF_FKEY_GROUP

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.
keyn	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.
progn	Application program number (may be specified as an integer OR by a $Pn $Im pointer/offset pair, where m is a multiple of 4).
diagn	Diagram number to display for associated application program (may be specified as an integer OR by a $Pn $Im pointer/offset pair, where m is a multiple of 4). Valid range is 0 - 65,535.
mn	Number of arguments to pass to progn (varies per program)
argsn	List of arguments to pass to progn. Valid argument types are: ▪ string (130 characters 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. This command must be created/edited via the source editor.
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\ LL
    \A001Z002\ LL 66 0 4 1 0 1 “Enter Alarm Status”
    6 0 5 \A001Z001\ LL \A001Z002\ LL 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.

4.10 DEF_MACRO_PARAMS

The DEF_MACRO_PARAMS command allows you to create/edit the macro parameter description labels as well as macro notes and comments.

SYNTAX

```
DEF_MACRO_PARAMS   #$D          desc$D2 ... desc$Dn

                   #$T          desc$T1 desc$T2 ... desc$Tn

                   #$T(bg)      desc$T1(bg) desc$T2(bg) ... desc$Tn(bg

                   #$Set        desc$Set1 desc$Set2 ... desc$Setn

                   #$Const      desc$Const1 desc$Const2 ... desc$Constn

                   #$Status     desc$Status1 desc$Status2 ... desc$Statusn

                   #$Color      desc$Color1 desc$Color2 ... desc$Colorn

                   #$OLcolor    desc$OLcolor1 desc$OLcolor2 ... desc$OLcolorn
```

where:

#$D	Number of process point parameters (0 through 99) you are defining labels for in this command.
desc$D1 - desc$Dn	If #$D > 0, this is the list of quoted label strings for $D1 - $Dn.
#$T	Number of text string parameters (0 through 50) in the foreground, trigger, and/or keyboard sections of the macro you are defining labels for in this command.
desc$T1 - desc$Tn	If #$T > 0, this is the list of quoted label strings for $T1 - $Tn in the foreground, trigger, and/or keyboard sections of the macro.
#$T(bg)	Number of text string parameters (0 through 50) in the background section of the macro you are defining labels for in this command.
desc$T1(bg) - desc$Tn(bg)	If #$T(bg) > 0, this is the list of quoted label strings for $T1 - $Tn in the background section of the macro.
#$Set	Number of set number parameters (0 through 256) you are defining labels for in this command.
desc$Set1 - desc$Setn	If #$Set > 0, this is the list of quoted label strings for $Set1 - $Setn.
#$Const	Number of integer/real constant parameters (0 through 256) you are defining labels for in this command.
desc$Const1 - desc$Constn	If #$Const > 0, this is the list of quoted label strings for $Const1 - $Constn.

#$Status	Number of status word parameters (0 through 256) you are defining labels for in this command.
desc$Status1 - desc$Statusn	If #$Status > 0, this is the list of quoted label strings for $Status1 - $Statusn.
#$Color	Number of color parameters (0 through 100) you are defining labels for in this command.
desc$Color1 - desc$Colorn	If #$Color > 0, this is the list of quoted label strings for $Color1 - $Colorn.
#$OLcolor	Number of OL color parameters (0 through 100) you are defining labels for in this command.
desc$OLcolor- desc$OLcolorn	If #$OLcolor > 0, this is the list of quoted label strings for $OLcolor1 - $OLcolorn.

RULES

1. The DEF_MACRO_PARAMS command is only valid in the DIAGRAM section of a MACRO file. A macro file is named macroN.src, where N = 1 through 65,535. There can be only one DEF_MACRO_PARAMS command per graphic.
2. The parameter labels defined by this command are displayed on the GBNT:Macro dialog when the macro this command resides in is selected on the dialog. This is the only purpose of this command.
3. This command is optional. If this command is omitted in a macro file, and that macro is selected on the GBNT:Macro dialog, the default parameter labels are used. The default labels are:
   - $D1 - $Dn,
   - $T1 - $Tn,
   - $T1(bg) - $Tn(bg)
   - $Set1 - $Setn
   - $Const1 - $Constn
   - $Status1 - $Statusn
   - $Color1 - $Colorn
   - $OL_color1 - $OL_Colorn
4. All arguments in this syntax above beginning with the “desc” are the ASCII description strings to use for the associated macro parameter. These strings must be enclosed in single/double quotes and can be a maximum of 80 characters.
5. The number of parameter within the macro that you want to specify descriptions/labels for is optional. For example, a macro may use 5 $D points but you may only want to specify descriptions for 3 of them. GBNT will default the labels for any parameters that you do not specify descriptions for according to Rule #3 above.

   You can pass blank descriptions/labels to this command. If you only want a description for $D3, then you must set #$D=3 and pass blank descriptions for $D1 and $D2. You cannot skip indices when defining the labels for any given type of parameter to this command. That is, you cannot specify a label for $D3 without specifying labels for $D1 and $D2, but you can specify blank strings for $D1 and $D2.

6. The GBNT:Macro Notes and Parameter Descriptions dialog is used to interactively create/edit this command in a macro graphic. Use of this dialog is encouraged instead of editing the source syntax because the dialog will automatically scan the macro file and build a table of all of the existing macro parameters for you. In this way, you can see what you have to define labels for. Then you can decide to ignore certain parameters is necessary. This dialog will automatically build/edit the source syntax in the macro graphic.
7. There is the optional capability to be warned of undefined/blank macro parameter descriptions when you save a macro file (and/or when you press the OK button on the GBNT:Macro Notes and Parameter Descriptions dialog). You will not be warned when compiling this command in the source editor or when loading the macro file. These warnings can be turned on/off from the Options menu on the main menu bar. (See *Ovation Graphics Builder User Guide*.) You can also bring up the GBNT:Macro Notes and Parameters Descriptions dialog from the Options menu. Both of these menu items are inactive if not editing a macro graphic. The missing /blank parameter description warning setting is saved as part of the Save Default function.

EXAMPLE

```
DEF_MACRO_PARAMS 3    "start/stop algorithm pt"

                      "auto/manual algorithm pt"

                      "local/remote digital"

                 0

                 1    "MA station title"

                 0

                 1    "control trigger"

                 0

                 6    "good quality color"

                      "fair quality color"

                      "bad quality color"

                      "poor quality color"

                      "timed-at color"

                      "travel mode color"

                 0
```

where:

#$D=3	label for $D1 = "start/stop algorithm pt"
	label for $D2 = "auto/manual algorithm pt"
	label for $D3 = "local/remote algorithm pt"
#$T=0	No $T listed.
#$T(bg)+1	label for $T1 in background = "MA Station title"
#$Set = 0	No $Set listed.
#$Const = 1	label for $Const1 = "control trigger"
#$Status = 0	No $Status listed.
#$Color = 6	label for $Color1 = "good quality color"
	label for $Color2 = "fair quality color"
	label for $Color3 = "bad quality color"
	label for $Color4 = "poor quality color"
	label for $Color5 = "timed-at color"
	label for $Color6 = "travel made color"
#$OLColor = 0	No $OLColor listed.

The example above shows how descriptions can be given for macro parameters.

4.11 DEF_QUAL

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.

SYNTAX

DEF_QUAL	good_fg_color	good_bg_color	good_blink	good_char
	fair_fg_color	fair_bg_color	fair_blink	fair_char
	poor_fg_color	poor_bg_color	poor_blink	poor_char
	bad_fg_color	bad_bg_color	bad_blink	bad_char
	timedout_fg_ color	timedout_bg_ color	timedout_ blink	timedout_ char

where:

good_fg_color	Foreground color name for displaying good quality character.
good_bg_color	Background color name for displaying good quality character.
good_blink	Blink state for good quality. The choices are: ▪ ON - good quality character blinks at runtime. ▪ OFF - good quality character does not blink at runtime.
good_char	Single ASCII character used to display good quality (must be enclosed in quotes).
fair_fg_color	Foreground color name for displaying fair quality character.
fair_bg_color	Background color name for displaying fair quality character.
fair_blink	Blink state for fair quality. The choices are: ▪ ON - fair quality character blinks at runtime. ▪ OFF - fair quality character does not blink at runtime.
fair_char	Single ASCII character used to display fair quality (must be enclosed in quotes).
. . .	. . .

timedout_char	Single ASCII character used to display timed out quality (must be enclosed in quotes).

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. There are five qualities to be specified by this command: good, fair, poor, bad, and timed-out. All five must be specified by this command. You cannot just specify the values for good quality, for example.
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 at runtime. 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.
6. Default values are defined for quality if you do not specify this command in a graphic. The default quality values are:

Quality	FG Color	BG Color	Blink	Character
good	green	black	OFF	'G'
fair	cyan	black	OFF	'F'
poor	yellow	black	OFF	'P'
bad	red	black	OFF	'B'
timed out	red	black	OFF	'T'

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 9620 1794 6 2 RIGHT0 ON HORZ VECTOR 321 734 1 \A300\ AV 0
100
```

In this example:

green	good_fg_color
black	good_bg_color
OFF	good_blink
“G”	good_char
cyan	fair_fg_color
black	fair_bg_color

OFF	fair_blink
“F”	fair_char
yellow	poor_fg_color
black	poor_bg_color
OFF	poor_blink
“P”	poor_char
magenta	bad_fg_color
black	bad_bg_color
OFF	bad_blink
“B”	bad_char
red	timedout_fg_color
black	timedout_bg_color
OFF	timedout_blink
“T”	timedout_char

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. The color and blink state for point \A300\ will vary depending on its quality as described below:

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.

4.12 DIAG_DISP

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 ▪ CONTROL
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 *Ovation Operator Station User Guide*.) Valid range = 0 through 5,000. Entering 0 does not allow the current group to be changed.
[POSITION x y]	Optional series of 3 parameters used to specify diagram size. **POSITION** - ASCII string indicating that x, y will follow **x** - x pixel position of upper left corner of diagram on CRT screen (optional; only valid after the ASCII string, POSITION). Valid range = 0 through 3200 - 1 (entered in screen coordinates) **y** - y pixel position of upper left corner of diagram on CRT screen (optional; only valid after the ASCII string, POSITION). Valid range = 0 through 1200 **-** 1 (entered in screen coordinates)
[SIZE w h]	Optional series of 3 parameters used to specify diagram size. **SIZE** - ASCII string indicating that w, h will follow **w -** pixel width of diagram (optional; only valid after the ASCII string, SIZE). Valid range = 100 through 3200 (entered in screen coordinates). **h** - pixel height of diagram (optional; only valid after the ASCII string, SIZE). Valid range = 100 through 1200 (entered in screen coordinates).

num_sids	Number of $W points in 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.

RULES

1. The DIAG_DISP command replaces the current diagram on the screen at run-time with a new diagram. If the location and dimensions are not specified in the syntax, the specified diagram will be displayed on top of the current diagram (using the location and dimensions of the current diagram). If the location and/or dimensions are specified in the syntax, the diagram will be displayed at that x, y location side to that w, h.
2. 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.*

3. The diagram's position (x, y) and size (w, h) are optional parameters. If these parameters are not used, x, y, w, of the previously displayed diagram will be used. The reserved words POSITION and SIZE must precede the x, y and w, h arguments in the source to indicate that the optional parameters are to follow.
4. GBNT does not execute this command.
5. The point names passed to this routine will be used to set the $W points for the system. For example, if 5 points are passed, you set $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.
6. The maximum screen size of a PC is 1600 x 1200 pixels. To support dual CRT's, the maximum width of a graphic is 3200 (1600 * 2) and maximum screen x coordinate is 3199 (3200 - 1).

EXAMPLE

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

In this example:

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

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.

4.13 DIAGRAM

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). It contains general information about the graphic, such as size, position, update rate, background, and so forth.

A DIAGRAM command automatically appears in the source editor when GBNT initially starts. You can use the given settings or change any of the parameters.

SYNTAX

```
DIAGRAM diag_type 0 x y w h background zoom_flag revision_num 0 0
16384 16384 update_rate positioning sizing [subscreen#] [diag_title]
```

where:

diag_type	Diagram type. Choose one of the following: ▪ MAIN ▪ SUBWIN ▪ WINDOW ▪ CONTROL
0	Not needed in the Ovation system. Enter 0 as a placeholder.
x	x coordinate that corresponds to the upper left corner of the diagram on the Operator Station. Only used if FIXED_POSITION is selected for the positioning parameter (see positioning parameter below). Valid range = 0 through 3200 - 1 (entered in screen coordinates).
y	y coordinate that corresponds to the upper left corner of diagram on the Operator Station. Only used if FIXED_POSITION is selected for the positioning parameter (see positioning parameter below). Valid range = 0 through 1200 - 1 (entered in screen coordinates)
w	Width of the diagram to be displayed at the Operator Station. Only used if FIXED_SIZE is selected for the sizing parameter (see sizing parameter below). Valid range = 100 through 3200 (entered in screen coordinates)
h	Height of the diagram to be displayed at the Operator Station. Only used if FIXED_SIZE is selected for the sizing parameter (see sizing parameter below). Valid range = 100 through 1200 (entered in screen coordinates)

background	Type of background used for the diagram. Choose one of the following: ▪ Standard background color - You can choose one of the eight standard colors (black, blue, green, cyan, red, magenta, yellow, and white) or one of the custom colors. (See *Ovation Graphics Builder User Guide*.) ▪ Bitmap file name - Bitmap background filename less the ".bmp" extension. When running GBNT in the offline mode, these files reside in **[InstallationPath]/mmi/graphics/cstfiles** (where "InstallationPath" is a user-define directory). **Note:** *When running GBNT from the Developer Studio, bitmap files must be imported using the Import Graphics program.* (See *Ovation Graphics Builder User Guide*.)
zoom_flag	Zoom flag. Choose one of the following: ▪ ZOOMABLE - the diagram can be zoomed at the Operator Station. ▪ NON_ZOOMABLE - the diagram cannot be zoomed at the Operator Station.
revision_num	Revision level of the diagram (user-defined). Valid range = 0 through 2,147,483,647.
0	x_extents is no longer used. However, this 0 must be present in the source syntax as a placeholder for backwards compatibility.
0	y_extents is no longer used. However, this 0 must be present in the source syntax as a placeholder for backwards compatibility.
16384	width_extents is no longer used. However, this 16384 must be present in the source syntax as a placeholder for backwards compatibility.
16384	height_extents is no longer used. However, this 16384 must be present in the source syntax as a placeholder for backwards compatibility.
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 of the diagram on the Operator Station at runtime. 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 of the diagram on the Operator Station at runtime. 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#]	Optional subscreen window number which can be specified when the diag_type parameter = MAIN. This parameter specifies the subscreen diagram, which will display along with the main screen diagram when the main screen displays at runtime. Valid range is 0 through 65,535.
[diagram_title]	Optional diagram name/title/description. Specifies an ASCII string enclosed in single or double quotes that names the diagram. The name must be 60 characters or less.

RULES

1. The DIAGRAM command statement is required. It must be the first command in the graphic. It can only be used once in the graphic.
2. A DIAGRAM command is created when GBNT starts. It can be edited via the source editor or the GBNT:Diagram Configuration dialog. However, this command can never be deleted. Only one DIAGRAM command may exist in a graphic.
3. The DIAGRAM command defines a section of the graphic which will be executed once and only once. Typically, pointers are defined and initialized in this section using the POINTER and PTR_VALUE commands.
4. The diagram section ends when a BACKGROUND, FOREGROUND, TRIGGER, or KEYBOARD command is encountered or when the end of the file is reached.
5. The diagram title in the syntax above is optional.
6. Diagram name displays in the Operator Station Graphic Favorites List, in the GBNT window title bar, and in the GBNT:Macro dialog list of available macros. If no title is defined for any graphic, then the title is blank for that graphic.
7. Neither the x or y parameters are used at runtime if DEFAULT_POSITION is specified for the positioning parameter. Likewise, neither the w or h parameters are used at runtime to size the graphic if DEFAULT_SIZE is specified for the sizing parameter.
8. All graphics are zoomable in GBNT. The zoom_flag parameter only applies to runtime.

WARNING: Caution should be used when selecting a diagram update rate of less than 1 second. Doing so may adversely affect the overall performance time of the graphic.

EXAMPLE

```
DIAGRAM MAIN 1 395 0 631 588 WhiteSmoke ZOOMABLE 1 0 0 16384 16384 3
DEFAULT_POSITION DEFAULT_SIZE 2500 “sample diagram with window 2500”
```

In this example:

MAIN	diag_type
placeholder	0
395	x
0	y
631	w
588	h

WhiteSmoke	background
ZOOMABLE	zoom_flag
1	revision_num
0 (placeholder)	0
0 (placeholder)	0
16384 (placeholder)	16384
16384 (placeholder)	16384
3	update_rate
DEFAULT_POSITION	positioning
DEFAULT_SIZE	sizing
“sample diagram with window 2500”	[diagram name/title]

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

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 at the Operator Station.

The name of the diagram is “sample diagram with window 2500.”

4.14 DOT

The DOT command displays a dot in one of three sizes 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

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:

688	x
1200	y
LARGE	size

This statement defines a large dot at 688, 1200.

4.15 DYNAMIC_LINE

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 NOSCALE
update_rate n x1 y1 x2 y2 ... xn yn line_width line_pat [conditional]
```

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 x_high y_low y_high	Low limit used to scale x coordinates. High limit used to scale x coordinates. Low limit used to scale y coordinates. High limit used to scale y coordinates. ▪ Valid values for limits are: ▪ Integers/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $pointer identifier and $offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
line_style	Specifies the line join style for the vertices. The choices are: ▪ NOT_FITTED - draws a straight line between vertices (same as the LINE command). ▪ FITTED - draws a curve that best fits vertices.
NOSCALE	Placeholder. This parameter is no longer used. Scales are never displayed.
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; x and y may be specified as same as limits. See valid values for x_low, x_high, y_low, y_high for xi, yi.)
line_width	Line width. Valid range = 1 through 16. The value corresponds to the line width array index.
line_pat	Line pattern name
[conditional]	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. However, it typically goes in the foreground section so its display updates.
4. The line pattern may be followed by a conditional expression which will determine which line pattern to display at runtime. Conditional expressions are optional.
5. Scale values are never automatically displayed (no matter what the scale parameter is set to). If you want to display the scale values, they must be displayed independently using the TEXT or PROCESS_PT command.
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 if it is less than the overall graphic 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.

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:

3604	x
6318	y
3770	w
2412	h
0	x_low

100	x_high
25	y_low
50	y_high
NOT_FITTED	line_style
NOSCALE	scale
2	update_rate
2	n
\A100\ AV, 50	x1,y1
\A200\ AV, \A300\ AV	x2,y2
1	line_width
solid	line_pat

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:

3604	x
6318	y
3770	w
2412	h
\A100\ LL	x_low
\A100\ HL	x_high
\A200\ LL	y_low
\A200\ HL	y_high
FITTED	line_style
SCALE	scale
5	update_rate
4	n
12.45, \A200\ AV	x1,y1
\A100\ HL, 0	x2,y2

20, \A200\ AV	x3,y3
100, \A200\ HL	x4,y4
3	line_width
dashed	line_pat
(testai01 av > 6.5) dotted	[conditional]

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 NOT displayed (scales are never displayed no matter what this setting says). The curve is drawn with line width #3 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.

4.16 DYNAMIC_POLYGON

The DYNAMIC_POLYGON command displays a polygon 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. The polygon is automatically closed (the first vertex does not have to equal the last vertex).

SYNTAX

```
DYNAMIC_POLYGON x y w h x_low x_high y_low y_high update_rate n x1 y1
x2 y2 x3 y3 ... xn yn line_width line_pat [conditional] fill_pat
[conditional]
```

where:

x	x coordinate of lower left corner of boundary area of dynamic polygon. Valid range = 0 through 16,383.
y	y coordinate of lower left corner of boundary area of dynamic polygon. Valid range = 0 through 16,383.
w	Width of boundary area of dynamic polygon. Valid range= 1 through 16,383
h	Height of boundary area of dynamic polygon. Valid range = 1 through 16,383
x_low x_high y_low y_high	Low limit used to scale x coordinates. High limit used to scale x coordinates. Low limit used to scale y coordinates. High limit used to scale y coordinates. Valid values for limits are: ▪ Integers/real numbers (for example, 123, -100, 50.5) ▪ Point/pointer name and record field pair (for example, \A100\ LL, $D1 AV, $G2 HL) ▪ $pointer identifier and $offset identifier (for example, $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
update_rate	How often the dynamic polygon should be redisplayed (in seconds) Valid range is 1 through 32767.
n	Number of x,y coordinate pairs (vertices) Valid range is 3 through 50.

xi, yi	Coordinate pairs making up the polygon. A minimum of 3 coordinate pairs must be defined. (x and y may be fixed or dynamic; x and y may be specified same as limits. See valid values for x_low, x_high, y_low, y_high for xi, yi.)
line_width	Line width. Valid range = 1 through 16. The value corresponds to the line width array index.
line_pat	Line pattern name
[conditional]	Optional line pattern conditional expression.
fill_pat	Fill pattern name
[conditional]	Optional fill pattern conditional expression.

RULES

1. The x and y coordinates of the rectangular boundary area of the dynamic polygon may be relative; the individual vertex points making up the polygon may not be relative.
2. The origin is the lower left corner of the boundary rectangle of the dynamic polygon (specified by the **x** and **y** parameters in the source syntax).
3. The DYNAMIC_POLYGON command may be used in the background, foreground, or trigger sections of the diagram. However, the dynamic polygon will not update unless it is in the FOREGROUND section, despite the update_rate parameter.
4. A minimum of three x, y coordinates must be defined for the dynamic polygon.
5. The line pattern and fill pattern names may be followed by conditional expressions which will determine which line/fill pattern to display at runtime. The conditional expressions are optional.
6. To display scale values at runtime, you must independently add them to the graphic using the TEXT or PROCESS_PT command.
7. The update_rate parameter overrides the overall diagram update rate. The update_rate for the dynamic polygon **cannot be less** than the overall diagram update rate (although this is not reported as an error, the dynamic polygon update rate will default to the diagram update rate if it is less than the overall diagram update rate).
8. It is possible to calculate vertex points to be used in the dynamic polygon 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.

EXAMPLES

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

In this example:

3604	x
6318	y
3770	w
2412	h
\A100\ LL	x_low
\A100\ HL	x_high
\A200\ LL	y_low
\A200\ HL	y_high
5	update_rate
4	n
12.45, \A200\ AV	x1,y1
\A100\ HL, 0	x2,y2
20, \A200\ AV	x3,y3
100, \A200\ HL	x4,y4
3	line_width
dashed	line_pat
(testai01 av > 6.5) dotted	[conditional]
unfilled	fill_pat

This statement displays a dynamic polygon with four vertices. The curve is drawn with line width #3 (which corresponds to a pixel line width = 4) 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 polygon 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. The dynamic polygon will not be filled.

4.17 EF_STATE

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

Rule

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:

9	entry_fld
ON	state

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).

