Honeywell

Experion PKS
GUS Display Scripting User's Guide

EPDOC-XX45-en-431A
February 2015

Release 431

Honeywell

Document	Release	Issue	Date
EPDOC-XX45-en-431A	431	0	February 2015

Disclaimer

Contents

1 About This Document

This document describes how to install and configure the Experion Station-TPS (ES-T) and Experion Server TPS (ESVT) nodes. The nodes become full members in Experion PKS as well as connect directly to the TPN (TotalPlant Network).

This document describes the basic concepts of this scripting language and the GUS Display Builder user interfaces for script editing. It also contains instructions for using the OLE for Process Control (OPC) Server Automation Interfaces.

2 References

The following list identifies all documents that may be sources of reference for material discussed in this publication.

Document Title	Document ID
Actors Manual	SW09-655
BasicScript 2.2 Language Reference	EPDOC-XX42-en-410
BasicScript 2.2 User's Guide	EPDOC-XX41-en-410
Display Authoring Tutorial	EPDOC-XX43-en-410
Display Builder User's Guide	EPDOC-XX44-en-410
HCI/OPC Data Access User's Guide	EPDOC-XX52-en-410
Picture Editor Reference Manual	SW09-650
SafeView User's Guide	EPDOC-X120-en-410

3 GUS Scripting

Related topics

3.1 Introduction

This section describes the basic concepts of this scripting language and describes the GUS Display Builder user interfaces for script editing.

3.2 Display Scripting Concepts

This section introduces the guiding principles of the scripting approach for GUS displays. If scripting concepts are new to you, make sure you understand the following information.

3.2.1 Common Script Syntax

GUS displays employ the same script syntax for all scripting applications within the display.

3.2.2 Scripts are associated with objects in the display or the display itself

A *script* is defined as a collection of functions, subroutines, and data declarations. A script will always be associated with a display object.

A *display object* is defined as a primitive element in a display, or the display itself. The GUS Display Builder user interface provides a mechanism for the user to define and modify the script associated with a particular display object.

3.2.3 Display objects handle events by executing subroutines within their associated scripts

The term *event* is very general.

- One class of events includes those that result from operator interaction with the display via the mouse or keyboard.
- Other events are related to process data access.
- Generally, events are occurrences to which the end user may wish to respond in a manner that can be programmed by using scripts.

Events are directed to display objects by the display runtime. Display objects handle events by executing subroutines in their associated script.

The kinds of events a display object can handle are defined by Honeywell. Each kind of event results in the execution of a subroutine with a particular name.

In this context, a subroutine is also sometimes called an *entry point* in the script.

For example, if the operator clicks on a display object with the left mouse button, the display will attempt to execute the'OnLButtonClick' subroutine in the script associated with the object on which the operator clicked.

NOTE: If a Group has an OnLButton_ or OnRButton_ script, that script will be activated only if the click is made on a member object whose selectable attribute is TRUE.

3.2.4 Scripts Access Display Element Data by Using Object Paradigm

The properties of GUS display elements are made available to the scripting language by using objects. Each display object is accessed by name from scripts.

For example, if you wish to set the text property of a text display element from within a script, you would enter the following:

```
'TextItem.Text ='Hello world'
```

- **TextItem** is the name you would give to the text object in a display.
- **Text** is the property being set.

3.2.5 Scripts Access Process Data Using Object Paradigm

Scripts refer to system or process data by using a syntax that is similar to the ‘point.parameter' syntax found in other Honeywell programming tools.

3.3 General Properties of the Scripting Language

This section introduces general properties of the GUS BasicScript language.

3.3.1 The Language is a clone of Visual Basic for Applications

Visual Basic for applications is becoming an industry standard for scripting languages. It is standard on Microsoft products and clones of it are very common in non-Microsoft products. The GUS display scripting language follows this trend. It is nearly syntactically identical to Visual Basic for Applications.

3.3.2 The Language supports access to objects

Although the language is not object-oriented in the manner of C++, it does provide access to objects such as those provided by OLE automation servers. In other words, it provides syntax for accessing objects and declaring objects, but no syntax for defining objects.

3.3.3 A Note of Caution

Visual Basic functions such as the InputBox or AskBox are based on standard Visual Basic functionality. They do not handle NaNs, and error handling is minimal.

The InputBox and AskBox are not good choices for process related entries. For example, unless you add appropriate script, selecting the CANCEL and CLOSE buttons on an AskBox will return a'0' for a numeric entry or a'null' string for a text entry. A similar situation occurs with the InputBox. This could cause a runtime error that closes your display, or send a bad value to the process that could cause a problem.

3.4 GUS Display Builder User-interface for Script Editing

This section presents the user-interface for editing GUS display scripts and the associated tools. These include:

The Menu Bar, which changes to include specific scripting features. These features are:

- The Scripting Edit Menu additions
- The View Menu containing the Properties Dialog
- The Scripting Run Menu

The Script Edit Window which contains:

- The browser bar component of the script edit window
- A Watch Variable Window
- The script editing window
- A Scripting Status Bar.