4.18 ELLIPSE

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 [conditional] fill_pat
[conditional]
```

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 the line width array index.
line_pat	Line pattern name.
[conditional]	Optional conditional for line pattern.
fill_pat	Fill pattern name.
[conditional]	Optional conditional for fill pattern.

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 (you 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

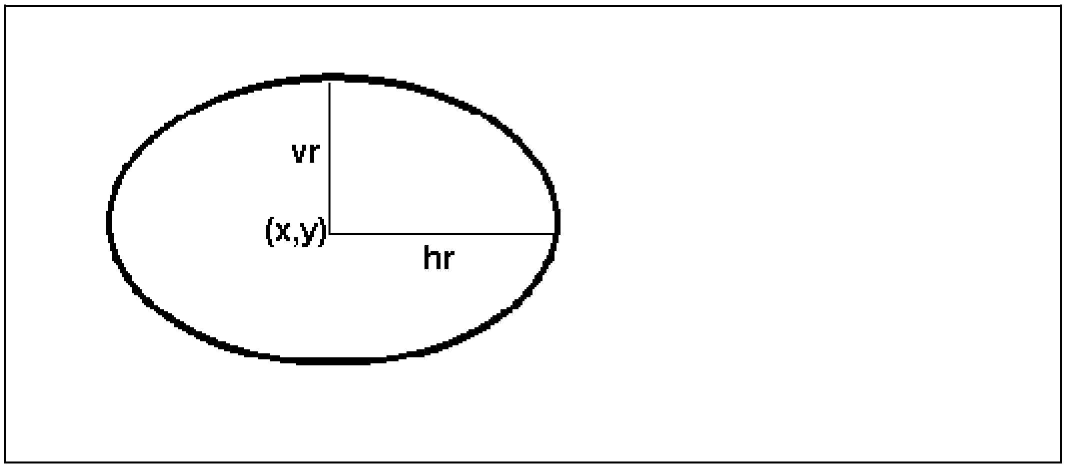
```
ELLIPSE 9089 6245 3563 4524 3 solid unfilled
```

In this example:

9089	cx
6245	cy
3563	hr

4524	vr
3	line_width
solid	line_pat
unfilled	fill_pat

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 (which corresponds to a pixel line width of 4 in the line width table), and the line pattern is solid. The ellipse is not filled.



Example 2 (Conditional)

```
ELLIPSE 1093 6500 292 572 4 dashed {(A100 AV > 100) dotted (LOWALARM
= A200 1W) solid} unfilled
```

In this example:

1093	cx
6500	cy
292	hr
572	vr
4	line_width
dashed	line_pat
{(A100 AV > 100) dotted (LOWALARM = A200 1W) solid}	[conditional]
unfilled	fill_pat

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 (which corresponds to a pixel line width = 6 in the line width table).

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 1W, the line pattern will be solid. If neither condition is met, the line pattern will be dashed (default). The fill pattern is unfilled.

4.19 ENTRY_FLD

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 font_size
```

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 text outlining rectangle (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 text outlining rectangle (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 Station). 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 - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ BITMAP_OVER - non-scalable; character and background drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**. ▪ VECTOR_OVER - scalable; character and background drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. **bitmap_font_number** - Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through 8. **charw** - Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.

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 Station 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 1
```

In this example:

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

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 1.

Example 2

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

In this example:

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

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.

4.20 FKEY_STATE

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

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:
This statement activates function key 1.

1	key
ON	state

4.21 FONT

The FONT command is used to specify a font name and a font style for all successive vector text-based commands in the graphic. The specified font name and style will be used for all vector text items until the next FONT command is encountered or until a BACKGROUND, FOREGROUND, TRIGGER, or KEYBOARD command is encountered.

The text-based commands are: TEXT, MULTI-TEXT, TIME, DATE, ENTRY_FLD, OL_EVENT_MENU, OL_BUTTON (text label only), OL_CHOICE (text label only), GTEXT and PROCESS_PT. The font name can be any font that is currently installed on the computer where GBNT is running.

SYNTAX

```
FONT “name” style
```

where**:**

name	Font name enclosed in single or double quotes. The font name must be one of the names listed in the Font pull-down menu on the GBNT: Font Attributes window. (See *Ovation Graphics Builder User Guide*.) The name is limited to 30 characters. The name must be typed in the same case as it appears on the window mentioned above. The name is case sensitive.
style	Style to display named font. The choices are: ▪ Regular ▪ Italic ▪ Bold ▪ Bold_italic **Note:** *Font style is NOT case sensitive.*

RULE

1. The FONT command may be used in the BACKGROUND, FOREGROUND, TRIGGER, and KEYBOARD sections of a diagram.
2. There is no display associated with this command. This command does not affect bitmap or bitmap_over type text. This command only affects vector and vector-over type text.
3. The FONT command sets the font name and style for all vector text display items until another FONT command is encountered or until a new diagram section is defined.
4. The FONT command does not set font size. The font size is specified in the vector text commands (TEXT, MULTI-TEXT, TIME, DATE, ENTRY_FLD, OL_EVENT_MENU, OL_BUTTON (text label only), OL_CHOICE (text label only), GTEXT and PROCESS_PT). Font size is specified on a per command basis.

Note: *When GBNT attempts to create/get a font of a particular size, it is only a request. The font mapper attempts to find the font in the requested size, and then gives the closest match if the exact font is not found. GBNT has no control over this. Sometimes the match is exact; sometimes it is not even close. The font mapper searches by height first and then by width. If the height exists, the font mapper returns that font whether it matches the width or not.*

Text sizes may change on a zoom/resize operation. This means that if you interactively resize text, the text drawn after you release the mouse pointer may not represent what the outlining rectangle handles lead you to believe the size would be while dragging for resize. GBNT may have requested a certain size font, but received a different size from the font mapper. This also means that if you draw a box around text and then resize the diagram, zoom the diagram, or group the box and text and resize the group, the text may not always be within the box after the resize. During the resize, GBNT requests a certain font size to keep the text in the box but may receive another size from the font mapper.

Note that the information above does NOT apply to Ovation Vector Font since it is an Ovation font - not a Windows font. It will size exactly per the requested size.

EXAMPLE

```
FONT “Garamond” bold
```

In this example:
This statement causes all vector text type commands which come after it to be drawn in the Garamond bold font.

“Garamond”	“name”
bold	style

4.22 FORCE_UPDATE

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 have not 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. This command must be added/edited via a source editor.
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:

TRUE	state

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

4.23 FOREGROUND

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.

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, or TRIGGER command is encountered.

4.24 FUNC_KEY

The FUNC_KEY command is used to run one or more application programs from a function key (F1 - F10) on the standard Ovation Operator Station keyboard and/or from a programmable function key (P1 - P10) on the membrane keyboard. (See *Ovation Operator Station User Guide*.)

Pressing P1 on the membrane keyboard will run the programs associated with key #1 in this command, as will pressing F1 on the standard keyboard. Pressing P2/F2 will run the programs associated with key #2, and so forth. The pkeys and the function keys are exactly the same functionality. The only difference between the function keys and the pkeys is the keyboard where they exist. The syntax for specifying the list of application programs to be associated with a given function key/pkey in this command is similar to a poke type #7 command.

The association of the application program(s) to the given function key/pkey is local to the graphic in which the FUNC_KEY command exists. When a function key/pkey 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 Operator Station 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.

In addition, if a function key was pressed (as opposed to a pkey) when there is no FUNC_KEY command defined for the given key, then the standard Windows function will prevail where applicable. This does not apply to pkeys.

The currently active function keys defined in a graphic may be dynamically changed by using the DEF_FKEY_GROUP command and the LOAD_FKEY_GROUP commands.

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 - 10.
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.
prog*n*	Application program number. This may be specified as an integer or by a $Pn $Im pointer/offset pair, where m is a multiple of 4.
diag*n*	Diagram number to display for associated application program. This may be specified as an integer or by a $Pn $Im pointer/offset pair, where m is a multiple of 4. Valid range is 0 - 65,535.
m*n*	Number of arguments to pass to prog*n* (varies per program)

args*n*	List of arguments to pass to prog*n*. Valid argument types are: ▪ text strings (130 characters or less) ▪ integers ▪ reals ▪ point name and record field (example: A100 AV) ▪ pointer and offset (example: $P1 $R0) ▪ set variable (example: SET1) ▪ macro input parameters (example: “$Tn”, $CONSTn, $SETn, $Dn, $STATUSn)

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. There are 10 pkeys (P1 - P10) on the membrane keyboard of which you can program any/all of using this command. There are 12 available function keys (F1 - F12) on the standard keyboard, of which only the first 10 (F1 - F10) can be programmed using this command. Programming function keys F11 and F12 is not supported in the Ovation system.
4. There can be only one FUNC_KEY definition for each of the possible 10 function/pkey in any one graphic. Thus, a graphic can have a maximum of 10 FUNC_KEY commands in it. An error will display if you try to add duplicate function/pkeys. Note that this rule also applies with respect to inclusion in macros. A macro will not load if it contains a FUNC_KEY command for a function key that is already defined in the current graphic.

EXAMPLE

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

In this example:

1	key
ON	state
3	n
117	prog1
7001	diag1
7	m1
5002,60,22,0,2,\A001Z001\ LL, \A001Z002\ LL	args1
66	prog2
0	diag2

4	m2
1,0,1,"Enter Alarm Status"	args2
6	prog3
0	diag3
5	m3
\A001Z001\ LL,\A001Z002\ LL, 1,1, 2	args3

This statement associates three application programs with function key/pkey number 1. When the F1 key is pressed on the standard keyboard (or P1 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.

4.25 GCODE

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

The GCODE command is reserved for Emerson use only.

4.26 GTEXT

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 *Ovation Developer Studio User Guide*.)

SYNTAX

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

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 outlining rectangle of the string (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 outlining rectangle of the string (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 types. The choices are: ▪ BITMAP - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ BITMAP_OVER - non-scalable; character and background drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**. ▪ VECTOR_OVER - scalable; character and background drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. **bitmap_font_number** - Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through. 8. **charw** - Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.

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 outlining rectangle 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:

2064	x
6600	y
1	string_number
20	number_of_chars
HORZ (horizontal)	orientation
VECTOR	font_type

161	charw
336	charh
1	lw

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.

4.27 IF_CHANGED/ENDIF

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 you enter 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). This must be a real number (for example, 0.3, 3.0, 3.6, and so forth). Enter 0.0 if none of the operands to this command are analog points. 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 and record field (for example, A100 AV, A200 LL) ▪ $pointer and $ offset (or record field if applicable) ($P1 $I0, $W1 1W) ▪ Set variable (SET1 through SET255) ▪ Status word

RULES

1. A status word cannot be the first operand. 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.
2. 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.
3. 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.
4. 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:

3.3	deadband
A100 AV	operand1

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.

Example 2

```
IF_CHANGED 0.0 A100 1W ON11
COLOR FG RED BG WHITE
RECTANGLE 3342 2504 65 34 8141 2 dot_dashed unfilled
ENDIF
```

In this example:

0.0	deadband
A100 1W	operand1
ON11	operand2

This statement indicates that if bit 11 of A100 1W 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).

4.28 IF/ENDIF

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. All commands following the IF command up to the matching ENDIF command are executed if the condition is met. When you enter 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 enclosed in parenthesis.

RULES

1. The COLOR command should be used within the IF/ENDIF command to ensure appropriate display of color and blink for the display items within IF/ENDIF.
2. Each conditional expression must be enclosed in parentheses.
3. 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:

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

These statements describe the graphics commands that will be executed 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.

4.29 IFELSE/ELSE/ENDIF

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. 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. When you enter 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 enclosed in parenthesis.

RULES

1. The COLOR command should be used within the IFELSE/ELSE/ENDIF command to ensure appropriate display of color and blink for display items within IFELSE/ELSE/ENDIF.
2. Each conditional expression must be enclosed in parentheses.
3. 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:

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

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.

4.30 KEYBOARD

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, or TRIGGER command is encountered.

4.31 LINE

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 [conditional]
```

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 the line width array index.
line_pat	Line pattern name.
[conditional]	Optional conditional for line pattern.

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 may be followed by a conditional expression which will determine which line pattern to display at runtime. Conditional expressions are optional.
4. The LINE command may be used in the background, foreground, or trigger sections of a diagram.

EXAMPLES

Example 1

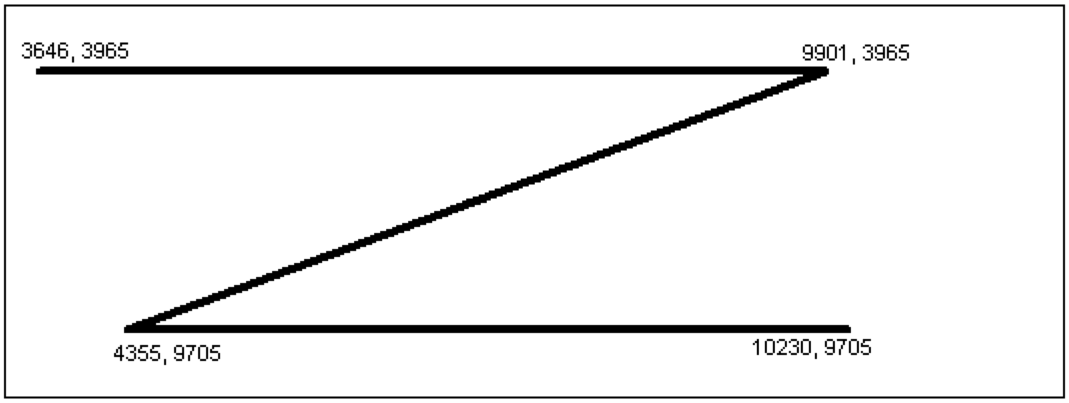
```
LINE 3646 3965 9901 3965 4355 9705 10230 9705 3 solid
```

In this example:

3646	x1
3965	y1
9901	x2

3965	y2
4355	x3
9705	y3
10230	x4
9705	y4
3	line_width
solid	line_pat

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 (which corresponds to a pixel line width of 4 in the line width table), and the line pattern is solid. 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:

1443	x1
10166	y1
3367	x2
1196	y2
5291	x3
10166	y3
1443	x4
10166	y4
1	line_width

solid	line_pat
(CASE) A5514 HL 1 1 2 near_solid dashed	[conditional]

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, which corresponds to a pixel line width of 1 in the line width table.

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.

4.32 LOAD_FKEY_GROUP

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, when that group is loaded, no function keys will be active. No error/warning is generated by GBNT 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.

4.33 LOOP/ENDLOOP

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 you enter 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)

RULE

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:

5	n

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:

10	n

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.

4.34 MACRO

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 you to continually redraw the same diagram or diagram section.

The MACRO command is used to integrate a separate diagram into the main diagram. When running GBNT in the offline mode, Macro files reside in the directory [InstallationPath]/mmi/graphics/macros (where “InstallationPath” is a user-defined directory). When running GBNT from the Ovation Developer Studio, they reside in the Macro node under the system-level Graphics node in the hierarchy. (See *“Developer Studio User Guide*.)

SYNTAX

`MACRO`	`macro#  x  y  wscale  hscale`
	`#$D  #$T  #$T(bg)  #$SET  #$CONST  #$STATUS`
	`$D1  $D2  ...  $Dn`
	`$T1  $T2  ...  $Tn`
	`$T1(bg)  $T2(bg)  ...  $Tn(bg)`
	`$SET1  $SET2  ...  $SETn`
	`$CONST1  $CONST2  ...  $CONSTn`
	`$STATUS1  $STATUS2  ...  $STATUSn`
	`#$COLOR  $COLOR1  $COLOR2  ... $COLORn`
	`#$OLCOLOR  $OLCOLOR1  $OLCOLOR2  ...  $OLCOLORn`

where:

macro#	Macro number (1 through 65,535) which specifies the macro file to read.
x	Virtual x coordinate (0 through 16,383) at which to position upper left corner of macro.
y	Virtual y coordinate (0 through 16,383) at which to position upper left corner of macro.
wscale	Width scale factor (positive real number).
hscale	Height of scale factor (positive real number).
#$D	Number of process points (0 through 99) passed to the macro (that is, number of $D parameters in the macro).
#$T	Number of text strings (0 through 50) passed to the macro (that is, number of $T strings in the foreground, triggers, and/or keyboard sections of the macro).

#$T(bg)	Number of text strings (0 through 50) passed to the macro (that is, number of $T strings in the background section of the macro).
#$SET	Number of set numbers (0 through 256) passed to the macro (that is number of $SET parameters in the macro).
#$CONST	Number of integer/real constants (0 through 256) passed to the macro (that is, number of $CONST parameters in the macro).
#$STATUS	Number of status words (0 through 256) passed to the macro (that is, number of $STATUS parameters in the macro).
$D1 - $Dn	If #$D > 0, this is the list of process point names (no record fields) to pass to the macro to replace $D parameters in the macro.
$T1 - $Tn	If #$T > 0, this is the list of quoted text strings to pass to the macro to replace $T parameters in the foreground, triggers, and/or keyboard sections of the macro.
$T(bg) - $Tn(bg)	If #$T(bg) > 0, this is the list of quoted text strings to pass to the macro to replace $T parameters in the background section of the macro.
$SET1 - $SETn	If #$SET > 0, this is the list of set numbers to pass to the macro to replace $SET parameters in the macro.
$CONST1 - $CONSTn	If #$CONST > 0, this is the list of integers/reals to pass to the macro to replace $CONST parameters in the macro.
#$COLOR	Number of colors (0 through 100) passed to the macro (that is, number of $COLOR parameters in the macro).
$COLOR1 - $COLORn	If #$COLOR > 0, this is the list of colors to pass to the macro to replace $COLOR parameters in the macro.
#$OLCOLOR	Number of OL colors (0 through 100) passed to the macro (that is, number of $OLCOLOR parameters in the macro).
$OLCOLOR1 - $OLCOLORn	If #$OLCOLOR > 0, this is the list of OL color indices to pass to the macro to replace $OLCOLOR parameters in the macro.

RULES

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

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

2. Macros are built within GBNT and can contain all of the graphics commands defined in the graphics language. (See *Ovation Graphics Builder User Guide*.)

3. When running GBNT in the offline mode, macro files reside in the directory:

   `[InstallationPath]/mmi/graphics/macros` (where "Installation Path is a user-defined directory).

   When running GBNT online from the Ovation Developer Studio, macro files reside in the Macro node under the System-level Graphics node in the hierarchy.

4. Macros can be imbedded in macros. You can pass a parameter coming into a top-level macro into a lower-level macro by passing $Dn, “$Tn”, $SETn, $CONSTn, and so forth, as the substitution values to the lower level macros. For example, you can call the following macro #2 from a parent macro #1:

   ```
   MACRO  2  1000  1000  1.0  1.0  1  1  0  1  2  0  $D1  “$T1”  $SET1
   $CONST1  $CONST2  1  $COLOR1  0
   ```

   In the above example, the first point passed into the parent macro #1 is passed into macro #2, along with the first text string, the first set, the first and second constants, and the first color.

5. If any of the count arguments (#$D, #$SET, #$COLOR, and so forth) are 0, then the associated list of substitution parameters ($D1...$Dn, $SET1...$SETn and so forth) is omitted from the source syntax.

6. 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 you specify 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, and so forth, 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 you specify 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, and so forth, 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, and so forth.

     $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 $SET1 0 1 6 0 5 A001X001 AV A001X002 AV 1 2 2”=>“POKE_FLD 1000 1000 2000 2000 ON 23 5 0 1 6 0 5 A001X001 AV A001X002 AV 1 2 2”
C. “COLOR FG RED ($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 a real number must have a digit before and after the decimal point! The first int/real number will be substituted for all occurrences of $CONST1 in the file, the second set will be substituted for $CONST2 in the file, and so forth. $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, and so forth. $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.
- A list of up to 100 colors to be substituted for $COLOR occurrences in the macro graphic. The colors passed to the macro are color names (red, green, blue, and so forth). The first color passed to the macro is substituted for all occurrences of $COLOR1 in the macro file, the second color passed is substituted for all occurrences of $COLOR2 in the file, and so on. The number of required colors that must be passed to the macro is the largest $COLOR index in the macro file. That is, if $COLOR1 and $COLOR10 are used in the macro file, 10 is the largest $COLOR index, so 10 colors are required to be passed to the macro file.
- A list of up to 100 OL colors to be substituted for $OL_COLOR occurrences in the macro graphic. The OL colors passed into the macro are specified as OL color indices. The first color passed to the macro is substituted for all occurrences of $OL_COLOR1 in the macro file, the second color passed is substituted for all occurrences of $OL_COLOR2 in the file, and so on. The number of required OL colors that must be passed to the macro is the largest $OL_COLOR index in the macro file.

7. When a macro is loaded into the parent diagram, GBNT checks that the correct number of parameters are passed to the macro for substitution. The exact number of expected parameters must be passed to the macro. It is not acceptable to pass too many or too few. The number expected of each parameter type is determined by the largest index used in the macro file. For example, if $D1 and $D34 are the only $D tagout parameters specified in the macro file, then the number of expected process point parameters to be passed to the macro will be 34 (since 34 is the largest $D index used in the macro file). Therefore, when creating macro files, do not skip indices.

8. The MACRO graphic command is valid in all sections of the parent graphic. However, depending on the contents of the particular macro, some macros are restricted as to where they can be added in the parent graphic. If the macro contains commands in more than one section (for example, commands in the background section and commands in the foreground section), the macro must be placed in the diagram section of the parent graphic. If the macro contains only keyboard data , the macro can go in the diagram or the keyboard section of the parent graphic. If the macro contains only background or only foreground or only triggerN data, the macro can go in any section of the parent graphic except the keyboard because the graphic commands that are valid in any one of these sections are also valid in the other sections.

   The place of items within the macro is always maintained if the macro is added to the DIAGRAM section of the parent graphic. It is possible to override the place of items defined in the macro in the parent graphic if/when the macro is added to the background, foreground, or triggerN sections of the parent graphic.

9. When multiple MACRO commands exist in the diagram section of the parent graphic, the order of the MACRO commands in the parent graphic defines the display order of the items comprising each macro on the parent graphic. When a macro is imported into the diagram section of a parent graphic, all of the commands in the background section of the macro are implicitly merged into the background section of the parent graphic. all of the commands in the foreground section of the macro are implicitly merged into the foreground section of the parent graphic, and so forth.

10. 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.

11. If the macro contains any items built with bitmap or bitmap_over type text, the macro cannot be scaled since bitmap text is not sizable. The wscale and hscale factors must be 1.0.

12. All parameters to the MACRO command after and including #$COLOR are new to version 2.1 and later of GBNT. There was/is no support for passing colors and OL colors into macros before that. For backwards compatibility, these parameters are optional in the source syntax. Pre-existing MACRO commands which do not have these arguments will compile and load. The #$COLOR and #$OLCOLOR parameters will default to 0 in those cases.

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 3 WHITE CYAN RED 1 2
```

In this example:

5	macro#
1234	x
5242	y
1.0	wscale
0.5	hscale
1	#$D

3	#$T
2	#$T(bg)
4	#$SET
2	#$CONST
2	#$STATUS
A100	$D1
“string 1” “string 2” “string 3”	$T1 - $T3
“bg string 1” “bg string 2”	$T(bg) - $T2(bg)
23 45 66 89	$SET1 - $SET4
21.76 89	$CONST1 - $CONST2
HDWRFAIL ALARMACK	$STATUS1 - $STATUS2
3	#$COLOR
WHITE CYAN RED	$COLOR1 - $COLOR3
1	#$OL_COLOR
2	$OL_COLOR1

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. “WHITE”, “CYAN”, and “RED” will be substituted for all occurrences of $COLOR1, $COLOR2, and $COLOR3 respectively in the file. The OL color index “2” will be substituted for all occurrences of $OL_COLOR1 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 $COLOR1 BG $COLOR2 ($SET2 > 45) $COLOR3
TEXT 444 71  “default bg” ($SET1) 2 “$T1” “$T2” HORZ VECTOR_OVER 152
562 1

FOREGROUND
COLOR FG $COLOR3 ($D1 AV > $CONST1) BLUE BG $COLOR1 (($D1 1W =
$STATUS1) OR ($D1 1W = $STATUS2)) YELLOW
MULTI_TEXT 701 2107 0 0 3 “$T1”  “$T2” “$T3” VECTOR_OVER 140 257 1

KEYBOARD
COLOR FG BLACK ($SET2 = 7) 3 $COLOR3 $COLOR1 BLUE OL $OL_COLOR1
POKE_FLD 398 655 1171 1078 ON 23 $SET4 0 3 6 0 5 A001X001 AV A001X002
AV 1 $CONST2 2 117 7001 7 5000 60 22 0 2 A001X011 AV A001X002 AV 119
0 4 1 0 3 “$T3”

OL_BUTTON 2107 655 HORZ ROUNDED SHAPE_LABEL 1000 1000 500 500 pointer
90 NONE EXEC_POKE 23 $SET3 0 2 6 0 5 A001X011 AV A001X012 AV 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\ 1W =
HDWRFAIL) OR (\A100\ 1W = 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 OL 2
POKE_FLD 398 655 1171 1078 ON 23 89 0 3 6 0 5 A001X001 AV A001X002 AV
1 89 2 117 7001 7 5000 60 22 0 2 A001X011 AV A001X002 AV 119 0 4 1 0
3 “string 3”

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

4.35 MATH

The MATH command evaluates a mathematical equation and then stores the result in an area of memory in the Operator Station called the scratch pad. The scratch pad is addressed by a “$P” pointer with a defined base address and specified offset.

SYNTAX

```
MATH pointer offset equation
```

where:

pointer	$P1 through $P99
offset	$P offset into the scratch pad ($Bn, $In, $Rn, or $Sn), where $Bn is a byte offset, $In is an integer offset, $Rn is a real number offset, and $Sn is a short integer offset. All offsets begin at 0 and increment by the type of offset up to the size of the scratch pad.
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 or as an operand to another MATH command.
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 equation may be fully parenthesized, but does not need to be fully parenthesized. The equation should not be delimited by quotes. Only the math operators listed in Rule #1 above are supported.
8. The pointer offset parameter is optional. If not specified, it defaults to $B0. You should specify the parameter, though, as a byte may not be large enough to store the MATH result (that is, if a real number result is returned, the real number will overflow the byte reserved for the result and undefined results occur). Typically, the offset parameter should be a $Rn offset to guarantee that the result will not overflow the pointer offset reserved for it.
9. 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:

$P1	pointer
$R0	offset
A100 + A200	equation

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 $R4 Sqrt ($P1 $R0)
```

In this example:

$P5	pointer
$R4	offset
Sqrt ($P1 $R0)	equation

This statement says to take the square root of $P1 $R0 and place it in $P5 $R4.