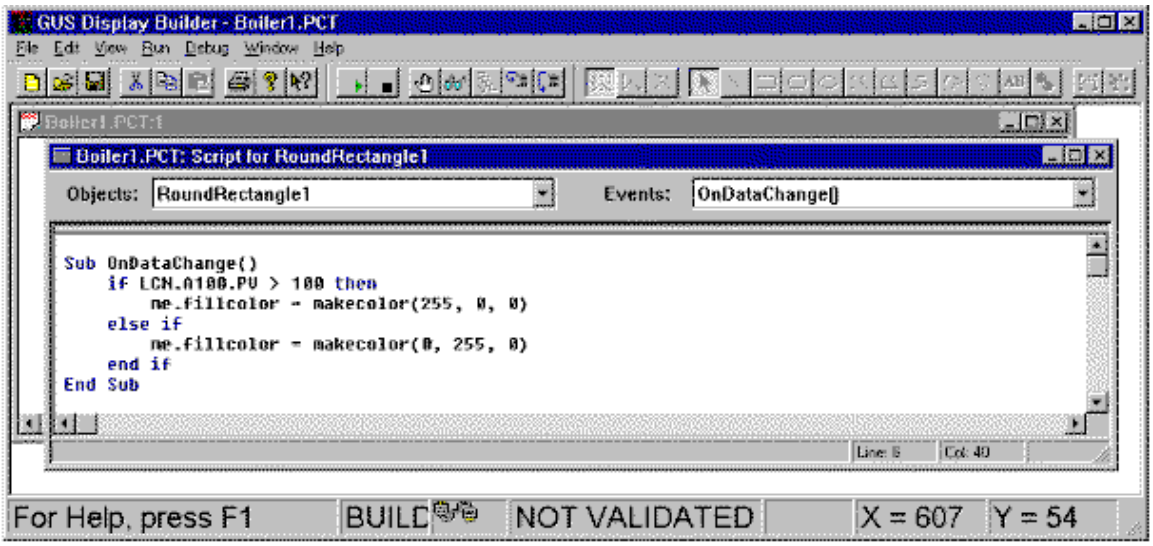
3.4.1 Script edit window

This section describes how to use the script edit window and its associated tools. The script edit window provides the following capabilities:

- Scripts can be created and modified by using a simple text editor;
- Scripts can be debugged (currently, this tool is useful only for debugging single functions that do not access LCN data);
- Templates for standard event handler subroutines can be automatically inserted, and;
- Additional script edit windows can be opened.

The script edit window is a'child' window of the main Display Builder window. The following figure shows the GUS Display Builder with an open script edit window.

NOTE: A script edit'child' window must always be closed before the main Display Builder window that it belongs to can be closed.



Detailed information on the Script Edit Window is available from:

- The browser bar component of the script edit window
- A Watch Variable Window, just below the browser bar
- The script editing window

- A Scripting Status Bar

3.4.2 Menu bar

The components of the menu bar in the following figure depend on the type of active window.

When the script edit window is active, the menu bar has the following components:

- **File** - The standard file menu items.
- **Edit** - The standard edit menu items such as cut and paste. Scripting commands are added when the script edit window is opened. For details, refer to “Scripting Edit Menu Additions” on page 30.
- **View** - The item'Properties' that causes the script property list to be visible. For details, refer to “Properties Dialog” on page 32.
- **Run** - Items related to running the script or the script compiler. For detailed information, refer to “Scripting Run Menu” on page 31.
- **Window** - Standard Window items. For details, refer to Switching Between Displays and Organizing Multiple Windows in the *Display Builder User's Guide*.
- **Help** - The standard help menu items. For details, refer to'Getting Help on the Display Builder' in the *Display Builder User's Guide*.

3.4.3 Script toolbar

The script toolbar is used only during script debugging. These icons duplicate functionality available through the menu bar. The operations include:

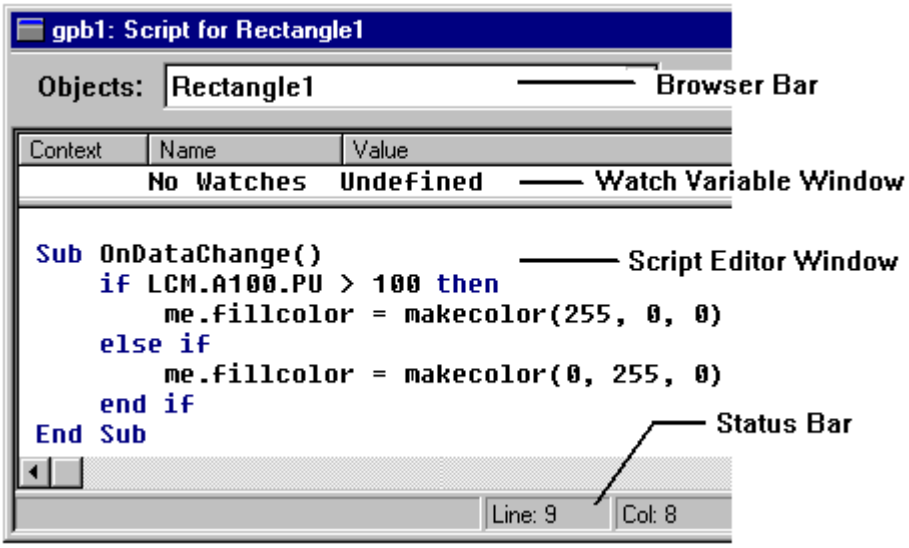
- **Start** - Begins execution of a script
- **End** - Stops execution of a script
- **Toggle Breakpoint** - Adds or removes a breakpoint on a line of BasicScript code.
- **Add Watch** - Displays the Add Watch dialog box, in which you can specify the name of a variable. That variable, together with its value (if any), is then displayed in the watch pane of Script Editing application window.
- **Calls** - Displays the list of procedures called by the currently executing script. Available only during break mode.
- **Single Step** - Each time it is clicked, it executes the next line of a script and then suspends execution of the script. If the script calls another procedure, execution will continue into each line of the called procedure.
- **Procedure Step** - Each time it is clicked, it executes the next line of a script and then suspends execution of the script. If the script calls another procedure, it runs the called procedure in its entirety without stopping.