4.36 MULTI_TEXT

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 one string is identical in function to a TEXT command (with horizontal orientation).

Syntax

```
MULTI_TEXT x y justify spacing n string1 string2 ... stringn
[conditional] font_type font_size
```

where:

x	x coordinate of baseline position of first character of the first string (if font_type = BITMAP, BITMAP_OVER). ***OR*** x coordinate of upper left corner of outlining rectangle of the overall multi-text item (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). ***OR*** y-coordinate of upper left corner of outlining rectangle of the overall multi-text item (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 between successive strings. 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.
[conditional]	Optional conditional expression which can display another set of n strings.

font_type	Font types. The choices are: ▪ BITMAP - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ BITMAP_OVER - non-scalable; character and background drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**. ▪ VECTOR_OVER - scalable; character and background drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. **bitmap_font_number** - Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through 8. **charw** - Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.

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. You 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. The conditional is optional.
5. Strings are limited to 30 characters in the MULTI_TEXT command.
6. 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:

2177	x
3090	y

0 (left)	justify
0	spacing
3	n
“string 1”, “of”, “series of 3”	strings
VECTOR	font_type
140	charw
257	charh
1	lw

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 140 x 257. Single line width is used. No conditional strings are specified.

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 one of the two sets of three strings specified, right justified, at position 2177, 3090 on the diagram. If the analog value (AV) of \A100\ is greater than 100, the conditional strings will display instead of the default strings. The first character of the longest 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.

4.37 OL_BUTTON

The OL_BUTTON command defines a button on a process diagram that is activated when selected with the mouse.

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 you press and release the OL_BUTTON. A poke functionality specifies that any of the standard poke functions will execute when you press and release the OL_BUTTON. Either a poke or a trigger functionality is associated with an OL_BUTTON - not both. When you press the OL_BUTTON, it appears depressed until you release it (by releasing the button on the mouse) or moves the mouse off the OL_BUTTON.

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. Valid range = 0 through 16,383.
y	y coordinate of the upper left corner of the outlining rectangle around the button. 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** = enter the following seven parameters: 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 *Ovation Graphics Builder User Guide*.) [conditional] = Optional shape name conditional expression. 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** = choose either vector or bitmap text followed by their applicable parameters: VECTOR - enter charw, charh, lw, string, and [conditional] BITMAP - enter font#, string, and [conditional] where: charw = Width of one character. Valid range = 3 through 16,383 charh = Height of one character. Valid range = 3 through 16,383 lw = Line width. Valid range = 1 through 16 for Ovation Vector font. For all vector fonts, 1 must be entered since only Ovation Vector font supports line width. 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. [conditional] = Optional string conditional expression.

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. Valid range = 1 through 255
	▪ **EXEC_POKE** = poke_type poke_args where: poke_type = One of the standard poke types. The choices are: 0, 2, 3, 6, 7, 8, 9, 20, 23 poke_args = Argument list for the specified poke_type

RULES

1. For an OL_BUTTON with a text label, the button orientation also defines the text orientation**.** The character width and character height 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.
3. If a shape label is selected for the OL_BUTTON, the shape name may be followed by a conditional expression which will determine which shape name to display at runtime. If a text label is selected for the OL_BUTTON, the string may be followed by a conditional expression which will determine which string to display at runtime. Conditional expressions are optional.
4. For an OL_BUTTON with a shape label, the shape is always centered in the middle of the button. The shape origin is ignored.
5. 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.
6. Text button labels are limited to 30 characters.
7. The origin of the button is the upper left corner of the button's outlining rectangle (x, y).
8. Poke type 0 (brings up an Ovation system Point Information window) is not supported for buttons.

9. If more than one application program is to be run with a poke type 7 or 23 button, 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.
10. The OL_BUTTON is only valid in the keyboard section of the diagram.

EXAMPLES

Example 1

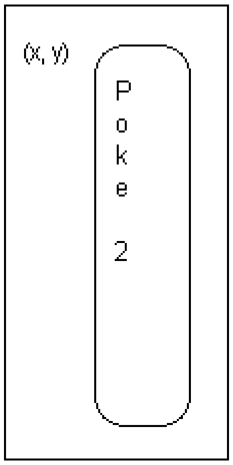
```
OL_BUTTON 4915 13739 VERT SQUARED TEXT_LABEL VECTOR 140 257 1 “poke
2” EXEC_POKE 2 3000 5000
```

In this example:

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

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 you press and release 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 the following figure:



Example 2

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

In this example:

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

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 you press and release the button.

4.38 OL_CHECKBOX

The OL_CHECKBOX command displays a checkbox item on a process diagram.

You specify the number of checkboxes, the size of the checkboxes, the orientation for the series of checkboxes, and the spacing between the checkboxes. You also specify 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. The value of the checkbox is read from and written to this location at runtime.

The scratch pad value will be read at runtime when the diagram is initially displayed to determine which boxes are initially checked on the Operator Station. 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. When you select or deselect a box at runtime, 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 you select a box for an exclusive item, a checkmark is drawn inside that box, and the existing checkmark is erased. When you select 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, you click on a box that is checked which erases the checkmark (non-exclusive items allow you to toggle the state of the checkboxes).

SYNTAX

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

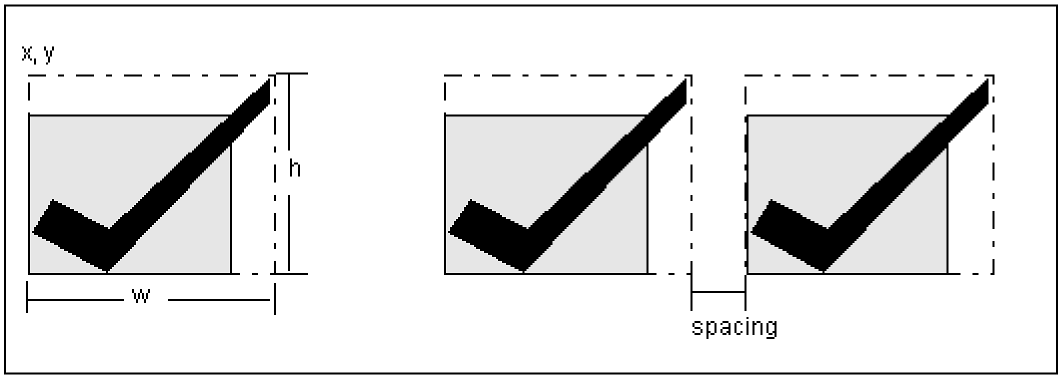
where:

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 $In offset) from which to read checkbox item value and to which to write checkbox item value.
trigger	Trigger to execute when the value of the checkbox item changes. (0 means do not execute a trigger.) 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 (when you select another box at runtime). You should set the ER color to the current BG color on which the checkbox item lies so that this erasure is not visible.
3. GBNT 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 Station 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 2boxes - 1 (for example, if boxes = 3, then 2 * 2 * 2 = 8, and 8 - 1 = 7, so the upper limit is 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

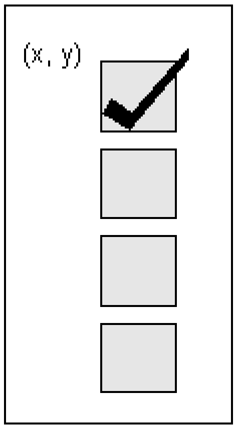
Example 1

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

In this example:

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

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 you check or uncheck 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:

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

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 you check 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).

4.39 OL_CHOICE

The OL_CHOICE command displays a choice item on a process diagram.

You specify 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. The value of the choice is read from and written to this location at runtime. 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. When you select or deselect a box at runtime, 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 scratch pad will be read at runtime when the diagram is initially displayed on the Operator Station to determine which boxes are initially depressed.

In addition, the value of the choice item is read each time the foreground of the diagram updates at runtime. If the value changes, the display of the choice item changes accordingly. When you select a box for an exclusive item, that box is depressed and the existing depressed box is raised. When you select 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, you click on a box that is depressed (non-exclusive items allow you 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 you select or deselect 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** = Choose either vector or bitmap text followed by their applicable parameters: VECTOR - enter char_w, char_h, lw BITMAP - enter 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. lw = Line width. Valid range = 1 through 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width. bitmap_font_number = Bitmap font number. Valid range = 1 through 8.
	▪ **SHAPE_LABEL** = Enter the following four parameters: 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** = enter shape1, rot1, inv1, ... shapen, rotn, invn where: shape1..shapen = ASCII shape name from Shape Library. (See *Ovation Graphics Builder User Guide*.)
	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 at runtime.
2. The choice labels (shape or text) are all the same color. You 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. 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. You 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.
6. GBNT always displays ol_choice with the first box depressed (only the first box). You cannot press/raise choice boxes in choice items in GBNT.
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 2n - 1 (for example, if n = 3, then 2*2*2 = 8 and 8 - 1 = 7, so the upper limit is 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 *Ovation Graphics Builder User Guide*.)

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

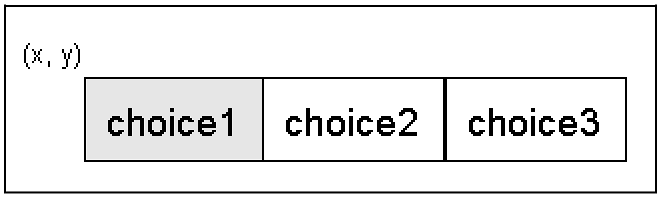
Example 1

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

In this example:

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

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 you select one of the 3 choice boxes on the Operator Station at runtime. See the following figure:



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:

1000	x
1000	y
3	n
VERT	orientation
NON_EXCL	type
$P2 $I10	value
0	trigger
SHAPE_LABEL	label_type
2000 (box_w) 1000 (box_h) 500 (shape_w) 500 (shape_h)	label_info

valve1 (shape_name1) 0 (rot1) NONE (inv1);
valve2 (shape_name2) 90 (rot2) NONE (inv2);
valve3 (shape_name3) 180 (rot3) NONE (inv3)
	label1...labeln

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. The three shapes used as labels are: valve1, valve2, and valve3 respectively. None of the shape labels are inverted as each shape label has an inversion parameter = NONE. The second and third shape labels are rotated and the first is not.

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 you select or deselect a choice since the trigger = 0.

4.40 OL_CYLINDER

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 *Ovation Record Types Reference Manual*.)

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. Valid range=0 through 16,383.
y	y coordinate of the upper left corner of the cylinder's outlining rectangle. 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. Valid values are process point names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names.
rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth).
low	Low limit used to scale the value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
high	High limit used to scale the value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)

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 at runtime.
5. The OL_CYLINDER command may reside in the background, foreground, or trigger sections of a diagram.

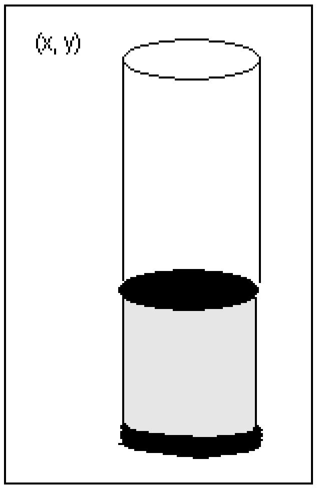
EXAMPLE

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

In this example:

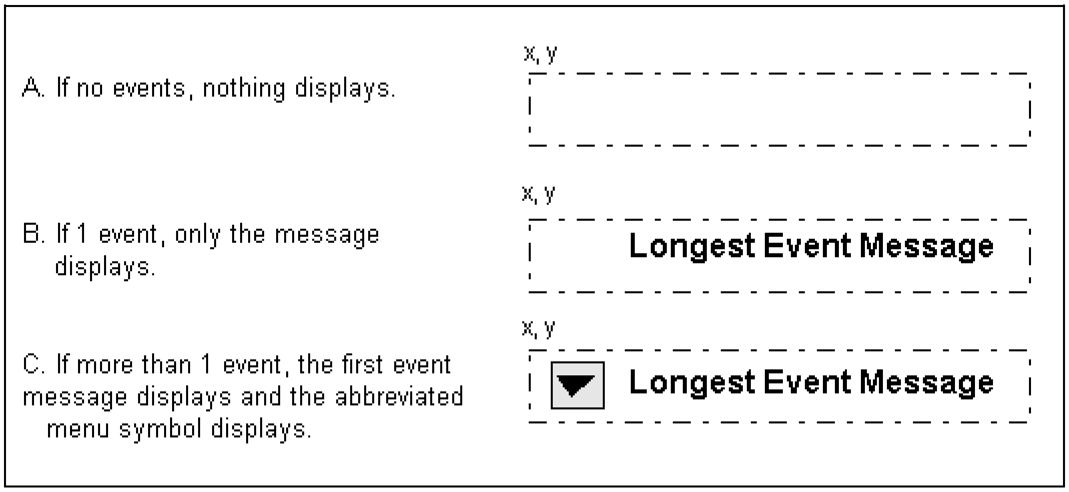
5951	x
13305	y
2179	w
10488	h
A3890	pt_name
AV	rec_fld
20	low
80	high

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.



4.41 OL_EVENT_MENU

The OL_EVENT_MENU command displays an abbreviated menu button on a diagram. 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 at runtime, nothing displays on the diagram (including the abbreviated menu symbol). If only one alarm condition is true at runtime, the associated alarm message displays, but no abbreviated menu symbol displays. If more than one alarm condition is true at runtime, the first alarm message is displayed along with the abbreviated menu symbol. See figure below.


A. If no events, nothing displays.
x, y
B. If 1 event, only the message displays.
x, y
Longest Event Message
C. If more than 1 event, the first event message displays and the abbreviated menu symbol displays.
x, y
Longest Event Message

If you click on the abbreviated menu symbol when displayed on the Operator Station at runtime, a window 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. You cannot select from this menu (that is, you cannot acknowledge alarms from this menu). If you click on the abbreviated menu symbol while the menu is being displayed, the menu will disappear.

Note: *GBNT will always display the abbreviated menu symbol and the longest alarm or event message. GBNT will NOT support popping up the event menu when you click 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_type font_size
```

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	Process point names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names.
recfld_1 ... recfld_n	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth) to be used with $P and $D points.
status_1 ... status_n	Status word.
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_type	Font types. The choices are: ▪ BITMAP - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. **bitmap_font_number** - Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through 8. **charw** - Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.

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.
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.
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. ASCII record fields are not supported.
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 1W HDWRFAIL “A100 is timed-out” A100
1W BAD “A100 is bad” A100 1W FAIR “A100 is fair” A100 1W 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	1W
status_1	HDWRFAIL
msg_1	“A100 is timed-out”
pt_name2	A100
rec_fld_2	1W
status_2	BAD
msg_2	“A100 is bad”
pt_name3	A100
rec_fld_3	1W

status_3	FAIR
msg_3	“A100 is fair”
pt_name4	A100
rec_fld_4	1W
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 1W 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. GBNT will display the “A100 is timed-out” string on its canvas since this is the longest string.

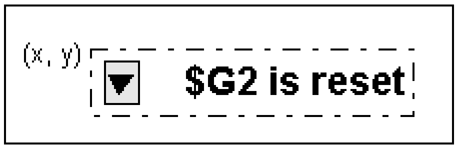
Example 2

```
OL_EVENT_MENU 1000 2000 3 $G1 1W SET “$G1 is set” $G2 1W RESET “$G2
is reset” $G3 1W SET “$G3 is set” BITMAP 2
```

In this example:

x	1000
y	2000
n	3
pt_name1 rec_fld_1 status_1 msg_1	$G1 1W SET “$G1 is set”
pt_name2 rec_fld_2 status_2 msg_2	$G2 1W RESET “$G2 is reset”
pt_name3 rec_fld_3 status_3 msg_3	$G3 1W SET “$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 1W field of a digital point against the SET or RESET status words. Three unique points are used in this example. GBNT will display the “$G2 is reset” string on its canvas since this is the longest string. See figure below:



4.42 OL_GAUGE

The OL_GAUGE command displays a gauge, similar in functionality to a bar graph. 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	Process point names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names.
rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth) to be used with $P and $D points.
low	Low limit used to scale the value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (e.g.: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
high	High limit used to scale the value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (e.g.: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)

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.
5. A rounded 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.

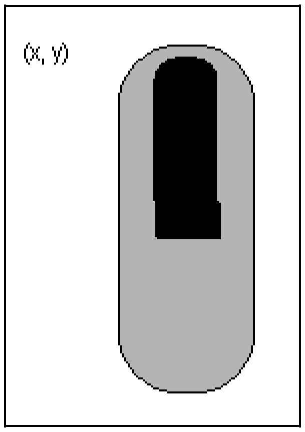
EXAMPLE

```
OL_GAUGE 5951 13305 2179 10488 DOWN A3890 AV 20 80
```

In this example:

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

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:



4.43 OL_RECTANGLE

The OL_RECTANGLE command displays a rectangle on the graphic. 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-dimensional 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.
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:

5000	x
6000	y
1000	w
3000	h
INVOKED	state

In this example, an OL 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.

4.44 OL_SLIDER

The OL_SLIDER command displays a slider on the graphic. The slider is both an input and output device at runtime. The slider control (the box you grab (with the mouse) that moves along the slider bar) represents the current value of the slider point at runtime. This value is read/written at a specified address in the scratch pad and is scaled between a low and high limit. When you move the slider control at runtime, the data within the scratch pad is updated. The value of the slider (scratch pad area) will be read each time the foreground updates on the Operator Station at runtime. If the value changes, the slider display will update accordingly. The scratch pad value will be read when the diagram is initially displayed at the Operator Station to set the slider's initial value on the graphic.

Two unique triggers may be associated with the slider. One trigger (trigger 1) is called continuously as you are moving the slider control. The other trigger (trigger 2) is called once when you release the mouse button after dragging the slider control.

Note: *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 you release 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.

SYNTAX

```
OL_SLIDER x y w h direction value low_limit high_limit 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 Þ - 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).

low_limit	Low limit used to scale the value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
high_limit	High limit used to scale the value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
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 you move or drag it. You should take care to set the ER color to the background color on which the slider lies. ER color is only used at runtime in conjunction with the OL Slider.
2. The value of the slider (scratch pad area) will be read each time the foreground updates on the Operator Station 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

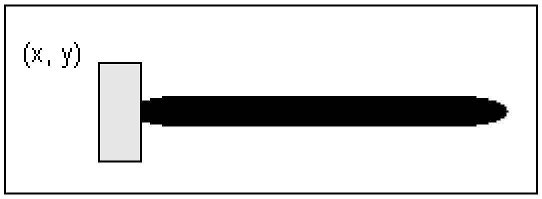
```
OL_SLIDER 5000 3000 4000 1000 LEFT $P1 $R0 0 100 44 54
```

In this example:

5000	x
3000	y
4000	w
1000	h
LEFT	direction

$P1 $R0	value
0	low
100	high
44	trigger1
54	trigger2

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 you move 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:



4.45 PAGE

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 Ovation Developer Studio User Guide.)*

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 Number	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.

Diagram Number	Corresponding Group	Result
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.

4.46 PLOT

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). You specify 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	Process point names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names.
rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth) to be used with $P and $D points.
low	Low scale value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)

high	High scale value. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
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. The plot_char_params arguments are required to define the shape to display with this option. ▪ SQUAREWAVE - t 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) Enter the following items: name, width, height, rotation, and 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 - choose one of the following: - 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:

5848	x
600	y
1200	l
DOWN	direction
A100	pt_name
AV	rec_fld
-10	low
10	high
LINES	plot_char
Not applicable for plot_char = LINES	plot_char_params

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:

5848	x
600	y
1200	l
DOWN	direction
A100	pt_name
AV	rec_fld
-10	low
10	high
SHAPE_PLOT	plot_char

ARROW 500 300 90 NONE
where:
name= ARROW
width=500
height=300
rotation=90
inversion= NONE
	plot_char_params

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:

[200]	x
[0]	y
3838	l
UP	direction
$P1	pt_name
$S0	rec_fld
0	low
1	high
SQUAREWAVE	plot_char
Not applicable for plot_char = SQUAREWAVE	plot_char_params

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.

4.47 POINTER

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 Station.

SYNTAX

```
POINTER pointer_name segment offset
```

where:

pointer_name	$P or $H names. Valid range = $P1 through $P99 and $H1 through $H99.
segment	Memory segment in the Operator Station to which to point. Must be one of the defined segments from the segment table (see below). Valid range = 0 through 255 (may be relative)
offset	Offset into memory segment in the Operator Station to which to point. Offset must not exceed the size of the particular segment as defined in the segment table. See Rules for valid ranges (may be relative).

RULES

1. You 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 Emerson 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, Controller, VAX, and so forth).
200	512	Reserved for Ovation internal processing.
207	1000	$G points CRT1.
209	140	Window diagram header gcode.
211	140	Main diagram header gcode.
212	140	Subwindow diagram header code.
213	internal	Paging Data - Main & Control windows
214	internal	Display Data-same as display data window

Segment Number	Segment Size	Description
215	4	Data structures containing valid drop and highway numbers.
216	4	Entry field data for main window.
217	4	Entry field data for window.
218	4	Entry field data for window.
219	20320	Entry field buffers for main window.
220	20320	Entry field buffers for window.
221	20320	Entry field buffers for window.
224	1	$G points CRT2 - (6- and 7-level software only).
234	20	Ovation Information.
237	12	Tune data information.
238	20	Control point ID's sysid1 and sysid2.
254	1	Customer scratch pad CRT2 - Only compatible with 6- and 7-software level Operator Station.
255	100	Customer scratch pad CRT1.

The following Emerson segments have specific portions that may be of use to you. The segment numbers, offsets and descriptions are listed below.

Segment Number	Segment Offset	Format	Description
207	0	long integer	References $G1, and any select on a poke type 0 fills in $G1.
211	24	long integer	Main diagram number
234	0	long integer	Display drop system ID (does not include network ID).
234	4	short integer	Set when window is selected
234	6	short integer	Main diagram group number
234	10	short integer	Window diagram group number
238	0	long integer	Algorithm point name and ID record field for programmable key functions 1 through 4, RAISE SETPOINT, LOWER SETPOINT, START, and STOP.

Segment Number	Segment Offset	Format	Description
238	4	long integer	Algorithm point name and ID record field for programmable key functions 5 through 8, AUTO MODE, MANUAL MODE, RAISE OUTPUT, and LOWER OUTPUT.
238	8	long integer	Value to which to change selected control ID.
238	12	long integer	Top of scale value for the algorithm being changed.
238	16	long integer	Bottom of scale value for the algorithm being changed.

EXAMPLE

```
POINTER $P2 20 99
```

In this example:

$P2	pointer_name
20	segment
99	offset

This statement initializes the value of $P2 to segment 20 offset 99.

4.48 POKE_FLD

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 = run 1 Application program (no arguments can be passed to it) ▪ 6 = Ladder diagram ▪ 7 = run 1 or more Application programs (arguments can be passed) ▪ 8 = Display window and pass $W points ▪ 9 = Operating system command ▪ 20 = Help (not yet implemented) ▪ 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, dummy point name, $pointer ($P, $G, $D, $W, $H, or $O)

- 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 use current group**.**

- POKE_FLD x y w h state 3 prog_num

  where:

prog_num	Application program number of a program that takes no arguments.

- 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. ***OR*** $Pn $Im pointer/offset pair (where m is a multiple of 4)
diag_num	Diagram number. Valid range = 0 through 65,535. ***OR*** $Pn $Im pointer/offset pair (where m is a multiple of 4).

num_of_args	Number of arguments to specify for the application program chosen.
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 and 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 use current group.
pt_name1, pt_name2, pt_nameN	Point name (without record field) to substitute for $W points in the window.

- POKE_FLD x y w h state 9 command_line

  where:

command_line	ASCII command string including arguments, enclosed in single or double quotes. String is limited to 130 characters.

- POKE_FLD x y w h state 20 “file:keyname”

  At this time, the Help poke field is not yet implemented.

- 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 and window diagrams. Set1 in the main screen is the same as set1 in the window.
set_val	Integer value to assign to variable Valid values are 0 through 32,767.
num_of_progs	Number of application programs to run. Valid values are 1 through 65,535.
prog_num	Application program number (may be specified as an integer or by a $Pn $Im pointer/offset pair, where m is a multiple of 4).
diag_num	Diagram number from 0 through 65,535 to display with associated program (may be specified as an integer or by a $Pn $Im pointer/offset pair, where m is a multiple of 4).
num_of_args	Number of arguments to specify for the application program chosen.
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 $SETn ▪ macro input parameters: “$T”, $CONSTn, $SETn, $Dn, $STATUSn

RULES

1. The possible poke field functions are:

   - **Poke type 0** displays information on the process point specified. When you click on this poke field, the Point Menu pop-up window displays with the point name given at the top of the window. From the Point Menu, you can access the Ovation Point Information program for the given point. (See *Ovation Operator Station User Guide*.)
   - **Poke type 2** displays a specified process diagram with a specified group of points.
   - **Poke type 3** runs an application program specified by a program number. No arguments can be passed using poke type 3, so this can only be used to run an application program which does not take any arguments. Also, this poke only runs one application program. Use poke type 7 to run application programs to take arguments and/or to run a set of application programs.
   - **Poke type 6** selects an element (contact/coil) on a ladder diagram for control. This poke field type is reserved for Emerson use only.
   - **Poke type 7** runs one or more application programs specified by program number and passes any required arguments.

- **Poke type 8** displays a specified window diagram with a point group and a list of points to substitute for $W points.
- **Poke type 9** runs an operating system process defined by a command line.
- **Poke type 20** displays help on a defined item. At this time, the Help poke field is not yet implemented.
- **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. GBNT always displays poke fields in the current color. Poke fields do not display at runtime, though, Pokes cannot be activated in GBNT (that is, nothing happens if you click on them in GBNT). Poke fields are only functional at runtime.
3. A poke field can be turned on or off with the POKE_STATE command. If a poke field is off at runtime, a box is not displayed around the poke field area at the Operator Station, and you cannot activate the poke by selecting it with the cursor positioning device at the Operator Station.
4. Poke type 0 only uses the point name, not the record field. The source syntax accepts a record field and defaults if it is not given. **At runtime, the record field is not used.** Therefore, you may pass any valid record field for the given point.
5. OPC points cannot be used with a poke type 0. An error will be generated.
6. 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.
7. The POKE_FLD command can only be used in the keyboard section.

EXAMPLES

Example 1

```
POKE_FLD 965 14447 869 1389 ON 0 A10ALM AV
```

In this example:

965	x
14447	y
869	w
1389	h
ON	state
0	poke_type
A10ALM AV	pt_name

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 Station. When you click on this poke field, (specified as type 0), the Point Information window will appear on the Operator Station, displaying information on A10ALM AV.

Note that the AV field is passed, but it will not be used at runtime. Passing HL, LL, ID, 1W, 2W, and so forth would all have the same effect. Any valid record field for A10ALM will do since the record field is not used. The record field is still part of the syntax for backward compatibility purposes.

Example 2

```
POKE_FLD 483 14400 546 904 ON 2 2450 1200
```

In this example:

483	x
14400	y
546	w
904	h
ON	state
2	poke_type
2450	diag_num
1200	group_num

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 you click 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 you 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:

965	x
14445	y
869	w
1389	h
ON	state
9	poke_type
“/usr/wdpf/mmi/bin/control_panel”	command_line

This statement defines a poke field at x, y position 965, 14445. It has a width of 869 and a height of 1389. When you click 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.

4.49 POKE_STATE

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. 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, you 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

```
KEYBOARD
POKE_FLD 1522 1425 869 637 ON 0 A100 AV
BACKGROUND
POKE_STATE 1522 1425 OFF
```

In this example:

1522	x
1425	y
OFF	state

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.

4.50 POLYGON

The POLYGON command draws a polygon defined by a set of coordinate pairs (which define the vertices of the polygon). This command automatically draws a line joining the first and last vertices, so it is not necessary to add the first point to the end of the polygon.

SYNTAX

```
POLYGON x1 y1 x2 y2 ... xn yn line_width line_pat [conditional]
fill_pat [conditional]
```

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 (3 <= *n* <= 255).
y*n*	y coordinate for end of the polygon. The maximum number of points per polygon is 255 (3 <= *n* <= 255).
line_width	Line width. Valid range = 1 through 16. The value corresponds to the line width array index.
line_pat	Line pattern name.
[conditional]	Optional conditional for line pattern.
fill_pat	Fill pattern name.
[conditional]	Optional conditional for fill pattern.

RULES

1. The x and y coordinates can be relative.
2. The origin is the first vertex of the polygon.
3. You specify the line and fill patterns to use when drawing the polygon.
4. The line pattern and fill pattern names may be followed by conditional expressions which will determine which line/fill pattern to display at runtime. The conditional expressions are optional.
5. The POLYGON command may be used in the background, foreground, or trigger sections of a diagram.

EXAMPLES

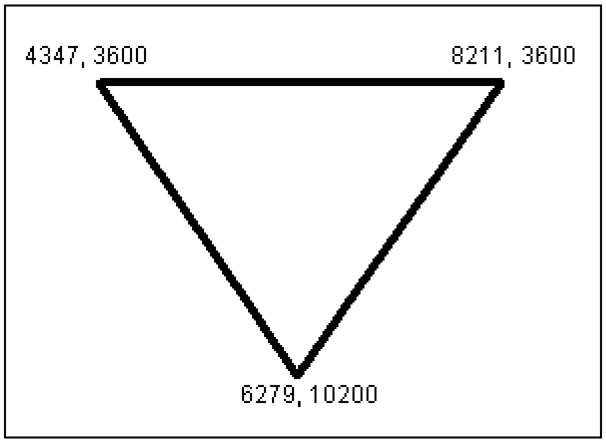
Example 1

```
POLYGON 4347 3600 8211 3600 6279 10200 4347 3600 3 solid unfilled
```

In this example:

4347	x1
3600	y1
8211	x2
3600	y2
6279	x3
10200	y3
4347	x4
3600	y4
3	line_width
solid	line_pat
unfilled	fill_pat

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 (which corresponds to a pixel line width = 4 in the line width table), 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:

4938	x1
1724	y1
9951	x2
4997	y2
7090	x3
10070	y3
3	line_width
solid	line_pat
unfilled	fill_pat (good_cond_value)
(QUALITY)	fill conditional expression
\T458\	pt_name
back_slash	fair_cond_value
asterisks	poor_cond_value
west_logo	bad_cond_value
blocks	timed_out_cond_value

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.

4.51 PROCESS_PT

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 font_size pt_name rec_fld [conditional]
[low_limit high_limit]
```

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. ▪ DATE_ONLY - Displays the value of the U4 or U6 record fields as only the date. For example, 12/31/01. ▪ TIME_ONLY - Displays the value of the U4 or U6 record fields as only the time. For example, 19:00:00. ▪ DATE_TIME - Displays the value of the U4 or U6 record fields as the date and time. For example, 12/31/01 19:00:00. ▪ DATE_TIME_ZONE - Displays the value of the U4 or U6 record fields as the date, time, and time zone. For example, 12/31/01 19:00:00 EST. ▪ DATE_TIME_NOSEC - Displays the value of the U4 or U6 record fields as the date and time, with no seconds shown. For example, 12/31/01 19:00. ▪ DATE_NOYEAR_TIME_NOSEC - Displays the value of the U4 or U6 record fields as the date with no year, and the time with no seconds shown. For example: 12/31 19:00.
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	Font types. The choices are: ▪ BITMAP - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ BITMAP_OVER - non-scalable; character and background drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**. ▪ VECTOR_OVER - scalable; character and background drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. **bitmap_font_number** - Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through 8. **charw** - Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.
pt_name	Process point names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names.
rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth) followed by optional conditional expression.
[conditional]	Optional conditional for process point record field.
[low_limit high_limit]	Optional low and high values to use for scaling to display the process point value as a percentage. Low/high limits can be specified by any of the following: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)

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 and bitmap_over).
   - Upper left corner of the text outlining rectangle (for vector and vector_over).
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>). You 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 may be followed by a conditional expression which will determine which record field within the given point will be displayed at runtime. The conditional expressions are optional.
6. OPC points (that is, values) can be displayed using this command. But, since OPC points do not have associated record fields, omit the rec_fld parameter when using an OPC point for the pt_name parameter. Also, since there is no record field for an OPC point, there can be no record field conditional either. You cannot specify a conditional for the entire OPC point in this command. In fact, there is no support in GBNT for OPC conditionals.
7. GBNT does not display the actual record field value. GBNT displays a series of "?" characters to represent a process point display. The series will reflect the number of characters, the decimal places, and the quality parameters. For example, if the number of characters is 7, the decimal places is 2, and the quality is ON, GBNT will display "???.??Q".

   If the quality = OFF in the prior example, GBNT will display "????.??". Note that the quality character (if specified) takes up one of the total number of characters of the display.
8. 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:

9620	x
1794	y
6	number_of_chars
2	decimal_places
RIGHT0	format

ON	quality
HORZ	orientation
VECTOR	font_type
321	charw
734	charh
1	lw
A300	pt_name
AV	rec_fld
-5	[low_limit]
10	[high_limit]

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. You 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 10 2 HEX OFF VERT BITMAP 5 A100 LL
((A500 LL > 50) AND (A010 <= A500 LL)) HL
```

In this example:

688	x
7800	y
10	number_of_chars
2	decimal_places
HEX	format
OFF	quality
VERT	orientation
BITMAP	font_type
5	bitmap_font_number

A500	pt_name
LL	rec_fld
((A500 LL > 50) AND (A010 <= A500 LL)) HL	[conditional]

This statement displays the contents of A500 LL located at 688, 7800. The width of the process point is 10, 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. The record field value is not displayed as a percentage as the optional low_limit and and high_limit parameters are omitted.

A conditional is used to determine which record field value is displayed at runtime. If A500 LL > 50 AND A010 <= A500 LL, the record field displayed will be HL. Otherwise, the record field will be LL.

4.52 PTR_EQUAL

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 Station memory. PTR_EQUAL changes location as opposed to PTR_VALUE which changes data.

Syntax

```
PTR_EQUAL pointer_name1 pointer_name2
```

where:

pointer_name1	$P or $H pointer name. Valid range = $P1 through $P99 and $H1 through $H99.
pointer_name2	$P or $H pointer name. Valid range = $P1 through $P99 and $H1 through $H99.

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 diagram, background, foreground, and trigger sections of a diagram.

Example

```
POINTER $P16 9 15
PTR_EQUAL $P3 $P16
```

In this example:

$P3	pointer_name1
$P16	pointer_name2

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.

4.53 PTR_LOOP/P_ENDLP

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 you enter 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	Process point name OR $ pointer identifier ($Pn, $Gn, $Dn, $Wn, $Hn, or $On)
rec_fld	Non-ASCII, two-character record field name (1W, AV, ID, and so forth) OR non-ascii $ offset identifier ($B0, $I4, $R0, and so forth)

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 $AnXi offsets are not. It is your responsibility to supply the correct value for n for the given offset (for example, n must be a multiple of 4 for $R offsets).
2. The PTR_LOOP/P_ENDLP commands may be used in the diagram, 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
   PTR_MOVE $P2 42
P_ENDLP
```

In this example:

$P1	pt_name
$B0	rec_fld

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 (which is segment #9, offset 14). The value of the first process point is obtained from $P2 $S0 (Bytes 0 and 1 of segment #9 for the first loop). The value of the second process point is obtained from $P2 $S2 (Bytes 2 and 3 of segment #9 for the first loop). The value of the third process point is obtained from $P2 $S4 (Bytes 4 and 5 of segment #9 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 PTR_MOVE command within the loop increments pointer $P2 by 42 bytes. Therefore, for the second loop, the process point values are obtained from Bytes 42 through 55.

4.54 PTR_MOVE

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 you to change the current pointer offset by an integral value, using mathematical operations (addition, subtraction, and multiplication). Note that the pointer address changes - not any data pointer to be the pointer. Use the PTR_VALUE command to change data addressed by a pointer.

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.
operation	Mathematical operation. Choose one of the following: ▪ **ADD** (addition; value of operand1 added to current offset of pointer_name) ▪ **SUB** (subtraction; value of operand1 subtracted from current offset of pointer_name) ▪ **MULT** (multiplication; value of operand1 is multiplied by operand3 and then either added or subtracted from current offset of pointer_name according to operand2)
operand1	Used for all operations. May be specified by any of the following: ▪ Point name and two-character non-ascii point record field (for example: A100 1W, \TESTAI01\ AV) ▪ $ Pointer notation and non-ascii $ offset or two-character point record field (for example: $P1 $R0, $D1 AV, $D1 $I0) ▪ Integer (between -2,147,483,647 and 2,147,483,647)
operand2	Only applicable for operation = MULT. Specifies whether (operand1 * operand3) should be added or subtracted from current pointer offset. The choices are: ▪ ADD (operand1 * operand3 will be added to current offset of pointer_name) ▪ SUB (operand1 * operand3 will be subtracted from current offset of pointer_name)
operand3	Only applicable for operation = MULT. Specifies the multiplier (0 through 255) to multiply by operand1, the result which will then be either added or subtracted from the current offset of pointer_name.

Rules

1. Only $P and $H pointer types are valid for this command.
2. A non-ASCII record field is required if operand1 is specified by a point name/record field or a $pointer name/$offset. For $ pointer names with $offsets, this means that $In, $Bn, $Sn, and $Rn offsets are supported, but $AnXi offsets are not. It is your responsibility to supply the correct value for n for the given $offset (for example, n must be a multiple of 4 for $R offsets). For process point names with two-character record fields, this means that 1W, AV, LL, HL, and so forth, are supported, but ED, EU, PN, and so forth, are not.
3. The PTR_MOVE command can be used in the diagram, background, foreground, and trigger sections of a diagram.

Examples

Example 1

```
POINTER $P16 0 12
PTR_MOVE $P16 ADD 10
```

In this example:

$P16	pointer_name
ADD	operation
10	operand1
(not used)	operand2
(not used)	operand3

Pointer $P16 initially is set to point to segment #0, offset 12. The PTR_MOVE command adds the integer 10 to the initial offset of 12 to get a new offset of 22 (12 + 10 = 22), so now $P16 points to segment #0, offset 22.

Example 2

```
POINTER $P16 0 12
PTR_VALUE $P2 $I0 2
PTR_MOVE $P16 SUB $P2 $I0
```

In this example:

$P16	pointer_name
SUB	operation
$P2 $I0	operand1
(not used)	operand2
(not used)	operand3

Pointer $P16 initially is set to point to segment #0, offset 12. The PTR_MOVE command subtracts the value of $P2 $I0 (2) from the initial offset of 12 to get a new offset of 10 (12 - 2 = 10), so now $P16 points to segment #0, offset 10.

Example 3

```
POINTER $P16 0 12
PTR_VALUE $P2 $I0 2
PTR_MOVE $P16 MULT $P2 $I0 SUB 4
```

In this example:

$P16	pointer_name
MULT	operation
$P2 $I0	operand1
SUB	operand2
4	operand3

This statement multiplies the integer at $P2 $I0 (2) by the multiplier of 4 to get 8 (2 * 4 = 8), and then subtracts that value from the current offset of $P16 (12) to get the new offset of $P16 of 4 (12 - 8 = 4).

4.55 PTR_VALUE

The PTR_VALUE command changes data at the specified pointer location as opposed to PTR_EQUAL and PTR_MOVE which change the specified pointer location (see applicable reference pages for more information). The PTR_VALUE command is used to change the data addressed by a pointer variable to a new value. Integer, real, byte, short, or even ASCII data can be written to the specified pointer location.

SYNTAX

```
PTR_VALUE pointer_name offset value
```

where:

pointer_name	$P pointer name. Valid range is $P1 through $P99.
offset	$Bn, $In, $Rn, $Sn, $AnXi
value	One of the following: ▪ Point name and two-character record field. ▪ $ Pointer name and $ offset ▪ Integer value. ▪ Real value. ▪ String enclosed in single/double quotes

RULES

1. Relative integer and real values can be used to increase or decrease the current value at the pointer location. That is, the value may be specified as [5], [123], [344.567], [-25], [-10], and so forth. In the latter cases, the value inside the brackets will be added to the current value addressed by the pointer. Thus, if the current value at the specified pointer location is 25 and the above relative values are used in a PTR_VALUE command, then resultant values would be 30[25+5], 148[25+123], 369.567[25+344.567], 0[25-25], 15[25-10], and so forth.
2. For the pointer_offset and $ offset part of the specified value, $In, $Bn, $sn, $Rn, and $AnXi are supported. It is your responsibility to supply the correct value for n for the given offset (for example, n must be a multiple of 4 for $r offsets).
3. If you wish to set/change ASCII data using this command, you must specify the pointer_offset as a $AnXi offset, where “n” is the offset to begin writing the ASCII data at and “i” is the maximum number of characters that can be copied to this location [1<=i<=80].

   If you specify the pointer_offset as such, then the specified value must also be ASCII data. It can be either a process point name with an ASCII record field (for example: A100 ED, A100 EU, A100 PN), a $pointer name and an ASCII offset (for example: $P1 $A10X16), or a literal ASCII string (for example: “string data”, “any quoted string”). If the pointer_offset is an ASCII offset, then the value must be an ASCII value. Conversely, if the pointer_offset is not an ASCII offset (for example, $B0, $I4, $R12), then the value cannot be an ASCII value.
4. The PTR_VALUE command can be used in the diagram, background, foreground, and trigger sections of a diagram.

EXAMPLES

Example 1

```
PTR_VALUE $P15 $I0 A100 1W
```

In this example:

$P15	pointer_name
$I0	offset
A100 1W	value

This statement stores the value of A100 1W in the integer at $P15 $I0.

Example 2

```
PTR_VALUE $P15 $I0 [5]
```

In this example:

$P15	pointer_name
$I0	offset
[5]	value

This statement adds 5 to the current integer value at $P15 $I0.

4.56 RECTANGLE

The RECTANGLE command displays a rectangle on the diagram. The rectangle has an associated line pattern and fill pattern.

SYNTAX

```
RECTANGLE x y w h line_width line_pat [conditional] fill_pat
[conditional]
```

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 line width array index.
line_pat	Line pattern name.
[conditional]	Optional conditional for line pattern.
fill_pat	Fill pattern name.
[conditional]	Optional conditional for fill pattern.

RULES

1. The x and y coordinates can be relative.
2. The origin is the upper left corner of the rectangle.
3. The line pattern and fill pattern names may be followed by conditional expressions which will determine which line/fill pattern to display at runtime. The conditional expressions are optional.
4. The RECTANGLE command may be used in the background, foreground, or trigger sections of a diagram.

EXAMPLES

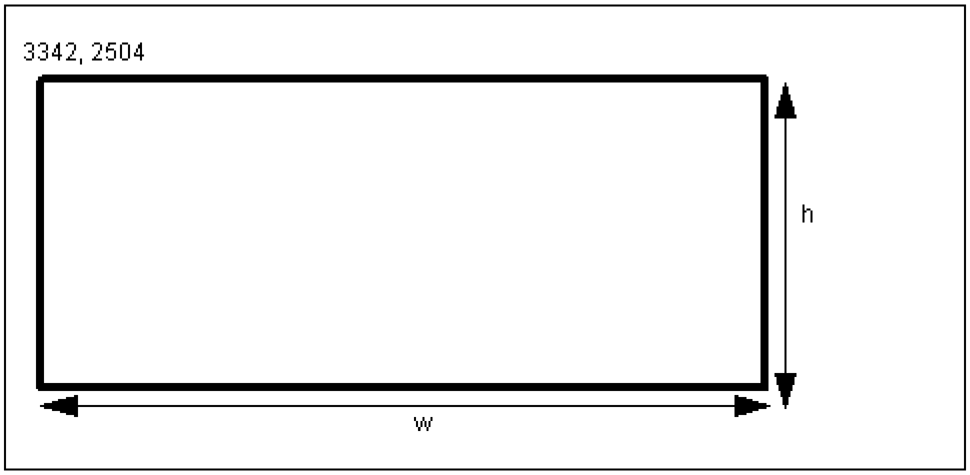
Example 1

```
RECTANGLE 3342 2504 6534 8141 3 solid unfilled
```

In this example:

3342	x coordinate
2504	y coordinate
6534	w
8141	h
3	line_width
dot_dash	line_pat
unfilled	fill_pat

This statement defines a rectangle located at 3342, 2504. The width and height of the rectangle is 6534, 8141. The line width is 3 (which corresponds to a pixel line width = 4 in the line width table) and the line pattern is solid. The rectangle is unfilled.



Example 2 (Conditional)

```
RECTANGLE 10898 8378 4119 2471 8 dotted solid { (AI101 BV <= 50.5)
asterisks (D200 1W = 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
[conditional]	{(AI101 BV <= 50.5) asterisks (D200 1W = 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 (which corresponds to a pixel line width = 14 in the line width table) and the line pattern is dotted.

A fill pattern conditional is specified. If AI101 BV <= 50.5, the fill pattern will be asterisks. If the first conditional evaluates to false, the second conditional will be considered. If D200 1W = RESET, the fill pattern will be back_slash. If the second conditional evaluates to false, the fill pattern will be solid (default).

4.57 RUN_PROGRAMS

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 without having to manually poke anything on the graphic. 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.

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 (may be specified as an integer OR by a $Pn $Im pointer/offset pair, where m is a multiple of 4).
diag_1, diag_2, ...diag_n	Diagram associated with application program (may be specified as an integer OR by a $Pn $Im pointer/offset pair, where m is a multiple of 4). 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 (varies per application program).
arguments	▪ Arguments passed to the associated application program. Valid arguments consist of the following: ▪ text strings (130 characters or less) ▪ integers ▪ reals ▪ point name and record field (example: A100 AV) ▪ pointer and offset (example: $P1 $R0) ▪ set variable (example: SET1) ▪ macro input parameters (example: “$Tn”, $CONSTn, $SETn, $Dn, $STATUSn)

RULES

1. It is recommended that the DIAG_DISP command be used to a display a diagram instead of using RUN_PROGRAMS with application program #1 (CHGDIAG).
2. 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 AV A004Z004 AV 1
9 2
```

In this example:

2	programs
117	prog_1
8104	diag_1
5	#args_1
0, 79, 27, 0, 0	arguments_1
6	prog_2
0	diag_2
5	#args_2
A004Z004 AV, A004Z004 AV, 1, 9, 2	arguments_2

In this example, two application programs are being run. The first program is 117. Diagram 8104 will be displayed for this program. There are five arguments passed to program 117: 0, 79, 27, 0, 0.

The second program is 6. No diagram is required for program 6 (since diag_2 = 0). It also requires five arguments. The first two arguments are point/record fields (A004Z004 AV and A004Z004 AV) and the next three are integers: 1, 9, and 2.

4.58 SETVAL

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 and window diagrams. Set1 in the main screen 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 you want to pass the set number into the macro instead of hard-coding the set number, you enter 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:

10	set
4	value

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.

4.59 SHAPE

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 *Ovation Graphics Builder User Guide*.)

SYNTAX

```
SHAPE x y w h shape_name [conditional] rotation inversion
```

where:

x	x coordinate of position at which to place the shape's origin. Valid range = 0 through 16,383.
y	y coordinate of position at which to place the shape's origin. Valid range = 0 through 16,383.
w	Width of outlining rectangle into which to scale the shape. Valid range = 1 through 16,383.
h	Height of outlining rectangle into which to scale the shape. Valid range = 1 through 16,383.
shape_name	Shape name from shape library.
[conditional]	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 in the shape library. (See *Ovation Graphics Builder User Guide*.) The origin can be any point within the shape's outlining rectangle. The origin point is not required to be on the shape, but it must be within the shape's outlining rectangle.
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 may be followed by a conditional expression which will determine which shape to display at runtime. The conditional expressions are optional.
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 your 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:

5291	x
1794	y
4336	w
8962	h
VALVE	shape name
90	rotation
RTL	inversion

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\ LL 1 1 2 VALVE TANK 0
NONE
```

In this example:

11214	x
4119	y
3071	w
6701	h
PUMP	shape name
(CASE) \T374H\ LL 1 1 2 VALVE TANK	[conditional]
0	rotation
NONE	inversion

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 LL as described below:

- If \T37H\ LL < 2, the shape will be PUMP.
- If \T37H\ LL >= 2 and if \T374H\ LL < 3, the shape will be VALVE.
- If \T374H\ LL >= 3 and if \T374H\ LL < 4, the shape will be TANK.
- If \T374H\ LL >= 4, the shape will be PUMP.