For a complete description of these items, refer to the “Scripting Run Menu” on page 31.

3.4.4 Script editing window

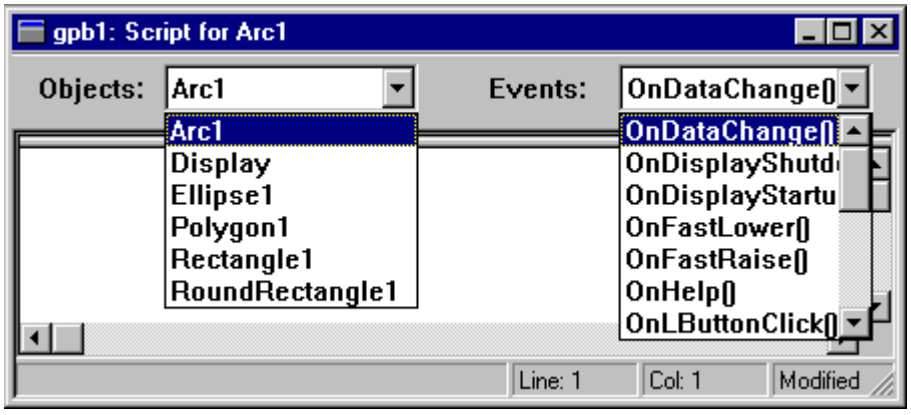
This window has three primary parts:

1. Browser Bar
2. Splitter Window consisting of
   - The Script Editor Window that displays a simple text editor for the attached object, and
   - The Watch Variable Window that displays the value of watch variables (by default, this window is closed).
3. **Status Bar** that displays the position of the insertion point within the text edit window.



3.4.5 The Browser bar in the script edit window

The browser bar is located at the top of the script edit window.



It consists of two combo boxes: Objects and Events.

- **Objects** - When the script edit window is first opened (Edit/Object Script) the first combo box displays the name of the object you have selected in your display window (Arc1 in the example). The script attached to that object, which you can edit, appears immediately below the browser bar.
- When you click on the Objects combo box arrow, all objects used in that display window are listed. If another object is selected from the list, a new script edit window opens with that object's script attached.

- Use this control to quickly find objects by name and to edit the scripts attached to them.
- **Events**- The second combo box displays the names of all possible event handlers for the object attached to the edited script.
- When you select an item from the Events drop-down list, the script editor either inserts a template for the event handler into the script, or it positions the editor at the existing template if it was previously inserted.

During runtime, various event handlers execute scripts in response to various conditions in the display objects. The kinds of events that occur for a particular display object depend on the type of object.

If the object is an OLE control, this events list includes the events that can be fired by the control to the container. For OLE events, the entries in the events list include the argument list of the event handler.

The events displayed in the *Events* combo box also define the entry point used by the debugger when execution is initiated from the Run menu item or from the Run button on the Script Toolbar.

3.4.6 Invoking a script edit window

Two different script edit windows can be invoked:

- One for the Display Script used with the full display picture, and
- One for the Object Script used with a single object in the display.

3.4.7 Invoking a Display script edit window

- Left mouse button method: From the main menu, select Edit/Display Script.
- Right mouse button method: Hold the selection pointer anywhere in the display window but not over an object, then click and hold the right mouse button. Select Edit Script and release.

RESULT: The script associated with the full display will be presented for editing.

3.4.8 Invoking an Object script edit window

- Left mouse button method: Select the desired object, and then from the Display Builder menu, select Edit/Object Script.
- Right mouse button method: With the selection pointer over the desired object, click and hold the right mouse button. Select Edit Script and release the right mouse button.

RESULT: The script associated with the selected object will be presented for editing.

3.4.9 Closing a script edit window

Click and hold the left mouse button on the icon in the left corner of the script edit window.

1	Select Close and release the mouse.
2	Alternate method—Click the close box in the right corner of the script edit window.

3.4.10 Saving a script

The script is automatically saved whenever the edit window loses focus. This occurs when:

- The script window is closed,
- Another script window is opened,
- The graphics window is selected, or
- The display is validated.

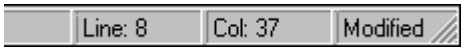
3.4.11 Printing a script

To print a script, press Ctrl-P from the active script edit window.

3.5 Scripting Interfaces

3.5.1 Status Bar

When the scripting window is opened, the Display Builder adds a scripting status bar to the script window.