4.60 TEXT

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 *Ovation Developer Studio User Guide*.)

The format of a line in this file is: index string, where index = 1-100, and string is 80 characters or less, enclosed in single or double quotes. 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] orientation font_type font_size
```

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]	Optional conditional for string
orientation	Orientation. The choices are: ▪ HORZ (horizontal) ▪ VERT (vertical)

font_type	Font types. The choices are: ▪ BITMAP - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ BITMAP_OVER - non-scalable; character and background drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**. ▪ VECTOR_OVER - scalable; character and background drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. ▪ **bitmap_font_number** - Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through 8. ▪ **charw** - Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. ▪ **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. ▪ **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.

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 and bitmap_over text).
   - Upper left corner of the text outlining rectangle (for vector and vector_over text).
3. You may select the size of the text characters used to display the string.
4. You may select the orientation of the characters (horizontal or vertical).
5. You may display the text in either a vector or bitmap font.
6. The text string parameter may be followed by a conditional expression which will determine which string to display at runtime. The conditional expressions are optional. You 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. 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.
8. Text strings are limited to 80 characters.
9. 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 1
```

In this example:

7912	x
6000	y
“Alert Cue”	“string”
HORZ	orientation
VECTOR	font_type
161	charw
336	charh
1	lw

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 1.

Example 2 (Conditional)

```
TEXT 11063 2990 “GOOD” (QUALITY) \T45J1\ “FAIR” “POOR” “BAD” “TOUT”
VERT VECTOR_OVER 990 957 1
```

In this example:

x	11063
y	2990
“string” [conditional]	“GOOD” (QUALITY) \T45J1\ “FAIR” “POOR” “BAD” “TOUT”
orientation	VERT
font_type	VECTOR_OVER
charw	990
charh	957
lw	1

This statement defines the text “GOOD” to be positioned at 11063, 2990. The string displays vertically in vector text, with the background of the characters also drawn. The width and height of each character in the text is 990, 957 respectively. The line width is 1.

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.”
- If \T454J1\ has fair quality, the text will read “FAIR.”
- If \T454J1\ has poor quality, the text will read “POOR.”
- If \T454J1\ has bad quality, the text will read “BAD.”
- If \T454J1\ has timed out quality, the text will read “TOUT.”

Example 3 (text group)

```
TEXT 7912 6000 “$C5X20” HORZ VECTOR 161 336 1
```

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:

7912	x
6000	y
“5th string in the te”	“string”
HORZ	orientation
VECTOR	font_type
161	charw
336	charh
1	lw

This statement displays the first 20 characters of the fifth string in the active text group file at 7912, 6000. The string displays horizontally in vector text. The width and height of each character of the text is 161, 336. The line width is 1. Note that “$C5X20” resolves to “5th string in the te” as only the first 20 characters of the fifth string are used as specified by the “20” in the text group string notation.

4.61 TIME

The TIME command displays the current time on the diagram in the specified format.

SYNTAX

```
TIME x y format font_type font_size
```

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 - non-scalable; character background not drawn. Followed by **bitmap_font_number** for **font_size**. ▪ BITMAP_OVER - non-scalable; character and background drawn. Followed by **bitmap_font_number** for **font_size**. ▪ VECTOR - scalable; character background not drawn. Followed by **charw, charh, lw** for **font_size**. ▪ VECTOR_OVER - scalable; character and background drawn. Followed by **charw, charh, lw** for **font_size**.
font_size	Number of parameters used varies depending on the font_type selected. ▪ **bitmap_font_number -** Standard bitmap font number (used with bitmap and bitmap_over font types). Valid range = 1 through 8. ▪ **charw -** Width of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. ▪ **charh** - Height of a vector character cell in pixels (used with vector and vector_over font types). Valid range = 3 through 16,383. ▪ **lw** - Line width. Valid range is 1 - 16 for Ovation Vector font. For all other vector fonts, 1 must be entered since only Ovation Vector font supports line width.

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 and bitmap_over text).
    - Upper left corner of the text outlining rectangle (for vector and vector_over text).
3. You 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. However, it should be in the foreground so that it updates as the time changes while the diagram is running.

EXAMPLES

Example 1

```
TIME 1720 600 1 VECTOR 161 336 1
```

In this example:

1720	x
600	y
1	format
VECTOR	font_type
161	charw
336	charh
1	lw

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:

1720	x
1200	y
2	format
BITMAP	font_type
2	bitmap_font_number

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.

4.62 TREND

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_limit high_limit
#values interval NOSCALE
```

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 names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names.
rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth).

low_limit	Low limit used to scale sampled values. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
high_limit	High limit used to scale sampled values. Valid values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
#values	Number of sampled values to trend. Valid range = 1 through 600.
interval	Number of seconds between successive sampled values. Valid range = 1 through 10,000.
NOSCALE	Placeholder. This parameter is no longer used. Scales are never displayed. Any setting will be saved as NOSCALE in the gcode.

RULES

1. The x and y coordinates can be relative.
2. The origin of a trend depends on the orientation parameter. The origin will be one of the following:
   - Upper left corner of the trend outlining rectangle for VERT trend.
   - Lower right corner of the trend outlining rectangle for HORZ trend.
3. The orientation of the trend is defined by the time axis. If the time is displayed along the x-axis, the trend is a horizontal trend. If time is displayed along the y-axis, the trend is a vertical trend.
4. You define the number of values to display and the time interval between successive values.
5. 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 at runtime. The color defined in the active COLOR command is used to display the trend for good quality at runtime. 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 at runtime.
6. 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.

7. Scale values are never automatically displayed (no matter what the scale parameter is set to). If you want to display the scale values, they must be displayed independently using the TEXT command or PROCESS_PT command.
8. 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 NOSCALE
```

In this example:

4810	x
8372	y
3818	w
6483	h
HORZ	orientation
A100	pt_name
AV	rec_fld
10	low
100	high
5	#values
15	interval
NOSCALE (placeholder)	NOSCALE

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 NOT display on the trend (scales never display, no matter what you set this value to in the source).

4.63 TRIGGER

The TRIGGER command defines a trigger section of the diagram. Trigger sections of the diagram are activated on demand by a graphics application program or by the user-specified TRIG_ON command.

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 sections of the graphic are only executed on demand at runtime. They are not executed when the graphic is initially displayed at runtime, nor when the graphic updates at runtime (unless there is a TRIG_ON or RUN_PROGRAMS command in the foreground of the graphic which causes the trigger to execute).
3. GBNT displays all trigger sections of the graphic when the graphic is displayed and when the GBNT window refreshes. There is no way to turn on/off a particular trigger section in GBNT. GBNT does nothing when it encounters a TRIG_ON command or a RUN_PROGRAMS command that calls an application program that executes a trigger. GBNT does not execute trigger sections on demand - instead it executes the trigger sections all of the time.
4. The trigger section of the diagram ends when another TRIGGER command or a BACKGROUND, FOREGROUND, or KEYBOARD command is encountered.

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:

255	n

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.

4.64 TRIG_ON

The TRIG_ON command is used to activate a trigger region of a diagram (that is, to cause the commands in the trigger region to be executed).

SYNTAX

```
TRIG_ON n [conditional]
```

where:

n	Trigger region number. Valid range = 1 through 255
[conditional]	Optional conditional for n

RULES

1. In this command, the trigger number may be followed by a conditional expression which will determine which trigger to execute at runtime. The conditional expression is optional.
2. The TRIG_ON command may be used in the background, foreground, or trigger sections of a diagram.

EXAMPLE

```
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
FOREGROUND
TRIG_ON 1 (A100 AV < 0.0) 2
```

In this example:

1	n
(A100 AV < 0.0) 2	[conditional]

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.

Note that when/if the value of A100 AV changes at runtime and causes the other trigger to execute, nothing is erased. This is **not** like a conditional shape where the current shape is erased before the new shape is displayed. The current trigger data is not erased before the new trigger data is displayed.

4.65 XY_PLOT

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. Typically, a dot is plotted at their point of intersection. 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 NOSCALE 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	Process point names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names to plot along x-axis.
x_rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth) to plot along x-axis.
x_low x_high	Low limit value to use to scale point on x axis values. High limit value to use to scale point on x axis values. Valid limit values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
y_pt_name	Process point names, dummy point names, $ pointers ($P, $G, $D, $W, $H, or $O) or OPC point names to plot along x-axis.
y_rec_fld	Two-character record field name (1W, AV, ID, and so forth) or $offset identifier ($B0, $I4, $R0, and so forth) to plot along the x-axis.

y_low y_high	Low limit value to use to scale point on y axis values. High limit value to use to scale point on y axis values. Valid limit values are: ▪ Integer/real numbers (for example: 123, -100, 50.5) ▪ Point/Pointer name and record field pair (for example: \A100\ LL, $D1 AV, $G2 HL) ▪ $ Pointer identifier and $ Offset identifier (for example: $P1 $R0, $D1 $I0) ▪ OPC point name (no record field)
NOSCALE	Placeholder. This parameter is no longer used. Scales are never displayed. Any setting will be saved as NOSCALE in the gcode.
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. You specify the low and high scale values for each axis.
4. In this command, the 12 parameters needed to specify that a shape is to be used to plot the intersection of the xy plot are optional. If these parameters are omitted, a dot will be used at runtime to plot the intersection of the xy plot. If you wish to plot a shape instead of a dot, he must enter ALL of the 12 parameters.
5. The colors defined in the DEF_QUAL command for fair, poor, bad, and timed-out quality are also used to display the xy_plot at runtime 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.
6. 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.
7. 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.
8. Scale values are never automatically displayed (no matter what the scale parameter is set to). If you want to display the scale values, they must be displayed independently using the TEXT or PROCESS_PT command.
9. 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 NOSCALE 3
```

In this example:

9686	x
12558	y
4263	w
6743	h
A2008	x_pt_name
AV	x_rec_fld
-10	x_low
10	x_high
A3003	y_pt_name

AV	y_rec_fld
0	y_low
50	y_high
NOSCALE (placeholder)	NOSCALE
3	update_rate

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 (no matter what you enter, scales always default to NOSCALE). 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:

9686	x
12558	y
4263	w
6743	h
A2008	x_pt_name
AV	x_rec_fld
-10	x_low
10	x_high
A3003	y_pt_name
AV	y_rec_fld
0	y_low
50	y_high
NOSCALE (placeholder)	NOSCALE
3	update_rate
ptr	current_shape
500	shapew1
300	shapeh1

-90	rot1
RTL	inv1
hollow_ptr	history_shape
250	shapew2
150	shapeh2
-90	rot2
RTL	inv2
20	n

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 5

APPLICATION PROGRAMS

In This Section

5.1 INTRODUCTION TO APPLICATION PROGRAMS

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 *Ovation Operator Station User Guide*.)

Being able to successfully use the application programs requires that you 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.

For these reasons, you must have a good understanding of the control logic and drop functions being implemented in a system before building custom control diagrams. (See *Ovation Record Types Reference Manual* and *Ovation Algorithms Reference Manual*.)

5.1.1 SECURITY

The ability to access secure Ovation functions and data is dictated by the security session to which the software belongs. (See *Ovation Developer Studio User Guide*.) A session is defined by the currently logged-in user, the roles to which that user belongs, the particular drop on which the software is operating, and the applicable display (that is, the drop console or a specified remote drop).

For example, MMI applications displayed on the drop console(s) have access to Ovation system functions and data as dictated by the currently logged-in user and the particular drop. An Ovation user who remotely logs into the drop (as well as remotely displayed MMI applications) will have a separate security session with a potentially different level of access to Ovation system functions and data.

Certain application programs are affected by security functions. These functions are described below:

- **Control Function Enabled** - in order for you to perform control functions for a particular algorithm (such as Auto, Manual, Raise, Lower, and so forth), the Control Functions option must be enabled. Refer to the System Functions tab of the Security item on the Ovation Developer Studio.
- **Tuning Function Enabled** - in order for you to perform tuning functions for a particular algorithm (such as change the value of a particular tuning variable), the Tuning Functions option must be enabled. Refer to the System Functions tab of the Security item on the Ovation Developer Studio.
- **Algorithm Point Access Enabled** - in order for you to perform control, tuning, or alarm acknowledge functions on a particular algorithm or point, the point must have Point Access Security Group enabled. Refer to the Point Access tab of the Security item on the Ovation Developer Studio
- **Alarm Acknowledge Enabled** - In order for you to perform an Alarm Acknowledgement for a particular point, the Alarm Acknowledge Function option must be enabled. Refer to the System Functions tab of the Security item on the Ovation Developer Studio.

5.1.2 Application Program Listing

The following table provides a quick reference list to the application programs discussed in this section. In addition, the table shows (by the use of "•") which programs can be locked out of the various Security functions.

Note: *In the table, if an application program does not have any of the security functions marked (that is, there are no "x" in the row), then it is not affected by any of the Security lockout parameters.*

Application Program Listing

Program Name	Program Number	Control Function Enable	Tuning Function Enable	Algorithm Point Access Enable	Alarm Acknowledge Enabled
CHGDIAG	1				
CHGDIAG_WX_GROUP	2				
CNTRL_POKE	6			x	
PAGE_TOP	13				
RECALL_FORWARD	14				
RECALL_BACKWARD	15				
PAGE_RIGHT	16				
PAGE_UP	17				
PAGE_LEFT	18				
PAGE_DOWN	19				
DIGITON	28	x		x	
DIGITOFF	29	x		x	
UPSET	30	x		x	
DOWNSET	31	x		x	
MANUAL	32	x		x	
AUTO	33	x		x	
RAISEOUT	34	x		x	
LOWEROUT	35	x		x	
DISP_EFDATA	66				
SEND_GENMSG	67				

Program Name	**Program Number**	**Control Function Enable**	**Tuning Function Enable**	**Algorithm Point Access Enable**	**Alarm Acknowledge Enabled**
SEND_CA	80		x		
CNTL_TUNE	95				
SEND_CA_CTL_LOCK	98	x			
NULL	99				
DDC_MODE	100	x		x	
SUPV_MODE	101	x		x	
CASC_MODE	102	x		x	
SEND_CA_EV	105		x		
SEND_CA_EV_CTL_LOCK	106	x			
SEND_CA_EV_BIT	108		x		
WINDOW_FUNC_KEY	117				
DISP_EFDATA	119				
XPID_DIGITAL	121	x		x	
EXEC_TRIGGER	122				
CNTRLBITS	124	x		x	
WINDOW_DELETE	125				
SUBWINDOW_DELETE	128				
POKE_EVENT	129				
CONT_RAISE_LOWER_VAL	130				
SEND_GENMSG_NETWORK	167				
SEND_CA_CTL_LOCK	180	x			
TREND_GROUP	201				
EXECUTE_PROCESS	202				
CLEAR_CONTROL	203				
RESET_CONTROL	204				
SEND_CA_EV_CTL_LOCK	205	x			
ACK_DROP_ALARM	210				x

Program Name	**Program Number**	**Control Function Enable**	**Tuning Function Enable**	**Algorithm Point Access Enable**	**Alarm Acknowledge Enabled**
CLEAR_DROP_FAULT	212				**x**
ALARM_ACKNOWLEDGE	214				**x**
XPID_DIGITAL_CTL_LOCK	221	**x**		**x**	
READ_ENTRY_FIELD	224				
WRITE_OPC_DATA	225				
SCADA_CMD	226			**x**	

5.1.3 REFERENCE PAGE FORMAT

The application programs are detailed in the following sections. A brief description of each program is given along with the necessary requirements and arguments. The reference pages are arranged in numerical order.

Note: *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.*

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.

5.2 CHGDIAG (1)

CHGDIAG displays a new diagram in the current process diagram window set. The diagram will appear in the main screen or window depending on the number of the diagram. No arguments are required for this application program. However, you must specify which diagram is to be displayed.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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.

EXAMPLE

```
POKE_FLD 800 15242 642 552 ON 7 1 1 2500 0
```

In this example:

800	x
15242	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
1	prog_num
2500	diag_num
0	num_of_args

When you click 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, any successive application program after this one in a poke 7 list will never execute.*

5.3 CHGDIAG_WX_GROUP (2)

CHGDIAG_WX_GROUP displays a new diagram and group in the specified window. The diagram will appear in the main screen or window depending on the number of the diagram. .

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args window_number group_number
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **2** 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, **2** will be entered on the command line.
window_number	Number of the window to display the requested diagram in. Valid range = 1 through 8.
group_number	Diagram group number. Valid range = 0 through 5,000.

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 12 1000 2 8 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
2	prog_num
1000	diag_num
2	num_of_args
8	window_number
0	group_number

When you click on this poke field (specified as type 7), program 2 (CHGDIAG_WX_GROUP) will execute displaying diagram 1000 in window 8.

5.4 CNTRL_POKE (6)

CNTRL_POKE is used to define the way control selections are made from an Operator Station process diagram. It defines the way the system IDs and algorithm records are interfaced to the Control Panel window. (See *Ovation Operator Station User Guide*.)

With CNTRL_POKE, up to 50 control items can be selected from a single diagram, which may be a main screen or window. By clicking on a poke field, you initiate 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, CNTRL_POKE clears previous trigger actions, and the new TRIGGER commands execute.

***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, you must reset the set value with a SETVAL command.*

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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	Application program number. For this application program, **6** will be entered in the command line.
diag_num	Diagram to display (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
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, and STOP.

point2	Algorithm point name and ID record field for programmable key functions 5 through 8, AUTO MODE, MANUAL MODE, RAISE OUTPUT, and 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 5 001-00123 ID 001-00124 ID 1 9
2
```

In this example:

965	x
14447	y
869	w
1389	h
ON	state
7	poke_type
1	num_of_progs
6	prog_num
0	diag_num
5	num_of_args
001-00123 ID	point1
001-00124 ID	point2
1	trig_num
9	set_num
2	setval

When you click 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 the ***TRIGGER keyword*** (see page 215) for more information. The 001-00123 and 001-00124 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 5 001-00123 ID
001-00124 ID 1 9 2 122 0 2 1 4
```

In this example:

965	x
14447	y
869	w
1389	h
ON	state
7	poke_type
2	num_of_progs
6	prog_num
0	diag_num
5	num_of_args
001-00123 ID	point1
001-00124 ID	point2
1	trig_num
9	set_num
2	setval
122	prog_num
0	diag_num
2	num_of_args
1	si
4	trig_num

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.

5.5 PAGE_TOP (13)

PAGE_TOP displays the top diagram.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **13** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 1 13 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
13	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 13 (PAGE_TOP) executes. PAGE_TOP takes you to the top diagram.

5.6 RECALL_FORWARD (14)

RECALL_FORWARD is used to page forward in the command paging sequence. This function is only enabled after a recall backward command (see RECALL_BACKWARD (15) application program).

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **14** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

Example

```
POKE_FLD 722 13256 642 552 ON 7 1 14 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
14	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 14 (RECALL_FORWARD) executes. RECALL_FORWARD takes you to the diagram one forward in the paging sequence.

5.7 RECALL_BACKWARD (15)

RECALL_BACKWARD displays the previously requested diagram.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **15** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 1 15 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
15	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 15 (RECALL_BACKWARD) executes. RECALL_BACKWARD takes you back to the previously requested diagram.

5.8 PAGE_RIGHT (16)

PAGE_RIGHT displays the diagram to the right of the current main diagram as defined by the PAGE command. See Section 4 for more information on the PAGE command.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **16** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 1 16 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
16	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 16 (PAGE_RIGHT) executes. PAGE_RIGHT takes you to the diagram to the right of the current diagram as defined by the PAGE command.

5.9 PAGE_UP (17)

PAGE_UP displays the diagram above the current main diagram as defined by the ***PAGE command*** (see page 161).

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **17** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 1 17 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
17	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 17 (PAGE_UP) executes. PAGE_UP takes you to the diagram above the current diagram as defined by the PAGE command.

5.10 PAGE_LEFT (18)

PAGE_LEFT displays the diagram to the left of the current main diagram as defined by the PAGE command. See Section 4 for more information on the PAGE command.

Function Lockout

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **18** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 1 18 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
18	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 18 (PAGE_LEFT) executes. PAGE_LEFT takes you to the diagram to the left of the current diagram as defined by the PAGE command.

5.11 PAGE_DOWN (19)

PAGE_DOWN displays the diagram below the current main diagram as defined by the PAGE command. See Section 4 for more information on the PAGE command.

Function Lockout

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **19** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 1 19 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
19	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 19 (PAGE_DOWN) executes. PAGE_DOWN takes you to the diagram below the current diagram as defined by the PAGE command.

5.12 DIGITON (28)

DIGITON activates the START/OPEN/TRIP function defined by a specific control algorithm. This program executes behind the START/OPEN/TRIP button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can only be used with the KEYBOARD and CRTML algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm to be used by DIGITON.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x		**x**	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **28** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLES

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1 9
2 28 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (KEYBOARD algorithm)	point 1
001-00123 ID	point 2
1	trig_num
9	set_num
2	set_val
28 (DIGITON)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the keyboard algorithm. Program 28 (DIGITON) will then execute setting the START/OPEN/TRIP output of the algorithm.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 28 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (KEYBOARD algorithm)	point 1
001-00123 ID (KEYBOARD algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val

1	num_of_progs
28 (DIGITON)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the keyboard algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 28 (DIGITON) will execute setting the START/OPEN/TRIP output of the algorithm.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.13 DIGITOFF (29)

DIGITOFF activates the STOP/CLOSE/RESET function defined by a specific control algorithm. This program executes behind the STOP/CLOSE/RESET button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can only be used with the KEYBOARD and CRTML algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm to be used by DIGITOFF.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x		**x**	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **29** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1 9
2 29 0 0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (KEYBOARD algorithm)	point 1
001-00123 ID	point 2
1	trig_num
9	set_num
2	set_val
29 (DIGITOFF)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the keyboard algorithm. Program 29 (DIGITOFF) will then execute setting the STOP/CLOSE/RESET output of the algorithm.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 29 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (KEYBOARD algorithm)	point 1
001-00123 ID (KEYBOARD algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val

1	num_of_progs
29 (DIGITOFF)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the keyboard algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 29 (DIGITOFF) will then execute setting the STOP/CLOSE/RESET output of the algorithm.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.14 UPSET (30)

UPSET raises the set point of a process variable as defined by a specific control algorithm. This program executes behind the Up arrow button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can be used with the KEYBOARD, MASTATION, SETPOINT, CRTML, XMA2, XML2, XPDSPV, XPID, XPIDOVD, XPIDSLI, and XPIDSPV algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm to be used by UPSET.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **30** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

Example

Example1

```
POKE_FLD 722 13256 642 552 ON 7 1 30 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
30	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 30 (UPSET) will execute raising the set point of the process variable.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00124 ID 001-00124 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 30 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type

1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00124 ID (SETPOINT algorithm)	point 1
001-00124 ID (SETPOINT algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val
1	num_of_progs
30 (UPSET)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the setpoint algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 30 (UPSET) will execute raising the set point of the process variable.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.15 DOWNSET (31)

DOWNSET lowers the set point of a process variable as defined by a specific control algorithm. This program executes behind the Down arrow button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can be used with the KEYBOARD, MASTATION, SETPOINT, CRTML, XMA2, XML2, XPDSPV, XPID, XPIDOVD, XPIDSLI, and XPIDSPV algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm to be used by DOWNSET.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **31** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 1 31 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
31	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 31 (DOWNSET) will execute lowering the set point of the process variable.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00124 ID 001-00124 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 31 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type

1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00124 ID (SETPOINT algorithm)	point 1
001-00124 ID (SETPOINT algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val
1	num_of_progs
31 (DOWNSET)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the setpoint algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 31 (DOWNSET) will execute lowering the set point of the process variable.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.16 MANUAL (32)

MANUAL places a specific control algorithm into MANUAL mode. This program executes behind the MANUAL button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can be used with the KEYBOARD, MASTATION, CRTMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, and XPIDSPV algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm used by MANUAL.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x		**x**	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **32** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1 9
2 32 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (MASTATION algorithm)	point 1
001-00123 ID (MASTATION algorithm)	point 2
1	trig_num
9	set_num
2	set_val
32 (MANUAL)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the MASTATION algorithm. Program 32 (MANUAL) will then execute placing the algorithm in MANUAL mode.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 32 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (MASTATION algorithm)	point 1
001-00123 ID (MASTATION algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val

1	num_of_progs
32 (MANUAL)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the MASTATION algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 32 (MANUAL) will then execute placing the algorithm in MANUAL mode.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.17 AUTO (33)

AUTO places a specific control algorithm into AUTO mode. This program executes behind the AUTO button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can be used with the KEYBOARD, MASTATION, CRTMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, and XPIDSPV algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm to be used by AUTO.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **33** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1 9
2 33 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123ID (MASTATION algorithm)	point 1
001-00123 ID (MASTATION algorithm)	point 2
1	trig_num
9	set_num
2	set_val
33 (AUTO)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the MASTATION algorithm. Program 33 (AUTO) will then execute placing the algorithm in AUTO mode.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 33 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (MASTATION algorithm)	point 1
001-00123 ID (MASTATION algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val

1	num_of_progs
33 (AUTO)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the MASTATION algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 33 (AUTO) will then execute placing the algorithm in AUTO mode.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.18 RAISEOUT (34)

RAISEOUT raises a specific control algorithm's output value. This program executes behind the Output Raise button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can be used with the KEYBOARD, MASTATION, CRTMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, and XPIDSPV algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm to be used by RAISEOUT.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **34** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 1 34 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
34	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 34 (RAISEOUT) will execute raising the output of the selected algorithm.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 34 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type

1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (MASTATION algorithm)	point 1
001-00123 ID (MASTATION algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val
1	num_of_progs
34 (RAISEOUT)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the MASTATION algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 34 (RAISEOUT) will execute raising the output of the selected algorithm.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.19 LOWEROUT (35)

LOWEROUT lowers a specific control algorithm's output value. This program executes behind the Output Lower button on the Control Panel window. (See *Ovation Operator Station User Guide*.)

This program can be used with the KEYBOARD, MASTATION, CRTMA, XMA2, XPDSPV, XPID, XPIDOVD, XPIDSLI, and XPIDSPV algorithms. (See *Standard Control Algorithm User Guide*.)

Use the CNTRL_POKE (6) program to activate the algorithm to be used by LOWEROUT.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **35** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 1 35 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
35	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 35 (LOWEROUT) will execute lowering the set point of the process variable.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
POKE_FLD 419 10111 412 322 ON 23 1 2 1 35 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type

1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (MASTATION algorithm)	point 1
001-00123 ID (MASTATION algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val
1	num_of_progs
35 (LOWEROUT)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the MASTATION algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 35 (LOWEROUT) will execute lowering the set point of the process variable.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.20 DISP_EFDATA (66)

DISP_EFDATA displays data in entry fields in either a main screen or subscreen. 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*** (see page 168), or a constant number or text string.

Note: *To write to a window, use program 119 and in place of argument "main," substitute "window."*

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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:

x. y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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.
main	Number of entry fields to write to in the main screen (or window if program 119 is used).
sub	Number of entry fields to write to on the subscreen.
ef	Entry field buffer number. This number should correspond to the entry field buffer number in the ENTRY_FLD 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.

Repeat “ef” and “data” fields for the number of entry fields specified in “main” then repeat “ef” and “data” for the number of entry fields specified in “sub”.

EXAMPLE

```
POKE_FLD 483 14400 546 904 ON 7 1 66 0 8 2 1 16 “ENTER ALARM STATUS”
17 A100 AV 2 1234
```

In this example:

483	x
14400	y
546	w
904	h
ON	state
7	poke_type
1	num_of_progs
66	prog_num
0	diag_num
8	num_of_args
2	main
1	sub
16	ef
“ENTER ALARM STATUS”	data
17	ef
A100 AV	data
2	ef
1234	data

When you click on the poke field, (specified as type 7), program 66 (DISP_EFDATA) executes. No new diagram will be displayed. Eight arguments follow. The arguments indicate that data will be entered into two entry field on the main screen. The first data item will be written to entry field number 16 on the main screen. The data is a fixed text string, “ENTER ALARM STATUS”. The second data item will be written to entry field number 17 on the main screen. The data is the value of the point A100. The third data item will be written to entry field number 2 on the subscreen. The data is a fixed constant 1234.

5.21 SEND_GENMSG (67)

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 window diagram and send data to another drop for processing. You are responsible for the proper formatting of the block data.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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 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 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 or window).
drpef1	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.
si1	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 7 = Window

id1	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.
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.
Note:1 *The drpef, si, and id arguments are used together, as shown in the following table.*	

Drpef, si, and id arguments.

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 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.

Note: *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.*

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 7 0 1 1 0
3000 0 0 165 0 1 0 30 2 2 2
```

In this example:

483	x
14400	y
546	w
904	h
ON	state
7	poke_type
1	num_of_progs
67	prog_num
0	diag_num
20	num_of_args
0	blk
3	total
1	sia

1	a
7	sib
0	b
1	drop
1	size
0	seg
3000	tdiag
0	drpef
0	si
165	id
0 1 *message type (bid a prgm) 0 30 *prgm 30	msgtype
2	ef
2	data
2 *send real number	dec

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 window (sib = 7, 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 window 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 window entry fields.

            •

            •

            •

ef #        data        dec

1           1                  *   read an integer from main screen entry field 1.

2           2           2      *   read a real from window entry field 2 (two decimal
                                   places

0           $S3         ID     *   read point name from window 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).

5.22 SEND_CA (80)

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 or window diagram.

It is recommended that only users who are experienced with graphic diagrams and are very familiar with algorithms, record fields, and their functions, use this program.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
	x		
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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 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 7 = Window
b	Number of entry fields to read from second diagram (defined by “sib” argument).
sysef1	Entry field buffer number containing the point name or ID of the point to be changed, or zero (0) if sending a constant.
si1	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. 7 = Entry field in window.
sysid1	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.
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. ▪ 4 = change sent to data record field - check record field for acknowledge. Limit Checking. Enter low limit and high limit (see below).

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 the 1W 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, TV, BV)
const	Constant data. When a constant is not sent, leave this argument blank (do not enter a zero instead of a blank).
low_limit	Only used when cmd = 4. Used for range checking. Low and high limits must be integer values.
high_limit	
Note:1 The sysef, si, and sysid arguments are used together, as shown in the following table.	

Sysef, si, and sysid arguments

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 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.

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.

When changing values in record fields, determine which record fields can be changed and the proper values to enter into these fields. (See *Ovation Record Types Reference Manual*.)

In some cases, algorithm points can receive change attribute messages. This occurs during algorithm tuning. You must determine which record fields need to be set to accomplish a specific tuning task. (See *“Ovation Algorithms Reference Manual*.)

EXAMPLES

Example 1

```
POKE_FLD 2452 14465 869 898 ON 7 1 80 0 19 1 3 1 2 7 0 0 0 $P90 $I0
1 1 $O1 B0
1 2 $O1 B1
2 0 $O1 ZY 0
```

In this example:

2452	x
14465	y
869	w
898	h
ON	state
7	poke_type
1	num_of_progs
80	prog_num
0	diag_num
19	num_of_args
1	blk
3	tot
1	sia
2	a
7	sib
0	b
0	sysef

0	si
$P90 $I0	sysid

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 ZY
	const	=	0 (sending a constant)

When you click 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 three 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 window (sib = 7, 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 ZY. Zero is the constant (const).

Example 2

```
POKE_FLD 2540 3597 1873 547 ON 7 1
80 0 28 1 4 1 1 7 0 0 0 \AI212S01\ ID
0 11 1 0 $O1 ZY 2
1 1 $O1 EV
0 10 1 0 $O1 ZY 9
2 0 $O1 ZY 1
```

In this example,

2540	x
3597	y
1873	w
547	h
ON	state
7	poke_type
1	num_of_progs
80 (SEND_CA_EV)	prog_num
0	diag_num
28	num_of_args
1 - ignore blank entry field	blk
4 - total # of items to be sent in message	tot
1 - screen index - main screen	sia
1 - number of entry fields on main screen to read	a
7 - screen index - window	sib

0 - number of entry fields on window to read	b
0 - sending a constant	sysef
0 - decode the system ID rather than reading it from an entry field	si
\AI212S01\ ID	sysid

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 ZY - 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 1W record field - current value = entered value
	bitval =	1 - wait for bit 10 to equal a “1”
	ef =	0 - entry field #
	off/typ =	$O1 ZY - 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 ZY - offset pointer / CM record field
	const =	1 - set on scan

When you click 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 four 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.

5.23 CNTL_TUNE (95)

CNTL_TUNE allows tuning of M/A station diagrams or other diagrams that include standard Emerson tuning diagrams. First, select 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.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args tune_id
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
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 RR, $G5 ID, $H90 RR, or $P90 $I0.

With this program, you 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 0 1 001-00123 ID
```

In this example:

965	x
14465	y
869	w
1389	h
ON	state
7	poke_type
1	num_of_progs
95	prog_num
0	diag_num
1	num_of_args
001-00123 ID	tune_id

In this example, the poke field calls one application program, number 95 (CNTL_TUNE). The point 001-00123 ID will be tuned. Based on the algorithm selected for tuning (001-00123 ID in this example), CNTL_TUNE displays the appropriate tuning screen(s).

5.24 SEND_CA_CTL_LOCK (98)

Refer to SEND_CA_CTL_LOCK (180) for information on this application program.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x			
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

5.25 NULL (99)

The NULL application program is used in special circumstances when you want no action to occur.

Function Lockout

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **99** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called).
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 800 15242 642 552 ON 7 1 99 0 0
```

In this example:

800	x
15242	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
99	prog_num
0	diag_num
0	num_of_args

When you click on this poke field (specified as type 7, program), program 99 (NULL) executes. Nothing will occur.

5.26 DDC_MODE (100)

DDC_MODE places a control algorithm into DDC mode. (See *Standard Control Algorithms User Guide*.)

This program can only be used with the XPIDOVD algorithm.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by DDC_MODE.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **100** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 2 6 0 5 001-00124 ID 001-00124 ID 1 9
2 100 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00124 ID (XPIDOVD algorithm)	point 1
001-00124 ID (XPIDOVD algorithm)	point 2
1	trig_num
9	set_num
2	set_val
100 (DDC_MODE)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the XPIDOVD algorithm. Program 100 (DDC_MODE) will then execute placing the algorithm in DDC mode.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00124 ID 001-00124 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 100 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00124 ID (XPIDOVD algorithm)	point 1
001-00124 ID (XPIDOVD algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val

1	num_of_progs
100 (DDC_MODE)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the XPIDOVD algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 100 (DDC_MODE) will then execute placing the algorithm in DDC mode.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.27 SUPV_MODE (101)

SUPV_MODE places a control algorithm into supervisory mode. (See *Standard Control Algorithms User Guide*.)

This program can only be used with the XPDSPV, XPIDSLI, and XPIDSPV algorithms.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by SUPV_MODE.

Function Lockout

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

Syntax

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **101** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1 9
2 101 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (XPDSPV algorithm)	point 1
001-00123 ID (XPDSPV algorithm)	point 2
1	trig_num
9	set_num
2	set_val
101 (SUPV_MODE)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the XPDSPV algorithm. Program 101 (SUPV_MODE) will then execute placing the algorithm in supervisory mode.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 101 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (XPDSPV algorithm)	point 1
001-00123 ID (XPDSPV algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val

1	num_of_progs
101 (SUPV_MODE)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the XPDSPV algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 101 (SUPV_MODE) will then execute placing the algorithm in supervisory mode.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.28 CASC_MODE (102)

CASC_MODE places a control algorithm into cascade mode. (See *Standard Control Algorithms User Guide*.)

This program can only be used with the XPDSPV, XPID, XPIDOVD, XPIDSLI, and XPIDSPV algorithms.

Use the CNTRL_POKE (6) program to activate the algorithm to be used by CASC_MODE.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x		**x**	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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, **102** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **0** will be entered on the command line.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1 9
2 102 0 0
```

In this example:

722	x
13526	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (XPDSPV algorithm)	point 1
001-00123 ID (XPDSPV algorithm)	point 2
1	trig_num
9	set_num
2	set_val
102 (CASC_MODE)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the XPDSPV algorithm. Program 102 (CASC_MODE) will then execute placing the algorithm in cascade mode.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 102 0 0
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (XPDSPV algorithm)	point 1
001-00123 ID (XPDSPV algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val

1	num_of_progs
102 (CASC_MODE)	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the XPDSPV algorithm, and putting the MA Station in control. By pressing the poke field specified as poke type 23, program 102 (CASC_MODE) will then execute placing the algorithm in cascade mode.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same those defined for Poke Type 23.*

5.29 SEND_CA_EV (105)

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 *eDB Historian User Guide*.)

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
	x		
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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 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 7 = Window
b	Number of entry fields to read from second diagram (defined by sib argument).
sysef1	Entry field buffer number containing the point name or ID of the point to be changed, or zero (0) if sending a constant.
si1	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. 7 = Entry field in window.
sysid1	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.
pnid	Point name or pointer for location of ASCII point name. Use this format: $Pn $AnXi 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 the 1W 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 *eDB Historian User Guide*.) ▪ 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 40 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, TV, BV)
const	Constant data. When a constant is not sent, leave this argument blank (do not enter a zero instead of a blank).
Note: *The sysef, si, and sysid arguments are used together, as shown in the following table.*	

Sysef, si, and sysid arguments

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 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.

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 7 0 0 0 $P90 $I0
$P93 PN
```

First Data Item	1 (cmd)	(bit)	(bitval)	1 (evflag)	'B0 FLD' (evdesc)	1 (ef)	$01 B0 (off/typ)	(const)
Second Data Item	1 (cmd)	(bit)	(bitval)	1 (evflag)	'B1 FLD' (evdesc)	2 (ef)	$01 B1 (off/typ)	(const)
Third Data Item	2 (cmd)	(bit)	(bitval)	(evflag)	(evdesc)	0 (ef)	$01 ZY (off/typ)	0 (const)

In this example:

2452	x
14465	y
869	w
898	h
ON	state
7	poke_type
1	num_of_progs
105	prog_num
0	diag_num
24	num_of_args
1	blk
3	tot
1	sia
2	a
7	sib
0	b
0	sysef
0	si
$P90 $I0	sysid
$P93 PN	pnid

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 ZY
	const	=	0 (sending a constant)

When you click 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 three 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 window (sib = 7, 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. The point name (pnid) is $P93 PN.

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 ZY. Zero is the constant (const).

5.30 SEND_CA_EV_CTL_LOCK (106)

Refer to SEND_CA_EV_CTL_LOCK (205) for information on this application program.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x			
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

5.31 SEND_CA_EV_BIT (108)

SEND_CA_EV_BIT is used to send a change attribute message for toggling a bit within a byte, word, or integer in a point from a control diagram. It also creates an Operator Event Message that is output to the printer or stored at the Historian. (See *eDB Historian User Guide*.)

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
	x		
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args blank sysef si sysid pnid cmd bit evflag evdesc off/typ
bitval
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, 108 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.
sysef1	Entry field buffer number containing the point name or ID of the point to be changed, or zero (0) if sending a constant.

si1	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. 7 = Entry field in window.
sysid1	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.
pnid	Point name or pointer for location of ASCII point name. Use this format: $Pn $AnXi 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).
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 the 1W 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.
evflag	Indicates whether or not an event message is to be sent to the printer at the Historian. (See *eDB Historian User Guide*.) ▪ 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 40 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.
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, TV, BV)
bitval	Value of "bit" should be changed to 0 or 1.
Note: The sysef, si, and sysid arguments are used together, as shown in the following table.	

Sysef, si, and sysid arguments

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 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.

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 1 108 0 11 1 0 0 $P90 $I0 $H93 PN 0
8 1 “G0 - Quality Reject” $O1 G0 $P70 $I0
```

In this example:

2452	x
14465	y
869	w
898	h
ON	state
7	poke_type
1	num_of_progs
105	prog_num
0	diag_num
11	num_of_args
1	blk
0	sysef

0	si
$P90 $I0	sysid
$H93 PN	pnid
0	cmd
8	bit
1 (send event and description)	evflag
“G0 - Quality Reject”	evdesc
$O1 G0	off/typ
$P70 $I0	bitval

When you click on the poke field (specified as type 7, program) program number 108 (SEND_CA_EV_BIT) is executed. A new diagram will not be displayed after the message is sent. Eleven arguments are associated with this command.

The system entry field buffer number (sysef) and screen index (si) arguments are both zero (0) so the system ID (sysid) is $P90 $I0.

The point name to be changed is $H93 PN. The field to be changed is $O1 G0. The bit to be changed is bit 8. The new value is contained in $P70 $I0.

5.32 WINDOW_FUNC_KEY (117)

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 4** for more information on this command).

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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 the ***POKE_FLD command*** (see page 171).
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

Example 1

```
POKE_FLD 2958 13603 1010 1437 ON 7 1 117 7001 5 0 0 0 0 0
```

In this example:

2958	x
13603	y
1010	w
1437	h
ON	state
7	poke_type
1	num_of_progs
117	prog_num
7001	diag_num
5	num_of_args
0	group
0	dispx
0	dispy
0	type
0	num_of_points

In this example, poke type 7 calls one application program, number 117 (WINDOW_FUNC_KEY). When the poke field is selected, window number 7001 will display. Five arguments will be used in this command. No groups will be used. No points will be passed.

Example 2

```
POKE_FLD 2958 13603 1010 1437 ON 7 1 117 7001 8 0 0 0 0 3
TANK101LEVEL ID TANK103STPT ID TANK103FILLDMD ID
```

In this example:

2958	x
13603	y
1010	w
1437	h
ON	state
7	poke_type
1	num_of_progs
117	prog_num
7001	diag_num
8	num_of_args
0	group
0	dispx
0	dispy
0	type
3	num_of_points
TANK101LEVEL ID TANK103STPT ID TANK103FILLDMD ID	point_list

In this example, poke type 7 calls one application program, number 117 (WINDOW_FUNC_KEY). When the poke field is selected, window number 7001 will display. Eight arguments will be used in this command. No point groups will be used. Three points (TANK101LEVEL ID, TANK103STPT ID, TANK103FILLDMD ID) were passed to replace $W pointers in this window diagram.

Example 3

```
POKE_FLD 2958 13603 1010 1437 ON 7 1 117 7001 5 1 0 0 0 0
```

In this example:

2958	x
13603	y
1010	w
1437	h
ON	state
7	poke_type
1	num_of_progs
117	prog_num
7001	diag_num
5	num_of_args
1	group
0	dispx
0	dispy
0	type
0	num_of_points

In this example, poke type 7 calls one application program, number 117 (WINDOW_FUNC_KEY). When the poke field is selected, window number 7001 will display. Five arguments will be used in this command. Points from group 1 will be used.

5.33 DISP_EFDATA (119)

DISP_EFDATA displays data in entry fields in a window. This data is read from the Engineer Station's memory or specified in the DISP_EPDATA command.

This command provides the same functionality and has the same arguments as ***DISP_EFDATA (66)*** (see page 278). 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 a main screen, as opposed to a window).

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

5.34 XPID_DIGITAL (121)

XPID_DIGITAL allows you to change the setpoint and output fields of an XPID-type algorithm.

Two formats exist for this program. The simple format requires one argument and allows you to move the setpoint/output to a predefined value immediately or to slew the setpoint/output to the value over a period of 20 seconds.

If you want to change the **setpoint**, the new value will be read from **entry field 1** on the currently displayed window. If you want to change the **output**, the new value will be read from **entry field 2** on the currently displayed window.

The enhanced format uses five 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, and so forth).

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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 (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments for this application program. Valid values are 1 or 5.

function	If **num_of_args = 1**, valid values are: ▪ 1 = change set point with no slew (read value from window entry field 1). ▪ 2 = change output with no slew (read value from window entry field 2). ▪ 3 = change setpoint with 20 second slew rate (read value from window entry field 1). 4 = change output with 20 second slew rate (read value from window entry field 2). ***OR*** If **num_of_args = 5**, valid values are: ▪ 1 = change setpoint ▪ 2 = change output
If num_of_args = 5, enter a value for ef_value, si_value, ef_slew, and si_slew.	
ef_value	Entry field number for value (enter 0 if using parameter list)
si_value value	Entry field screen index for value (enter if ef_value > 0). Valid entries are: ▪ 1 = Entry field in main screen. ▪ 2 = Entry field in subscreen. ▪ 7 = Entry field in window. ***OR*** New set value (enter if ef_value = 0)
ef_slew	Entry field number for slew rate (enter 0 if using parameter list)
si_slew slew	Entry field screen index for slew rate (enter if ef_slew > 0). Valid entries are: ▪ 1 = Entry field in main screen. ▪ 2 = Entry field in subscreen. ▪ 7 = Entry field in window. ***OR*** Slew rate in seconds (enter if ef_slew = 0) Valid range = 1 through 255.

EXAMPLES

Example 1

```
POKE_FLD 274 2911 989 1081 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1 9
2 121 0 1 1
```

where:

274	x
2911	y
989	w
1081	h
ON	state
7	poke_type
2	num_of_progs
6	prog_num
0	diag_num
5	num_of_args
001-00123 ID	point1
001-00123 ID	point2
1	trig_num
9	set_num
2	setval
121	prog_num
0	diag_num
1	num_of_args
1 = change setpoint with no slew.	function

In Example 1, the set point of 001-00123 changes with no slew. When the poke field is selected, application program number 6, CNTRL_POKE, is executed to select point 001-00123 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 window, and the necessary point attributes are changed by the program to change the set point.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 274 2911 989 1081 ON 7 2 6 0 5 001-00123 ID 001-000123 ID 1
9 2

POKE_FLD 419 10111 412 322 ON 23 1 2 1 121 0 5 1 0 $P3 $S0 0 30
```

where:

POKE #1	
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	001-00123 ID
point2	001-00123 ID
trig_num	1
set_num	9
setval	2
POKE #2	
x	419
y	10111
w	412
h	322
state	ON
poke_type	23
set_num	1
set_val	2

num_of_progs	1
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 001-00123 changes using a slew rate. The program CNTRL_POKE is used to select the point for control as in the first example (this is defined by poke type 7). Poke type 23 is used to execute the XPID_DIGITAL program. 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.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same as those defined for Poke Type 23.*

5.35 EXEC_TRIGGER (122)

EXEC_TRIGGER executes a diagram trigger from a poke field.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args si trigger (set_num set_val)
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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	Diagram number to be displayed (0 is required if no other diagram is called). For this application program **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **2** or **4** will be entered on the command line. If 4 is entered, the set_num and set_val must be defined.
si	Identifies the type of diagram containing the trigger. The choices are: ▪ 1 = Main screen ▪ 2 = Subscreen ▪ 7 = Window
trigger	Number of the diagram trigger to be executed. Valid trigger numbers are 1 through 255.
set_num	Only used if num_of_args is set to 4. Set variable number. Valid range = 1 through 255 or $SET1 through $SET255.
set_val	Only used if num_of_args is set to 4. Integer value to be assigned to the set variable. Valid range = 0 through 32,767.

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.

5.36 CNTRLBITS (124)

CNTRLBITS activates the Operator Station P-Key functions to interface with certain control algorithms.