The Status Bar shows this information:

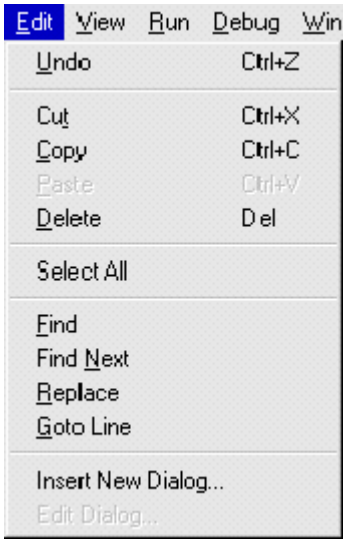
Line Number - The number of the line in your script where your insertion bar is placed.

Column Number - The horizontal position of the character that your insertion bar is on.

Status - This is the status of the script. In this example, the script has been'Modified' so the display must be validated before it is run again.

3.5.2 Scripting Edit Menu Additions

When the scripting window is opened, the Display Builder adds seven new commands to the Edit Menu.



The seven new commands are:

Select All - Selects the entire text of the script.

Find - Displays the Script Editor Find dialog box.

1	In the Find What field, specify the text you want to find.
2	Click Find Next or press Enter.
	RESULT: The first instance of the specified text is highlighted.
3	To continue searching, either press the FindNext button or select Find Next from the Edit Menu.

Find Next—Use to find the next instance of the specified text.

Replace—Displays the Replace dialog box, which allows you to substitute replacement text for instances of the specified text.

Goto Line—Presents the Goto Line dialog box, which allows you to move the insertion point to the start of a specified line number in your script.

Insert New Dialog—Invokes the Dialog Editor, which you can use to create a new dialog box for insertion into your script.

Edit Dialog—Invokes the Dialog Editor, which you can use to edit the selected dialog box template. This command is only enabled if a dialog box template is currently selected.

3.5.3 Scripting Run Menu

When the scripting window is opened, the Display Builder adds the Run Menu.

The commands are:

Start* - Begins execution of a script.

End* - Stops execution of an executing script.

Syntax Check - Verifies the syntax of the statements in your script by compiling it.

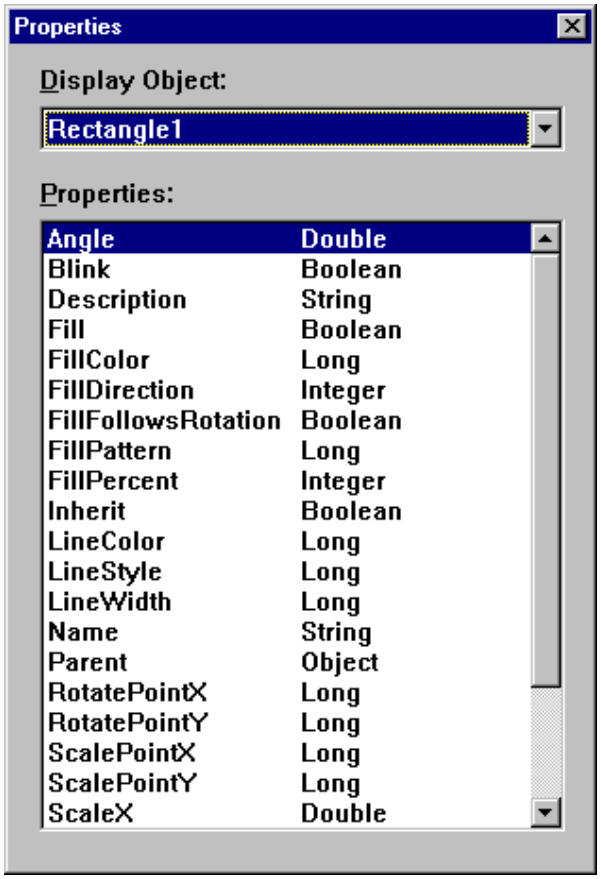
NOTES:

- If an asterisk (*) follows the command name, it is also on the Script Toolbar.
- This function applies only to the script in its own context. Consequently, it may report errors on references to display parameters and external data access.

3.6 Properties Dialog

The properties dialog box displays a list of properties for a selected object. It is useful when editing scripts to determine the properties of an object that are available from within a script.

The property dialog looks like this:



For more information, refer to “Display Object Properties” on page 122 and “OLE Controls - Properties, Methods, and Events” on page 151.

3.6.1 Description of the properties dialog

The properties dialog box consists of a combo box at the top, listing all the primitive objects and OLE control objects in the display. For each object (Rectangle1 in this example), that object's properties and associated types are listed in the properties list below.

- When another object in the combo box is selected, the properties list changes to show the properties for that object.
- If the Properties dialog box is open when you activate the script edit window for an object, the properties dialog automatically shows that object and displays its properties.

3.6.2 How to invoke the properties dialog

The properties dialog can be invoked only from an active script edit window. It can be invoked in either of the following two ways:

- Press Alt/Enter;

- Select the Properties item from the view menu.

The properties dialog is not modal. In other words, the presence of the properties dialog on the screen does not prevent you from accessing the display builder. The properties dialog is always on top and can be closed by clicking the close button on its title bar.

3.7 Relationship between Display Objects and Scripts

A display object is either a display element drawn by the user, or the display itself.

Display objects are further divided into

- *Primitive objects* that are drawn by using the drawing tools, and
- *OLE control* objects that are inserted from the insert object dialog
- *A group* of objects
- Displays *embedded* into the main display.

All objects have methods and properties that are accessible by scripts, and are the target of events that result in the execution of scripts.

This section discusses the relationship between display objects and scripts. Subjects discussed are

- Access to Display Data and Display Operations
- Display Objects Handle Events with User Defined Scripts
- Use of cached or immediate data depends on event type
- Process Data Change Event: OnDataChange Subroutine
- Periodic Execution Event: OnPeriodicUpdate Subroutine
- User-interface Events: Ex. OnLButtonDown Subroutine
- Event Propagation in Groups
- Scripting OLE Controls.

3.8 Access to Display Data and Display Operations

Scripts need to access data and invoke functions supplied by the objects that make up the display. Examples of display data that scripts will access include properties of display elements such as color and size.

Scripts access properties of display objects by using the object syntax of the Basic language. For example, to change the text displayed by a text primitive object called

Text1

to the string

Hello

the script will contain the assignment statement

Text1.text ='Hello'

'Display' object represents the display itself.

The *Display* object contains properties and methods that apply to the display as a whole. The properties of this object determine the overall appearance of the display.

3.8.1 The concept of Me

Scripts will frequently contain references to the properties and methods of the object being scripted. This is accomplished using the'me' identifier. The'me' identifier is an object that represents the script's associated object.

For example, a script attached to a rectangle, that changes the color of the rectangle's edge, contains the statement:'me.edgecolor = makecolor (150, 0, 0)'.

3.8.2 Result of simultaneous writes from both a script and a Basic Dynamic is unpredictable

It is possible to assign a value to a display object property from both an OnDataChange subroutine (described below) and a Basic Dynamic. For example, you may assign to the text property of a text object from both an OnDataChange subroutine and the expression on the value tab of the property sheet for that object. If the subroutine and the Basic Dynamic run because of the same data change event, it is not possible to predict which value the property will have at the completion of handling the data change event. The property will in fact be assigned both values, but it is not possible to predict the order in which the assignment will occur.

3.9 Display Objects Handle Events with User-Defined Scripts

At display runtime, events occur that are handled by display objects.

In the display builder, you can program a customized response to those events by using the GUS Scripting Language.

3.9.1 Subroutines called as event handlers

Each object has a predefined set of events that it can handle. To program a response to an event, create a subroutine that corresponds to the event, then place it in the script associated with a particular object.

3.9.2 Kinds of events

This is a list of the kinds of events a typical display object can be programmed to handle:

- User-interface events such as mouse clicks,
- Process data change events,
- On periodic execution events, and
- Display startup and shutdown events.

The event types are discussed in detail in Use of cached or immediate data depends on event type.

3.10 Use of cached or immediate data depends on event type

The data access subsystem provides both *cached* and *immediate* data for reads on system or process values.

- **Cached** data is read from a local copy of the data held by the data access subsystem. This data may be several seconds old, but access is very fast.
- **Immediate** reads cause access to the actual process values. All writes to system or process data are immediate. That is, they always go to the system or process and are never written to a cache. Immediate data access is slower than cached data access. See below for more details on immediate data access compared to cached data access. The type of data access that occurs for a read depends on the type of event being handled. Their behaviors are based on the following criteria:
- How frequently the event occurs, and
- Does the data have to be current.

For example, subroutines invoked in response to user-interface events need to read and write data directly from the system or process and not from a cache, to support ‘test and set' type logic.

The event kinds are discussed in more detail below. The data access behavior is described for each event kind.

3.10.1 Complete List of Events

The following tables contain a complete list of events defined by GUS Displays. The tables describe the events, describe the data access behavior of the events, and indicate the type of object that may be the target of the event.

For a comprehensive list of events for each OLE control, refer to “OLE Controls - Properties, Methods, and Events” on page 151.

Some events are discussed in more detail below.

Name	Description	Data Access Behavior	Object Type
OnPeriodicUpdate	Executed periodically in half-second steps. See below for more details on this event.	Read from cache. Write directly to LCN. Wait for data access completion before continuing after a write.	All
OnDataChange	Execute when referenced LCN data changes. See below for more details on this event.	Read from cache. Write directly to LCN. Wait for data access completion before continuing after a write.	All
OnDisplayStartup	Execute when display starts.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a write.	All
OnDisplayShutdown	Execute when display is about to exit.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a write.	All
OnEnterFocus	Execute when an object within this group has become the item of focus. Note: Will not execute when focus is returned to the display from another window.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a write.	Group and Embedded displays
OnExitFocus	Execute when the focus within the display moves from an object within this group to another object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a write.	Group and Embedded displays

Name	Description	Data Access Behavior	Object Type
OnGotFocus	Execute when the object has become the item of focus **within** the display. Note: Will not execute when focus is returned to the display from another window.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a write.	Primitive objects and controls
OnLostFocus	Execute when the object has ceased to be the item of focus **within** the display. Note: It does not execute when focus is returned to the display from another window.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a write.	Primitive objects and controls
OnLButtonDown	Execute when left mouse button is pressed while cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnLButtonUp	Execute when left mouse button is released while cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnLButtonClick	Execute when left mouse button is pressed and released while cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnLButtonDblClick	Execute when left mouse button is double-clicked while cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnRButtonDown	Execute when right mouse button is pressed while cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnRButtonUp	Execute when right mouse button is released while cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnRButtonClick	Execute when the right mouse button is pressed and released while the cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnRButtonDblClick	Execute when right mouse button is double-clicked while cursor is over object.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive, Group, and Embedded displays
OnPageForward	Execute when the PAGE FWD key is pressed by the operator.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnPageBackward	Execute when the PAGE BACK key is pressed by the operator.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnDisplayForward	Execute when the DISP FWD key is pressed by the operator.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnDisplayBackward	Execute when the DISP BACK key is pressed by the operator.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnHelp	Execute when the HELP key is pressed by the operator.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display