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 keyboard function for the selected algorithm.

Note that the first algorithm or point name listed in the CNTRL_POKE program is assigned to keyboard functions 1 through 4 and the second algorithm or point name listed is assigned to keyboard functions 5 through 8.

***Note:** To use all 8 keyboard functions with the same point, enter the same point name into both arguments of the CNTRL_POKE program.*

To use only P1 through P4, 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 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 *Ovation Record Types Reference Manual*.)

At the Operator Station, the CP field is read into a temporary variable. Each keyboard 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 record.

The algorithm 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 Station always reads from the CP field and writes to the CN field. The Controller 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 type LC which include these fields. The most commonly used algorithms of these types are KEYBOARD, 2XSELECT, MASTATION, and SETPOINT. (See *Ovation Algorithms Reference Manual*.)

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args Pkey
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** or **23** may be used 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 to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments for the application program specified
Pkey	Number that corresponds to the keyboard functions. Valid range = 1 through 8.

EXAMPLE

Example 1

```
POKE_FLD 2452 14465 869 1389 ON 7 2 6 0 5 001-00123 ID 001-00123 ID 1
9 2 124 0 1 1
```

In this example:

2452	x
14465	y
869	w
1389	h
ON	state
7	poke_type
2	num_of_progs
6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID	point1

001-00123 ID	point2
1	trig_num
9	set_num
2	setval
124 (CNTRLBITS)	prog_num
0	diag_num
1	num_of_args
1	Pkey

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 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 keyboard 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 keyboard function 1, the action defined by the PK1 algorithm parameter will be taken.

Example 2

For windows with multiple MA Stations, it is recommended that both Poke Type 7 and 23 be used for control, as shown below:

```
POKE_FLD 722 13256 642 552 ON 7 1 6 0 5 001-00123 ID 001-00123 ID 1 1
2
```

```
POKE_FLD 419 10111 412 322 ON 23 1 2 1 124 0 1 1
```

In this example:

POKE #1	
722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs

6 (CNTRL_POKE)	prog_num
0	diag_num
5	num_of_args
001-00123 ID (KEYBOARD algorithm)	point 1
001-00123 ID (KEYBOARD algorithm)	point 2
1	trig_num
1	set_num
2	set_val
POKE #2	
419	x
10111	y
412	w
322	h
ON	state
23	poke_type
1	set_num
2	set_val
1	num_of_progs
124 (CNTRLBITS)	prog_num
0	diag_num
0	num_of_args
1	pkey

When you click on the poke field (specified as type 7), program 6 (CNTRL_POKE) will execute enabling the keyboard algorithm, and putting the MA Station in control.

When the poke field specified as poke type 23 is pressed, CNTRLBITS is executed. Since the poke field is associated with keyboard function 1, the action defined by the PK1 algorithm parameter will be taken.

Note: *Make sure that the set numbers and set values defined for Poke Type 7 are the same as those defined for Poke Type 23.*

5.37 WINDOW_DELETE (125)

WINDOW_DELETE removes the control window from the CRT screen.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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	Application program number. For this program, **125** is entered on the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments for this application program. 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:

12636	x
2730	y
2809	w
2185	h
ON	state
7	poke_type

1	num_of_progs
125	prog_num
0	diag_num
0	num_of_args

When you click 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.

5.38 SUBWINDOW_DELETE (128)

SUBWINDOW_DELETE removes the control subwindow from the CRT screen.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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	Application program number. For this program, **128** is entered on the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments for this application program. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 12636 2730 2809 2185 ON 7 1 128 0 0
```

In this example:

12636	x
2730	y
2809	w
2185	h
ON	state
7	poke_type

1	num_of_progs
128	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7) program 128 (SUBWINDOW_DELETE) executes. The control subwindow (if one is displayed with this frame) will be removed.

5.39 POKE_EVENT (129)

POKE_EVENT executes the poke field at the virtual window coordinates in the parameter list.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args screen_index xcoord ycoord
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **129** will be entered on the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. For this application program, **3** will be entered on the command line.
screen_index	Screen index. Valid entries are: ▪ 1 = Poke field in main screen. ▪ 2 = Poke field in subscreen. ▪ 7 = Poke field in window.
xcoord	x coordinate of the poke field to be executed.
ycoord	y coordinate of the poke field to be executed.

EXAMPLE

```
POKE_FLD 12636 2730 2809 2185 ON 7 1 129 0 3 7 13458 15425
```

In this example:

12636	x
2730	y
2809	w
2185	h
ON	state
7	poke_type
1	num_of_progs
129	prog_num
0	diag_num
3	num_of_args
7	screen_index
13458	xcoord
15425	ycoord

When you click on the poke field (specified as type 7) program 129 (POKE_EVENT) executes. In this example, the poke field at coordinates 13458 15425 executes in a window.

5.40 CONT_RAISE_LOWER_VAL (130)

The CONT_RAISE_LOWER_VAL (Continuous Raise/Lower Value) program is used to gradually raise or lower the value of a point or pointer variable. You specify the point/pointer to change (Target Point), a start value, high and low limits, the percent change in the first four seconds, and the number of seconds to reach its maximum value.

Note: *This program should NOT be used to raise/lower the setpoint or output of an algorithm. Programs Upset (30), Downset (31), Raiseout (34), and Lowerout (35) should be used for these purposes.*

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num   diag_num
num_of_args direction target_point start_point high_limit low_limit
4_sec_change sec_to_max
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **130** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments for the application program specified. For this application program, **7** will be entered on the command line.
direction	Identifies the direction ▪ 0 = Lower ▪ 1 = Raise
target_point	Name of point to change. Valid values are point names with record fields and $P pointers with offsets ($B, $I, $R, or $S).

start_point	Name of point from which to read the initial value. Valid values are point names with record fields and $P pointers with offsets ($B, $I, $R, or $S), integers and real numbers.
high_limit	Name of point from which to read the high limit. Valid values are point names with record fields and $P pointers with offsets ($B, $I, $R, or $S), integers and real numbers.
low_limit	Name of point from which to read the low limit. Valid values are point names with record fields and $P pointers with offsets ($B, $I, $R, or $S), integers and real numbers.
4_sec_change	Name of point from which to read the 4 second change value. Valid values are point names with record fields and $P pointers with offsets ($B, $I, $R, or $S), integers and real numbers.
sec_to_max	Name of point from which to read the seconds to maximum value. Valid values are point names with record fields and $P pointers with offsets ($B, $I, $R, or $S), integers and real numbers.

EXAMPLES

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 1 130 0 7 1 \A100\ AV \001-00123\ ID
100.0 0.0 10.0 25
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
130	prog_num
0	diag_num
7	num_of_args
1 (Raise)	direction
\A100\ AV	target_point
\001-00123\ ID	start_point
100.0	high_limit
0.0	low_limit
10.0	4_sec_change
25	sec_to_max

When you click on the poke field (specified as type 7) program 130 (CONTINUOUS RAISE/LOWER VALUE) executes. In this example, the value of the AV field of point \A100\ will continuously increment (raise). The initial value will be read from the ID field of algorithm \001-00123\. The limits are hard-coded at 0.0 and 100.0. The percent change in the first four seconds is set to 10% of the range between the top and bottom scale. The number of seconds to ramp to full scale is set to 25 seconds.

Example 2

```
POKE_FLD 722 13256 642 552 ON 7 1 130 0 7 0 \A100\ AV $P1 $R0 $P1 $R4
$P1 $R8 $P1 $I12 $P1 $I16
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
130	prog_num
0	diag_num
7	num_of_args
0 (Lower)	direction
\A100\ AV	target_point
$P1 $R0	start_point
$P1 $R4	high_limit
$P1 $R8	low_limit
$P1 $I12	4_sec_change
$P1 $I16	sec_to_max

When you click on the poke field (specified as type 7) program 130 (CONTINUOUS RAISE/LOWER VALUE) executes. In this example, the value of the AV field of point \A100\ will continuously decrement (lower). The initial value will be read from pointer $P1 $R0. The limits are $P1 $R4 (high) and $P1 $R8 (low). The percent change in the first four seconds is read from pointer $P1 $I12. The number of seconds to ramp to full scale is read from pointer $P1 $I16.

5.41 SEND_GENMSG_NETWORK (167)

SEND_GENMSG_NETWORK reads data from a list of arguments within the POKE_FLD command, sends a general message over the Ovation Network 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 window diagram and send data to another drop for processing. You are responsible for the proper formatting of the block data. For example, there are byte order differences between the Station and the Controller. When sending data from the Station to the Controller, alternate bytes of data must be swapped for ASCII data and floating point data.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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 net_ef net_si
net_num drpef si id msgtype ef data dec
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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.
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 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 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 or window).
NET_EF1	Entry field number to read the network number. If the network number will not be read from an entry field, this entry should be 0.
NET_SI1	Screen index for entry field to contain the network number. The choices are: 0 = Not used. 1 = Entry field in main screen. 7 = Entry field in window.
NET_NUM1	Network number of receiving drop. This is the network 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.
drpef2	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.

si2	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 7 = Window
id2	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.
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 entry field to read, or 0 for data to be specified in 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.
Note 1 The net_ef, net_si, and net_num arguments are used together.	
Note 2 The drpef, si, and id arguments are used together.	

Net_ef, net_si, and net_num arguments

Entry field number (net_ef argument)	**Screen Index (net_si argument)**	**System ID (net_num argument)**
0	0	Network number. For example, $P1 $I0 or constant value (for example, 1) representing the actual network number.
Entry field number (N)	Screen index (S) to location of entry field: 1 = Main 3 = Window	0 = Entry field N on screen S is a point name. 1 = Entry field N on screen S is an integer network number.

Drpef, si, and id arguments

Entry field number (drpef argument)	**Screen Index (si argument)**	**System ID (id argument)**
0	0	Point name (for example, \A100\ AV 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 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.

Note: *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.*

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 7 0 1 1 0
3000 0 0 $P1 $I0 0 0 165 0 1 0 30 2 2 2
```

In this example:

483	x
14400	y
546	w

904	h
ON	state
7	poke_type
1	num_of_progs
67	prog_num
0	diag_num
20	num_of_args
0	blk
3	total
1	sia
1	a
7	sib
0	b
1	drop
1	size
0	seg
3000	tdiag
0	net_ef
0	net_si
$P1 $I0	net_num
0	drpef
0	si
165	id
0 1 *message type (bid a prgm) 0 30 *prgm 30	msgtype
2	ef
2	data
2 *send real number	dec

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 window (sib = 7, 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 window 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 window entry fields.
	•				
	•				
	•				
ef #	data		dec		
1	1			*	read an integer from main screen entry field 1.
2	2		2	*	read a real from window entry field 2 (two decimal places
0	$S3		ID	*	read point name from window 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).

5.42 SEND_CA_CTL_LOCK (180)

The SEND_CA_CTL_LOCK program is identical to the SEND_CA (80) program except for the lock-out procedure.

Note: *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.*

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x			
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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.

Note: *The SEND_CA_CTL_LOCK program is accessed by two numbers: 98 and 180.*

5.43 TREND_GROUP (201)

TREND_GROUP sends a request to the Trend System to display a live or historical trend group.

Function Lockout

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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). For this application program, **0** will be entered on the command line.
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:

965	x
1447	y
869	w

1389	h
ON	state
7	poke_type
1	num_of_progs
201	prog_num
0	diag_num
2	num_of_args
1	type
350	trend_num

When you click 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 *Ovation Operator Station User Guide*.)

5.44 EXECUTE_PROCESS (202)

EXECUTE_PROCESS mirrors a command line prompt with parameters. It is run from a poke type 7 command. It is also used to implement the poke type 9 command. You may want to run this program from a poke type 7 if additional application programs need to be run when this program is run.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

POKE_FLD x y w h state poke_type progs prog_num diag_num num_of_args process

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** is required for this application program).
progs	Number of programs to execute (usually 1, unless additional application programs are linked to this poke field)
prog_num	Application program number. For this application program, **202** will be entered on the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is to be displayed). For this application program, **0** will be entered on the command line. Valid range is 0 through 65635.
num_of_args	Number of arguments. For this application program, **1** will be entered on the command line.
process	The complete path name plus parameters of the process to be executed (string must be in quotes). ***OR*** A pointer to a string.

Examples

Example 1

```
POKE_FLD 2013 3560 1422 1373 ON 7 1 202 0 1
"c:\ovation\ovationbase\trend.exe"
```

where:

2013	x
3560	y
1422	w
1373	h
ON	state
7	poke_type
1	progs
202	prog_num
0 - no diagram to be displayed	diag_num
1	num_of_args
“c:\ovation\ovationbase\trend.exe”	process

When you click on the poke field (specified as type 7), program 202 (EXECUTE_PROCESS) runs. The program starts the trend executable with no parameters.

Example 2

```
POINTER $P1 255 0
PTR_VALUE $P1 $A0X32 "c:\ovation\ovationbase\trend.exe"
POKE_FLD 100 10000 1422 1373 ON 7 1 202 0 1 $P1 $A0X32
```

where:

100	x
10000	y
1422	w
1373	h
ON	state
7	poke_type
1	progs
202	prog_num

0 - no diagram to be displayed	diag_num
1	num_of_args
$P1 $A0X32	process

When you click on the poke field (specified as type 7), program 202 (EXECUTE_PROCESS) runs. A PTR_VALUE command defined a pointer to execute the trend program.

5.45 CLEAR_CONTROL (203)

CLEAR_CONTROL clears the control algorithm selections for all of the windows in the Process Diagram System.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type progs prog_num diag_num args
```

OR

```
POKE_FLD x y w h state poke_type progs prog_num diag_num num_of_args
trig_num set_num setval
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	7
progs	Number of programs to execute (usually 1, unless additional application programs are linked to this poke field)
prog_num	Application program number. For this application program, **203** will be entered on the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is to be called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments. You have two choices: ▪ If the number of arguments is 0, the program uses the last trigger number and set number associated with control with a set value of 0. ▪ If the number of arguments is 3, use the trigger number, set number, and set value specified in this application program.
trig_num	Trigger number to be executed when selected. Valid range is 1 through 254.
set_num	Number of the set to be used in the trigger. Valid range is 1 through 255.
setval	Integer value to be assigned to the set variable in the trigger. Valid range is 0 through 32,767.

EXAMPLE

```
POKE_FLD 965 14447 869 1389 ON 7 1 203 0 3 1 9 2
```

In this example:

965	x
14447	y
869	w
1389	h
ON	state
7	poke_type
1	num_of_progs
203	prog_num
0	diag_num
3	num_of_args
1	trig_num
9	set_num
2	setval

When you click on the poke field (specified as type 7, program), program 203 (CLEAR_CONTROL) will execute. The system IDs and algorithm records that are interfaced to the Control Panel window are cleared.

Since the number of arguments is 3, the program will use the trigger number, set number, and set value specified in this example (1, 9, and 2 respectively). Refer to the ***SETVAL*** (see page 201) ***and TRI***G_ON (see page 216) commands.

5.46 RESET_CONTROL (204)

RESET_CONTROL clears the control algorithm selection for the current process diagram window set (Main, Subscreen, and Window).

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
poke_type	Poke type (**7** is required for this application program).
progs	Number of programs to execute (usually 1, unless additional application programs are linked to this poke field)
prog_num	Application program number. For this application program, **204** will be entered on the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called). For this application program, **0** will be entered on the command line.
num_of_args	Number of arguments for this application program. For this application program, **0** will be entered on the command line.

EXAMPLE

```
POKE_FLD 12636 2730 2809 2185 ON 7 1 204 0 0
```

In this example:

12636	x
2730	y
2809	w
2185	h

ON	state
7	poke_type
1	num_of_progs
204	prog_num
0	diag_num
0	num_of_args

When you click on the poke field (specified as type 7) program 204 (RESET_CONTROL) executes. The algorithm is cleared from the process diagram.

5.47 SEND_CA_EV_CTL_LOCK (205)

The SEND_CA_EV_CTL_LOCK program is identical to the SEND_CA_EV (105) program except for the lock-out procedure.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
x			
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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.

Note: *The SEND_CA_EV_CTL_LOCK program is accessed by two numbers: 106 and 205.*

5.48 ACK_DROP_ALARM (210)

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 *Ovation Error Codes and Messages*.) 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.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
			x
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type prog_num
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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:

8288	x
14697	y
445	w
415	h
ON	state
3	poke_type
210	prog_num

When you click on the poke field (specified as type 3), program 210 (ACK_DROP_ALARM) will execute on the Drop Status record that you selected.

5.49 CLEAR_DROP_FAULT (212)

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 *Ovation Record Types Reference Manual*.) This program is intended for use on the System Status Display. (See *Ovation Error Codes and Messages*.)

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
			x
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type prog_num
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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:

8288	x
15318	y
445	w
415	h
ON	state
3	poke_type
212	prog_num

When you click on the poke field (specified as type 3), program 212 (CLEAR_DROP_FAULT) will execute on the Drop Status record that the operator selected.

5.50 ALARM_ACKNOWLEDGE (214)

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.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
			x
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

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 the ***POKE_FLD command*** (see page 171).
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 for which to acknowledge alarms. 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:

8892	x
4368	y
2341	w
2185	h

ON	state
7	poke_type
1	progs
214	prog_num
0 - no new diagram will be displayed	diag_num
3	num_pts
\AI212S01\ ID, \AI212S02\ ID, \AI212S03\ ID	points

When you click 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 num_pts parameter is greater than 40, then the first 40 points in alarm will be acknowledged when the poke is selected. You 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, you would have to select the poke a maximum of three times to acknowledge all alarms, assuming all 100 points were in alarm.*

5.51 XPID_DIGITAL_CTL_LOCK (221)

The XPID_DIGITAL_CTL_LOCK program is identical to the XPID_DIGITAL (121) program except for the lock-out procedure.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
x			
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

Refer to the ***XPID_DIGITAL (121)*** (see page 329) section for the syntax and example.

5.52 READ_ENTRY_FIELD (224)

READ_ENTRY_FIELD reads data from an entry field on either a main screen, a window, or a subscreen into a pointer value ($P value). The program also returns a status value indicating the success or failure of the operation. The status is also stored in a pointer value specified by the user.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args si ef pointer_value/offset return_value/offset
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **224** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called).
num_of_args	Number of arguments. For this application program, **4** will be entered on the command line.
si	Identifies the type of diagram of the entry field to read from. The choices are: ▪ 1 = Main screen ▪ 2 = Subwindow ▪ 7 = Window
ef	Entry field number containing the data to be read.

pointer_value/offset	$P pointer where the data will be read into. Valid choices are: ▪ $Px $Ry (real value ▪ $Px $Iy (integer value) ▪ $Px $Sy (SID value) ▪ $Px $By (byte value) ▪ $Px $AyXz (text string)
return_value/offset	$P pointer where the status of the read will be stored. The only valid choice is $Px $By (byte value). The status returned will one of the following: ▪ 0 = OK ▪ 1 = Error in read ▪ 2 = Entry field is blank

EXAMPLE

```
POKE_FLD 722 13256 642 552 ON 7 1 224 0 4 7 1 $P2 $R0 $P2 $B4
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
224	prog_num
0	diag_num
4	num_of_args
7	si
1	ef
$P2 $R0	pointer_value/offset
$P2 $B4	return_value/offset

When you click on this poke field (specified as type 7), program 224 (READ_ENTRY_FIELD) will execute. The program will read the value of entry field 1 on the currently displayed window and store the value in pointer $P2 offset 0. The value will be stored as a real value. If the read completed successfully, a return value of 0 will be stored in $P2 offset 4. If the read failed, a value of 1 will be stored in $P2 offset 4.

5.53 WRITE_OPC_DATA (225)

WRITE_OPC_DATA reads data from an Ovation highway point, a pointer value ($P value), a group point ($G value), a window point ($W value) or another OPC point value and writes the resulting value to an OPC point defined by the user.

FUNCTION LOCKOUT

Control Function Enabled	Tuning Function Enabled	Algorithm Point Access Enabled	Alarm Acknowledge Enabled
N/A	N/A	N/A	N/A
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args read_value/offset OPC_point_name
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **225** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called).
num_of_args	Number of arguments. For this application program, **2** will be entered on the command line.
read_value/offset	Field to read the data from.
OPC_tag_name	Name of the OPC point where the data will be written.

EXAMPLE

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 1 225 0 2 A100 AV
\OPC$SERVER1$TESTPOINT.UNIT0@WN3.AV\
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
225	prog_num
0	diag_num
2	num_of_args
A100 AV	read_value/offset
\OPC$SERVER1$TESTPOINT.UNIT0@WN3.AV\	OPC_tag_name

When you click on the poke field (specified as type 7), program 225 (WRITE_OPC_DATA) will execute. The program will read the value of A100 AV and write the resulting value to the OPC tag. TESTPOINT.UNIT0@WN3.AV in logical server SERVER1.