Name	Description	Data Access Behavior	Object Type
OnAssociate	Execute when the ASSOC DISP key is pressed by the operator.	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnACK	Execute when the ACK key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnClear	Execute when the MSG CLEAR key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnConfirm	Execute when the MSG CONFIRM key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnDetail	Execute when the DETAIL key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Main display
OnRaise	Execute when the raise key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive object and main display. If no object is selected, the event is sent to the main display.
OnFastRaise	Execute when the fast raise key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive object and main display. If no object is selected, the event is sent to the main display.
OnLower	Execute when the lower key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive object and main display. If no object is selected, the event is sent to the main display.
OnFastLower	Execute when the fast lower key is pressed by the operator	Read directly from LCN. Write directly to LCN. Wait for data access completion before continuing after a read or write.	Primitive object and main display. If no object is selected, the event is sent to the main display.

3.11 Process Data Change Event: OnDataChange Subroutine

The subroutine in a display object script with the name'OnDataChange' will be executed when a change is detected in the value of one or more of the data items it references. The triggering of the OnDataChange subroutine is designed for efficient monitoring of process variables.

The rules for DataChange detection are:

- Changes in scripting variables made by another script (declared as Public or Dim) never activate an OnDataChange subroutine.
- Data items that are referenced within the OnDataChange subroutine only as the target of an assignment never activate an OnDataChange subroutine.
- Changes in TPN variables and collectors (example: lcn.A100.MODE) are detected by comparing values on each Data Collection cycle, at the rate configured for that item.
- Changes in the Display DataBase and Display Parameters (example: dispdb.INT01 and display.params.P1) are detected as soon as they are made by any scripted statement.

CAUTION
INFINITE LOOPS

This means that an OnDataChange subroutine that contains a statement such as'dispdb.INT01 = dispdb.INT01+1' will execute as an infinite loop.

- Only changes in data items directly referenced in the body of the OnDataChange subroutine script will cause it to execute. Data references made in functions or other subroutines called from within the OnDataChange subroutine WILL NOT affect the activation of the subroutine.

CAUTION
IMMEDIATE DATA REQUESTS

On Button Click scripts, requesting immediate data from the ACKSTAT collector may return out-of-date information. Collectors are always read at the collection cycle. Immediate reads are performed from the cache only.

For example, when a script does an alarm acknowledgment immediately followed by an

object.text = collector('ACKSTAT(pointname)'), it will not produce ACK status if the cache still shows unacknowledged status.

3.11.1 OnDataChange Subroutine Data access behavior

The display data access subsystem executes the affected OnDataChange subroutines as data changes.

The OnDataChange subroutine reads data from the cache maintained by display data access.

Writes to data are allowed in OnDataChange subroutines, but they are not recommended. Because writes are time consuming, they may prevent the display from keeping up with incoming OnDataChange events.

3.11.2 OnDataChange Subroutine Usage Guidelines

The intended use for the OnDataChange subroutine is to make displays dynamic, based on changing data values. It provides an efficient mechanism for making displays dynamic because it runs only when the relevant data changes, rather than using the less efficient polling approach.

An OnDataChange subroutine should execute quickly to allow the display to keep up with rapidly changing data. You should avoid performing time-consuming operations in an OnDataChange subroutine.

Examples of time-consuming operations are as follows:

- File input and output,

- Interaction with the operator via dialogs or message boxes, and
- Loops that process large amounts of data.

CAUTION

REFERENCING DATA FROM MULTIPLE SERVERS

If you include an expression in your OnDataChange script that references data from more than one server, the script will execute as soon as data is returned from the *first* server. At this time, the variant data type for variables from other servers may still be'unknown', and this will trigger a Type Mismatch error report. You should, therefore, always include an explicit error handler in your script, to prevent displaying these Type Mismatch error messages during normal startup.

3.12 Periodic Execution Event: OnPeriodicUpdate Subroutine

The subroutine in a display object script with the name'OnPeriodicUpdate' is called periodically, on the period configured for the display. In other words, each display object will be sent a periodic event, whose period is configured in the display. When it receives the periodic event, it will invoke the'OnPeriodicUpdate' subroutine in its associated script.

3.12.1 OnPeriodicUpdate Subroutine Data access behavior

The OnPeriodicUpdate subroutine starts at the same time as the start of data collection. It reads process data from the cache, which is initialized by the scanned OnDataChange data.

Occasionally OnPeriodicUpdate reads a cache item before the cache data has been read from the TPN. When this happens, the script waits up to three seconds for the process data to arrive. If the process data does not arrive in that time, the error'property or method not found' is reported.

To avoid this situation, add an'OnError' clause to your OnPeriodicUpdate script, similar to the one that follows. This error handler will be activated when the scan data is late, displaying question marks for the item of the display until the data arrives.

Sample Script

```
Sub OnPeriodicUpdate
On Error Goto Errorlabel
Me.text = lcn.a100.pv
Exit Sub
Errorlabel:
Select case err.number
Case 423
‘Property or method not found code
Me.text ='?????'
Case Else
Me.text ='!!!!!'
End Select
End Sub
```

3.12.2 Writes to data not recommended in OnPeriodicUpdate

Writes to data are allowed in an OnPeriodicUpdate subroutine, but they are not recommended. Because writes are time consuming, they may prevent the display from keeping up with incoming OnPeriodicUpdate events.

3.12.3 OnPeriodicUpdate Subroutine Usage Guidelines

The intended use for the OnPeriodicUpdate entry point is to support simple types of display animation or to support types of data access that require polling. In general, it is recommended that you use the OnDataChange entry point to produce dynamic displays based on process data because it is more efficient than polling for changes via the OnPeriodicUpdate entry point.

The OnPeriodicUpdate event occurs frequently. OnPeriodicUpdate subroutines must execute quickly, or the display update will fall behind. You should avoid performing time-consuming operations in an OnPeriodicUpdate subroutine. Examples of time consuming operations are file input and output, interaction with the operator via dialogs or message boxes, and loops that process large amounts of data.

3.13 User-interface Events: Ex. OnLButtonDown Subroutine

User-interface events result from operator interaction with the display. The most common events correspond to mouse usage. There will be a subroutine for each user-interface event. An example is OnLButtonDown for a left mouse button press. Other examples include OnRButtonDown and OnLButtonDoubleClick.

In Windows in general, these kinds of events are handled by the window of input focus. The main program of the display runtime will receive a message from Windows indicating that the user has performed some operation on its window. The main program will perform'hit testing' to determine whether or not the operation was performed on a primitive display object. If it was, the main program will determine if the object has an event handler subroutine defined for the user's operation. If it has, the subroutine will be executed.

3.13.1 Ex. OnLButtonDown Subroutine Data access behavior

All data reads and writes in user-interface event handlers are immediate. This supports'test and set' type logic common to this type of subroutine.

3.13.2 List of User-interface Events

The following table lists all of the user-interface events.

This table summarizes information with respect to user-interface events found in the section titled'Complete list of events' in Use of cached or immediate data depends on event type:

Name	Description	Object Type
OnLButtonDown	Execute when left mouse button is pressed while cursor is over object	Primitive, Group, and Embedded displays
OnLButtonUp	Execute when left mouse button is released while cursor is over object	Primitive, Group, and Embedded displays
OnLButtonClick	Execute when left mouse button is pressed and released while cursor is over object	Primitive, Group, and Embedded displays
OnLButtonDoubleClick	Execute when left mouse button is double-clicked while cursor is over object	Primitive, Group, and Embedded displays
OnRButtonDown	Execute when right mouse button is pressed while cursor is over object	Primitive, Group, and Embedded displays
OnRButtonUp	Execute when right mouse button is released while cursor is over object	Primitive, Group, and Embedded displays
OnRButtonClick	Execute when the right mouse button is pressed and released while cursor is over the object	Primitive, Group, and Embedded displays
OnRButtonDoubleClick	Execute when right mouse button is double-clicked while cursor is over object	Primitive, Group, and Embedded displays
OnPageForward	Execute when the PAGE FWD key is pressed by the operator	Main display
OnPageBackward	Execute when the PAGE BACK key is pressed by the operator	Main display
OnDisplayForward	Execute when the DISP FWD key is pressed by key is pressed by the operator	Main display
OnDisplayBackward	Execute when the DISP BACK key is pressed by the operator	Main display
OnHelp	Execute when the HELP key is pressed by the operator	Main display

Name	Description	Object Type
OnAssociate	Execute when the ASSOC DISP key is pressed by the operator	Main display
OnACK	Execute when the ACK key is pressed by the operator	Main display
OnClear	Execute when the MSG CLEAR key is pressed by the operator	Main display
OnConfirm	Execute when the MSG CONFIRM key is pressed by the operator	Main display
OnDetail	Execute when the DETAIL key is pressed by the operator	Main display

3.13.3 More than one event is generated on a single'button click'

Be aware that multiple events occur on button clicks when scripting an object.

For instance, three events (LButtonDown, LButtonUp, and LButtonClick) are generated in sequence when a single'Left Button Click' is done on an object.

In the same way, seven events are generated in sequence when a'Left Button Double-Click' is done on an object. In the order of their occurrence, these events are: LButtonDown, LButtonUp, LButtonClick, LButtonDown, LbuttonClick, LButtonUp, and LButtonClick.

3.14 Event Propagation in Groups

This section describes the effect of grouping on the propagation of events to display object scripts. Events propagate to grouped objects as follows:

- User-interface events propagate to the object that was the target of the operator interaction.
- The OnDisplayStartup event propagates through the objects contained in groups. The OnDisplayStartup handler in the group object itself is executed before propagating the event to the objects in the group.
- The OnDisplayShutdown event propagates through the objects contained in groups. The OnDisplayShutdown handler in the group object itself is executed after propagating the event to the objects in the group.

3.15 Scripting OLE Controls

This section discusses how to script an OLE control.

Scripting an OLE control embedded in a GUS display is very similar to scripting a primitive object. The display associates a script with the OLE control the same way as a primitive object. The edit window is invoked in the same way, the concept of entry points applies, and there is a *me* object in the script that represents the associated OLE control.