Example 2

```
POKE_FLD 722 13256 642 552 ON 7 2 224 0 4 1 1 $P2 $R0 $P2 $B4 225 0 2
$P2 $R0 \OPC$SERVER1$TESTPOINT.UNIT0@WN3.AV\
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
2	num_of_progs

224	prog_num
0	diag_num
4	num_of_args
1	si
1	ef
$P2 $R0	pointer_value/offset
$P2 $B4	return_value/offset
225	prog_num
0	diag_num
2	num_of_args
$P2 $R0	read_value/offset
\OPC$SERVER1$TESTPOINT.UNIT0@WN3.AV\	OPC_tag_name

When you click on the poke field (specified as type 7), program 225 will execute. The program will read the value of A100 AV and write the resulting value to the OPC tag TESTPOINT.UNIT0@WN3.AV in logical server SERVER1.

5.54 SCADA_CMD (226)

SCADA_CMD will send a command to a SCADA communication server. The SCADA server is determined to be the originating drop/partner drop of the defined point. The allowed commands are defined in the SCADA Commands table and are grouped by category. The categories are: Server, RTU, Communication Line and Control Point.

FUNCTION LOCKOUT

Control Function Enabled	**Tuning Function Enabled**	**Algorithm Point Access Enabled**	**Alarm Acknowledge Enabled**
		x	
Refer to ***Security*** (see page 224) for an explanation of the above functions.			

SYNTAX

```
POKE_FLD x y w h state poke_type num_of_progs prog_num diag_num
num_of_args command point_name [value pulse_duration]
```

where:

x, y, w, h, state	Standard POKE_FLD parameters. See the ***POKE_FLD command*** (see page 171).
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, **226** will be entered in the command line.
diag_num	Diagram number to be displayed (0 is required if no other diagram is called).
num_of_args	Number of arguments to follow.
command	Command number. Refer to the following table.
point_name	Name of the Ovation point to associate with the command.
value/offset	For command = 31/33 only. Optional setpoint value (may be a constant, point or pointer value).
pulse_duration	For command = 33 only. Duration of the pulse in seconds.

SCADA Commands

Category	Command Value	Description	Point Type
Server	1	Fail primary SCADA Server	Drop Status (DU)
RTU	11	Fail RTU	“any”
	12	Restore RTU	“any”
	13	Move blocks to fast scan	“any”
	14	Reset communication error counters	“any”
	15	Warm restart of RTU	“any”
Communication Line	21	Fail Primary Port	“any”
	22	Restore Primary Port	“any”
	23	Fail Backup Port	“any”
	24	Restore Backup Port	“any”
	25	Reset counters	“any”
	26	Warm restart of the line	“any”
	27	Cold restart of the line	“any”
	28	Fail Line	“any”
	29	Restore Line	“any”
Control Point	31	Set Value	“any”
	33	Pulse Value (1 to 0)	“any”
	34	Clear Control Point status	“any”

EXAMPLES

Example 1

```
POKE_FLD 722 13256 642 552 ON 7 1 226 0 2 1 DROP51
```

In this example:

722	x
13256	y
642	w
552	h

ON	state
7	poke_type
1	num_of_progs
226	prog_num
0	diag_num
2	num_of_args
1	command (Fail primary SCADA server)
DROP51	point_name

When you click on the poke field (specified as type 7), program 226 (SCADA_CMD) will execute. The program will send the command (1 - Fail primary SCADA server) to the SCADA server (drop51).

Example 2

```
POKE_FLD 722 13256 642 552 ON 7 1 226 0 2 31 DROP51 $P1 $R0
```

In this example:

722	x
13256	y
642	w
552	h
ON	state
7	poke_type
1	num_of_progs
226	prog_num
0	diag_num
2	num_of_args
command	31 (Set SCADA value)
point_name	DROP 51
value/offset	$P1 $R0 (Read value from $P1 $R0

When you click on the poke field (specified as type 7), program 226 will execute. The program will send the command (31 - Set SCADA value) to the SCADA server (drop51). The SCADA value will be read from $P1 $R0.

APPENDIX A

RESERVED WORDS

A.1 What are Reserved Words?

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, $W, and $O) and offsets ($In, $Rn, $Bn, $Sn, $AnXi).
- Record type and record field names. (See *Ovation Record Types Reference Manual*.)
- 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. The are alphabetically arranged left to right.

ALARM	ALARMACK	ALARMOFF
ARC	ASCII	BACKGROUND
BAD	BAR	BETTER
BIAS	BINARY	BITMAP
BITMAP_OVER	BLINK	BOLD
BOLD_ITALIC	BOTH	BYTE
CIRCLE	COLOR	CURSOR
CUTOUT	DA	DATE
DD	DEF_FKEY_GROUP	DEF_MACRO_PARAMS
DEF_QUAL	DIAG_DISP	DIAGRAM
DISABLE	DOT	DOWN
DP	DROPALM	DROPCLEAR

DROPFAULT	DU	DYNAMIC_LINE
DYNAMIC_POLYGON	EF_STATE	ELLIPSE
ELSE	ENDIF	ENDLOOP
ENTERVALUE	ENTRY_FIELD	EXCL
EXEC_POKE	EXEC_TRIG	EXIT
EXPONENTIAL	FAIR	FALSE
FG	FITTED	FKEY_STATE
FONT	FORCE_UPDATE	FOREGROUND
FORMAT	FUNC_KEY	GCODE
GOOD	GTEXT	HDWRFAIL
HEX	HEX_H	HIGHALARM
HORZ	IF	IF_CHANGED
IFELSE	INT	INVOKED
ITALIC	KEYBOARD	LA
LARGE	LC	LD
LEFT	LIMITOFF	LINE
LOAD_FKEY_GROUP	LOOP	LOWALARM
LP	MACRO	MAIN
MATH	MCB0OFFLIN	MCB1OFFLIN
MEDIUM	MULT	MULTI_TEXT
NONE	NON_EXCL	NORMAL
NOSCALE	NOT_FITTED	OFF
OFF0	OFF1	OFF2
OFF3	OFF4	OFF5
OFF6	OFF7	OFF8
OFF9	OFF10	OFF11
OFF12	OFF13	OFF14
OFF15	OFF16	OFF17
OFF18	OFF19	OFF20
OFF21	OFF22	OFF23
OFF24	OFF25	OFF26

OFF27	OFF28	OFF29
OFF30	OFF31	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	ON16
ON17	ON18	ON19
ON20	ON21	ON22
ON23	ON24	ON25
ON26	ON27	ON28
ON29	ON30	ON31
OPATTN	PB	P_ENDLOOP
PAGE	PLOT	POINTER
POKE_FIELD	POKE_STATE	POLYGON
POOR	PRESET0	PRESET1
PRESET2	PRESET3	PRESET4
PRESET5	PRESET6	PRESET7
PRESET8	PRESET9	PRESET10
PRESET11	PRESET12	PRESET13
PRESET14	PRESET15	PRESET16
PRESET17	PRESET18	PRESET19
PRESET20	PRESET21	PRESET22
PRESET23	PRESET24	PRESET25
PRESET26	PRESET27	PRESET28
PRESET29	PRESET30	PRESET31
PROCESS_PT	PSET0	PSET1

PSET2	PSET3	PSET4
PSET5	PSET6	PSET7
PSET8	PSET9	PSET10
PSET11	PSET12	PSET13
PSET14	PSET15	PSET16
PSET17	PSET18	PSET19
PSET20	PSET21	PSET22
PSET23	PSET24	PSET25
PSET26	PSET27	PSET28
PSET29	PSET30	PSET31
PTR_EQUAL	PTR_LOOP	PTR_MOVE
PTR_VALUE	REAL	RECTANGLE
REGULAR	RESET	RESETALM
RIGHT	RIGHT0	RM
RN	ROTATION	ROUNDED
RTL	RUN_PROGRAMS	SCALE
SCANOFF	SENSORALM	SENSORMODE
SET	SETALM	SETVAL
SHAPE	SHAPE_LABEL	SHAPE_PLOT
SMALL	SQUARED	SUB
SUBWIN	TEXT	TEXT_LABEL
TIME	TIMED_OUT	TITLE
TOGGLE	TREND	TRIGGER
TRIG_ON	TRUE	TTB
UP	UPDATETIME	VECTOR
VECTOR_OVER	VERT	WIN
WORSE	XY_PLOT	

APPENDIX B

STATUS WORDS

B.1 What are Status Words?

This appendix lists the Graphics Language status words. The general purpose words (ONx, OFFx, and so forth) enable you 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 *Ovation Record Types Reference Manual*.) 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).

Note: *In the above example with NORMAL, bits 16 through 31 are 0.*

B.2 List of Status Words

The following table provides the following information on the Graphics Language status words: description, mask, pattern, and record type (default record fields).

Graphics Language Status Words

Status Words	**Description**	**Mask** **32 0**	**Pattern** **32 0**	**Record Type (Record Field)**
ALARM	Point in Alarm	0000 0000 0000 0000 1000 0000 1000 0000	0000 0000 0000 0000 0000 0000 1000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)

Status Words	Description	Mask 32 0	Pattern 32 0	Record Type (Record Field)
ALARMACK	Unacknowl-edged Alarm	0000 0000 0000 0000 1000 0000 0010 0000	0000 0000 0000 0000 0000 0000 0010 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
ALARMOFF	Alarm Check Off	0000 0000 0000 0000 1010 0000 0000 0000	0000 0000 0000 0000 0010 0000 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
BAD	Quality = Bad	0000 0000 0000 0000 1000 0011 0000 0000	0000 0000 0000 0000 0000 0011 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
BETTER	Better	0000 0000 0000 0000 1000 0000 0000 0011	0000 0000 0000 0000 0000 0000 0000 0001	Long Analog (LA) Deluxe Analog (DA)
CUTOUT	Cutout	0000 0000 0000 0000 1000 0000 0100 0000	0000 0000 0000 0000 0000 0000 0100 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
DROPALM	Drop in Alarm	0000 0000 0000 0000 0000 0000 1000 0000	0000 0000 0000 0000 0000 0000 1000 0000	Drop Status (DU)
DROPCLEAR	Drop not in Alarm	0000 0000 0000 0000 0000 0000 1000 0000	0000 0000 0000 0000 0000 0000 0000 0000	Drop Status (DU)
DROPFAULT	Drop Fault	0000 0000 0000 0000 0000 0000 0100 0000	0000 0000 0000 0000 0000 0000 0100 0000	Drop Status (DU)
ENTERVALUE	Value was Entered	0000 0000 0000 0000 1000 0100 0000 0000	0000 0000 0000 0000 0000 0100 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
FAIR	Quality = Fair	0000 0000 0000 0000 1000 0011 0000 0000	0000 0000 0000 0000 0000 0001 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)

Status Words	Description	Mask 32 0	Pattern 32 0	Record Type (Record Field)
GOOD	Quality = Good	0000 0000 0000 0000 1000 0011 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
HDWRFAIL	Hardware Fail (timed-out)	0000 0000 0000 0000 1000 0000 0000 0000	0000 0000 0000 0000 1000 0000 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD) Long Packed (LP) Deluxe Packed (LP)
HIGHALARM	High Alarm	0000 0000 0000 0000 1000 0000 1000 1100	0000 0000 0000 0000 0000 0000 1000 1000	Long Analog (LA) Deluxe Analog (DA)
LIMITOFF	Limit Check Off	0000 0000 0000 0000 1001 0000 0000 0000	0000 0000 0000 0000 0001 0000 0000 0000	Long Analog (LA) Deluxe Analog (DA)
LOWALARM	Low Alarm	0000 0000 0000 0000 1000 0000 1000 1100	0000 0000 0000 0000 0000 0000 1000 0100	Long Analog (LA) Deluxe Analog (DA)
MCB0OFFLIN	MBC 0 Inactive	0000 0000 0000 0000 0000 0001 0000 0000	0000 0000 0000 0000 0000 0001 0000 0000	Drop Status (DU)
MCB1OFFLIN	MBC 1 Inactive	0000 0000 0000 0000 0000 0010 0000 0000	0000 0000 0000 0000 0000 0010 0000 0000	Drop Status (DU)
NORMAL	Point not in Alarm	0000 0000 0000 0000 1000 0000 1000 0000	0000 0000 0000 0000 0000 0000 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)

Status Words	**Description**	**Mask** **32 0**	**Pattern** **32 0**	**Record Type (Record Field)**
OFF0	Specified bit is OFF	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0000	OFF0 through OFF31 are applicable to any integer record field of any point type.
OFF1		0000 0000 0000 0000 0000 0000 0000 0010	0000 0000 0000 0000 0000 0000 0000 0000	
OFF2		0000 0000 0000 0000 0000 0000 0000 0100	0000 0000 0000 0000 0000 0000 0000 0000	
OFF3		0000 0000 0000 0000 0000 0000 0000 1000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF4		0000 0000 0000 0000 0000 0000 0001 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF5		0000 0000 0000 0000 0000 0000 0010 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF6		0000 0000 0000 0000 0000 0000 0100 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF7		0000 0000 0000 0000 0000 0000 1000 0000	0000 0000 0000 0000 0000 0000 000 0000	
OFF8		0000 0000 0000 0000 0000 0001 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF9		0000 0000 0000 0000 0000 0010 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF10		0000 0000 0000 0000 0000 0100 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF11		0000 0000 0000 0000 0000 1000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF12		0000 0000 0000 0000 0001 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF13		0000 0000 0000 0000 0010 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF14		0000 0000 0000 0000 0100 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	

Status Words	**Description**	**Mask** **32 0**	**Pattern** **32 0**	**Record Type (Record Field)**
OFF15	Specified bit is OFF	0000 0000 0000 0000 1000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	OFF0 through OFF31 are applicable to any integer record field of any point type.
OFF16		0000 0000 0000 0001 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF17		0000 0000 0000 0010 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF18		0000 0000 0000 0100 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF19		0000 0000 0000 1000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF20		0000 0000 0001 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF21		0000 0000 0010 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF22		0000 0000 0100 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF23		0000 0000 1000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF24		0000 0001 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF25		0000 0010 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF26		0000 0100 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF27		0000 1000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF28		0001 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF29		0010 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	
OFF30		0100 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	

Status Words	**Description**	**Mask** **32 0**	**Pattern** **32 0**	**Record Type (Record Field)**
OFF31	Specified bit is OFF	1000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	OFF0 through OFF31 are applicable to any integer record field of any point type.
ON0	Specified bit is ON	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0001	ON0 through ON31 are applicable to any integer record field of any point type.
ON1		0000 0000 0000 0000 0000 0000 0000 0010	0000 0000 0000 0000 0000 0000 0000 0010	
ON2		0000 0000 0000 0000 0000 0000 0000 0100	0000 0000 0000 0000 0000 0000 0000 0100	
ON3		0000 0000 0000 0000 0000 0000 0000 1000	0000 0000 0000 0000 0000 0000 0000 1000	
ON4		0000 0000 0000 0000 0000 0000 0001 0000	0000 0000 0000 0000 0000 0000 0001 0000	
ON5		0000 0000 0000 0000 0000 0000 0010 0000	0000 0000 0000 0000 0000 0000 0010 0000	
ON6		0000 0000 0000 0000 0000 0000 0100 0000	0000 0000 0000 0000 0000 0000 0100 0000	
ON7		0000 0000 0000 0000 0000 0000 1000 0000	0000 0000 0000 0000 0000 0000 1000 0000	
ON8		0000 0000 0000 0000 0000 0001 0000 0000	0000 0000 0000 0000 0000 0001 0000 0000	
ON9		0000 0000 0000 0000 0000 0010 0000 0000	0000 0000 0000 0000 0000 0010 0000 0000	
ON10		0000 0000 0000 0000 0000 0100 0000 0000	0000 0000 0000 0000 0000 0100 0000 0000	
ON11		0000 0000 0000 0000 0000 1000 0000 0000	0000 0000 0000 0000 0000 1000 0000 0000	
ON12		0000 0000 0000 0000 0001 0000 0000 0000	0000 0000 0000 0000 0001 0000 0000 0000	
ON13		0000 0000 0000 0000 0010 0000 0000 0000	0000 0000 0000 0000 0010 0000 0000 0000	

Status Words	Description	Mask 32 0	Pattern 32 0	Record Type (Record Field)
ON14	Specified bit is ON	0000 0000 0000 0000 0100 0000 0000 0000	0000 0000 0000 0000 0100 0000 0000 0000	ON0 through ON31 are applicable to any integer record field of any point type.
ON15		0000 0000 0000 0000 1000 0000 0000 0000	0000 0000 0000 0000 1000 0000 0000 0000	
ON16		0000 0000 0000 0001 0000 0000 0000 0000	0000 0000 0000 0001 0000 0000 0000 0000	
ON17		0000 0000 0000 0010 0000 0000 0000 0000	0000 0000 0000 0010 0000 0000 0000 0000	
ON18		0000 0000 0000 0100 0000 0000 0000 0000	0000 0000 0000 0100 0000 0000 0000 0000	
ON19		0000 0000 0000 1000 0000 0000 0000 0000	0000 0000 0000 1000 0000 0000 0000 0000	
ON20		0000 0000 0001 0000 0000 0000 0000 0000	0000 0000 0001 0000 0000 0000 0000 0000	
ON21		0000 0000 0010 0000 0000 0000 0000 0000	0000 0000 0010 0000 0000 0000 0000 0000	
ON22		0000 0000 0100 0000 0000 0000 0000 0000	0000 0000 0100 0000 0000 0000 0000 0000	
ON23		0000 0000 1000 0000 0000 0000 0000 0000	0000 0000 1000 0000 0000 0000 0000 0000	
ON24		0000 0001 0000 0000 0000 0000 0000 0000	0000 0001 0000 0000 0000 0000 0000 0000	
ON25		0000 0010 0000 0000 0000 0000 0000 0000	0000 0010 0000 0000 0000 0000 0000 0000	
ON26		0000 0100 0000 0000 0000 0000 0000 0000	0000 0100 0000 0000 0000 0000 0000 0000	
ON27		0000 1000 0000 0000 0000 0000 0000 0000	0000 1000 0000 0000 0000 0000 0000 0000	
ON28		0001 0000 0000 0000 0000 0000 0000 0000	0001 0000 0000 0000 0000 0000 0000 0000	
ON29		0010 0000 0000 0000 0000 0000 0000 0000	0010 0000 0000 0000 0000 0000 0000 0000	

Status Words	Description	Mask 32 0	Pattern 32 0	Record Type (Record Field)
ON30	Specified bit is ON	0100 0000 0000 0000 0000 0000 0000 0000	0100 0000 0000 0000 0000 0000 0000 0000	ON0 through ON31 are applicable to any integer record field of any point type.
ON31		1000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000 0000 0000	
OPATTN	Operator Attention Required	0000 0000 0000 0000 0000 0000 0001 0000	0000 0000 0000 0000 0000 0000 0001 0000	Drop Status (DU)
POOR	Quality = Poor	0000 0000 0000 0000 1000 0011 0000 0000	0000 0000 0000 0000 0000 0010 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
PRESET0	Specified bit is reset (= 0)	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 1111 1111 1111 1111	All record types.
PRESET1		0000 0000 0000 0000 0000 0000 0000 0010	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET2		0000 0000 0000 0000 0000 0000 0000 0100	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET3		0000 0000 0000 0000 0000 0000 0000 1000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET4		0000 0000 0000 0000 0000 0000 0001 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET5		0000 0000 0000 0000 0000 0000 0010 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET6		0000 0000 0000 0000 0000 0000 0100 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET7		0000 0000 0000 0000 0000 0000 1000 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET8		0000 0000 0000 0000 0000 0001 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET9		0000 0000 0000 0000 0000 0010 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET10		0000 0000 0000 0000 0000 0100 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	

Status Words	**Description**	**Mask** **32 0**	**Pattern** **32 0**	**Record Type (Record Field)**
PRESET11	Specified bit is reset (= 0)	0000 0000 0000 0000 0000 1000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	All record types.
PRESET 12		0000 0000 0000 0000 0001 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET13		0000 0000 0000 0000 0010 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET14		0000 0000 0000 0000 0100 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET15		0000 0000 0000 0000 1000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1111	
PRESET16		0000 0000 0000 0001 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET17		0000 0000 0000 0010 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET18		0000 0000 0000 0100 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET19		0000 0000 0000 1000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET20		0000 0000 0001 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET21		0000 0000 0010 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET22		0000 0000 0100 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET23		0000 0000 1000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET24		0000 0001 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET25		0000 0010 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET26		0000 0100 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET27		0000 1000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	

Status Words	Description	Mask	Pattern	Record Type (Record Field)
		32 0	32 0	
PRESET28	Specified bit is reset (= 0)	0001 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	All record types.
PRESET29		0010 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET30		0100 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PRESET31		1000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1110	
PSET0	Specified bit is set (= 1)	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 1111 1111 1111 1101	All record types.
PSET1		0000 0000 0000 0000 0000 0000 0000 0010	0000 0000 0000 0000 1111 1111 1111 1101	
PSET2		0000 0000 0000 0000 0000 0000 0000 0100	0000 0000 0000 0000 1111 1111 1111 1101	
PSET3		0000 0000 0000 0000 0000 0000 0000 1000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET4		0000 0000 0000 0000 0000 0000 0001 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET5		0000 0000 0000 0000 0000 0000 0010 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET6		0000 0000 0000 0000 0000 0000 0100 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET7		0000 0000 0000 0000 0000 0000 1000 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET8		0000 0000 0000 0000 0000 0001 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET9		0000 0000 0000 0000 0000 0010 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET10		0000 0000 0000 0000 0000 0100 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET11		0000 0000 0000 0000 0000 1000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET12		0000 0000 0000 0000 0001 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	

Status Words	**Description**	**Mask** **32 0**	**Pattern** **32 0**	**Record Type (Record Field)**
PSET13	Specified bit is set (= 1)	0000 0000 0000 0000 0010 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	All record types.
PSET14		0000 0000 0000 0000 0100 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET15		0000 0000 0000 0000 1000 0000 0000 0000	0000 0000 0000 0000 1111 1111 1111 1101	
PSET16		0000 0000 0000 0001 0000 0000 0000 0000	0000 0000 0000 0001 1111 1111 1111 1100	
PSET17		0000 0000 0000 0010 0000 0000 0000 0000	0000 0000 0000 0010 1111 1111 1111 1100	
PSET18		0000 0000 0000 0100 0000 0000 0000 0000	0000 0000 0000 0100 1111 1111 1111 1100	
PSET19		0000 0000 0000 1000 0000 0000 0000 0000	0000 0000 0000 1000 1111 1111 1111 1100	
PSET20		0000 0000 0001 0000 0000 0000 0000 0000	0000 0000 0001 0000 1111 1111 1111 1100	
PSET21		0000 0000 0010 0000 0000 0000 0000 0000	0000 0000 0010 0000 1111 1111 1111 1100	
PSET22		0000 0000 0100 0000 0000 0000 0000 0000	0000 0000 0100 0000 1111 1111 1111 1100	
PSET23		0000 0000 1000 0000 0000 0000 0000 0000	0000 0000 1000 0000 1111 1111 1111 1100	
PSET24		0000 0001 0000 0000 0000 0000 0000 0000	0000 0001 0000 0000 1111 1111 1111 1100	
PSET25		0000 0010 0000 0000 0000 0000 0000 0000	0000 0010 0000 0000 1111 1111 1111 1100	
PSET26		0000 0100 0000 0000 0000 0000 0000 0000	0000 0100 0000 0000 1111 1111 1111 1100	
PSET27		0000 1000 0000 0000 0000 0000 0000 0000	0000 1000 0000 0000 1111 1111 1111 1100	
PSET28		0001 0000 0000 0000 0000 0000 0000 0000	0001 0000 0000 0000 1111 1111 1111 1100	
PSET29		0010 0000 0000 0000 0000 0000 0000 0000	0010 0000 0000 0000 1111 1111 1111 1100	

Status Words	**Description**	**Mask** **32 0**	**Pattern** **32 0**	**Record Type (Record Field)**
PSET30	Specified bit is set (= 1)	0100 0000 0000 0000 0000 0000 0000 0000	0100 0000 0000 0000 1111 1111 1111 1100	All record types.
PSET31		1000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 1111 1111 1111 1100	
RESET	Reset	0000 0000 0000 0000 1000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0000	Long Digital (LD) Deluxe Digital (LD)
RESETALM	Reset Alarm State	0000 0000 0000 0000 1000 0000 0000 0010	0000 0000 0000 0000 0000 0000 0000 0000	Long Digital (LD) Deluxe Digital (LD)
SCANOFF	Point not Scanned	0000 0000 0000 0000 1000 1000 0000 0000	0000 0000 0000 0000 0000 1000 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
SENSORALM	Sensor Alarm	0000 0000 0000 0000 1000 0000 1000 1100	0000 0000 0000 0000 0000 0000 1000 1100	Long Analog (LA) Deluxe Analog (DA)
SENSOR-MODE	Sensor Mode	0000 0000 0000 0000 1000 0000 0001 0000	0000 0000 0000 0000 0000 0000 0001 0000	Long Analog (LA) Deluxe Analog (DA)
SET	Set	0000 0000 0000 0000 1000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0001	Long Digital (LD) Deluxe Digital (LD)
SETALM	Set Alarm State	0000 0000 0000 0000 1000 0000 0000 0010	0000 0000 0000 0000 0000 0000 0000 0010	Long Digital (LD) Deluxe Digital (LD)
TOGGLE	Toggle	0000 0000 0000 0000 1100 0000 0000 0000	0000 0000 0000 0000 0100 0000 0000 0000	Long Analog (LA) Deluxe Analog (DA) Long Digital (LD) Deluxe Digital (LD)
UPDATETIME	Update Clock	0000 0000 0000 0000 0000 0100 0000 0000	0000 0000 0000 0000 0000 0100 0000 0000	Drop Status (DU)
WORSE	Worse	0000 0000 0000 0000 1000 0000 0000 0011	0000 0000 0000 0000 0000 0000 0000 0010	Long Analog (LA) Deluxe Analog (DA)

Note that the following status word masks and patterns are exactly the same for the following pairs:

- ON4, OPATTN
- ON6, DROPFAULT
- ON7, DROPALM
- ON8, MCB0OFFLIN
- ON9, MCB1OFFLIN
- ON10, UPDATETIME
- ON15, HDWRFAIL
- OFF7, DROPCLEAR

The mask and pattern are what is stored in the gcode for any status word (as opposed to the ASCII status word). When translating a gcode (that is, when generating the updated source command after interactively moving, resizing, and so forth, of any graphic display) GBNT must determine which ASCII text to display for the pairs listed above. There is no ambiguity for any of the other status words.

GBNT uses the following rules to determine which status word to display when there is ambiguity:

1. ON15 will always translate to HDWRFAIL.
2. For a DU point (or a dummy point) paired with an FA record field, the following occurs:

   ON4 => OPATTN
   ON7 => DROPALM
   ON6 => DROPFAULT
   ON8 => MCB0OFFLIN
   ON9 => MCB1OFFLIN
   ON10 => UPDATETIME
   OFF7 => DROPCLEAR
3. For any other point name/record field pair, the following occurs:

   OPATTN => ON4
   DROPALM => ON7
   DROPFAULT => ON6
   MCB0OFFLIN => ON8
   MCB1OFFLIN => ON9
   UPDATETIME => ON10
   DROPCLEAR => OFF7

***Note:** Because the exact same mask and pattern is stored in the gcode for any of these pairs a graphic will function exactly the same no matter which status word displays in the source. It is NOT an error to use one status word as opposed to the other in the source code. Be advised that whenever GBNT has to regenerate the source from the gcode, the status words may be changed as specified by the rules above.*

INDEX