The properties of an OLE control that are accessible from scripts are determined entirely by the control. GUS Basic uses the mechanism of OLE automation to access these properties. The properties accessible from scripting can be listed by using the properties dialog invoked from the script editor.

The set of possible event handlers in the script for an OLE control is a combination of events that can be supported by the control to the container and the standard events generated by the display runtime. The standard events that apply to an OLE control are OnDisplayStartup, OnDisplayShutdown, OnHelp, and OnPeriodicUpdate.

The event handlers for events supported by the control are subroutines that are invoked by the display runtime in response to the firing of an event from the control to the display. The event handler concept in GUS Basic enables the user to program a response to a control event. For example, a button control might fire a'Click' event to the container when the operator presses and releases the left mouse button with the cursor over the control. The programmer can program a response to the ‘Click' event by including a subroutine named'Click' in the script attached to the control. If the event has arguments, the subroutine in the script must declare arguments that match those of the supported events. The Browser bar on the script edit window shows the possible events of the OLE control (in the'Methods' drop-down list) and can be used to insert a template event handler into the script.

3.16 Differences - Basic Scripting Vs. GUS Scripting

3.16.1 Behaviors

There are occasional differences in the behaviors of subroutines created using the Summit Basic Scripting Language when executed on ES-T. Therefore, the use of some keywords in Basic Scripting Language will not produce the same results on a GUS display as they would on other computers.

Tip
All variables that are passed to a subroutine are passed by reference (ByRef), no matter how they are defined in the script.

Two optional keywords,'ByVal' and'ByRef' do not affect scripting behavior as you might expect from other programming environments. Here is what happens: If you create a subroutine that assigns a calculated result to the parameter as an intermediate step and a numeric is passed as the'ByRef' constant, it will validate and run without reporting an error. Also, specifying the parameter as'ByVal' does not prevent the subroutine from altering its value.

3.16.2 Which Basic Scripting Language keywords are affected

The following Basic Scripting Language keywords do not behave as expected when executed on the GUS display.

Basic Scripting Keyword	Behavior
Sub MySub	If ‘Sub MySub' is configured in the Display script and in an object (Text3, for example), when a different object (Text2, for example) calls MySub, it gets the version defined in the last object to start (Text3 in this example) instead of the version ‘inherited' from the Display.
Dim a (type)	You can only specify Explicit object types for data objects and for early bound OLE automation objects (e.g.: Objects whose type libraries have been registered with BasicScript). Example that will fail: ‘Dim a As Excel.Application'. The correct way to do this is: ‘Dim a As object'. Then set a = createobject(‘excel.application').
Net.Browse$(1)	Type 1 of Net.Browse$, which is suppose to allow the user to browse the network's printer queues, does not work.
Command$	The ‘Command$' scripting function will always return a null string.
PopupMenu	If you display a pop-up menu by executing ‘PopupMenu' in the script, you cannot cancel the pop-up menu.
Msg.Open	The syntax is described as: Msg.Open *prompt,timeout,cancel,thermometer [,Xpos,Ypos],* But within our multi-threaded use, the'timeout' argument has no effect. In the following example, the Sleep 3000 statement actually governs the length of time before the dialog box is removed. `Sub Main() Msg.Open'Printing. Please wait…',0,True,False Sleep 3000 Msg.Close Msg.Open'Printing. Please wait…',0,True,True For x = 1 to 100 Msg.Thermometer = x Next x Sleep 1000 Msg.Close End Sub`
AppGetState	The AppGetState command fetches the wrong integers for the Maximized and Minimized states.

Basic Scripting Keyword	Behavior
Resume (Next)	A portion of the description in the BasicScript Language Reference manual says that the error trap ‘Resume Next' will clear any previous error branching and'tell' the program to continue execution of the program, even if an error is encountered. Actually, if ‘Resume Next' is included on the ‘On Error' line in a labeled error handling routine, it does NOT clear the previous error branching.

3.16.3 DefaultButton argument

The DefaultButton argument allows you to specify which button is the default. However, this argument has no effect within the ES-T environment.

3.17 Multithreaded Script Execution

The display runtime is designed to ensure that the display is always updated with current information, within the limits of the computer. The handling of user-interface events and execution of periodic update scripts should not interfere with updating the display. To accomplish this, scripting uses three threads of execution, each of which operates at a lower than normal priority. The main user-interface thread, operating at normal priority, does not execute scripts directly but dispatches the request to a script thread. The threads are summarized in this table and are described in the text that follows.

Thread Name	Purpose
Data change	Execute OnDataChange events
User-interface event	All user-interface events run on this thread
Periodic update	Execute OnPeriodicUpdate events

3.17.1 Data Change thread

Scripts handling an OnDataChange event are run on the data change thread. Data change events are dispatched by the data access subsystem whenever a change in the data values referenced in the OnDataChange handlers of the script is detected.

3.17.2 User-interface event thread

Scripts handling any of the user-interface events are run on the user-interface thread. All mouse button clicks and keystroke event handlers are dispatched on the user-interface thread.

3.17.3 Periodic Update thread

The periodic update thread is used to run the OnPeriodicUpdate entry points of all scripts. This thread repeats the following two steps during the life of the display execution:

1	Wait for a timer event. These events occur at the fixed periodic update rate of the display. This rate is currently set at a half-second and cannot be changed by the user.
2	Execute all the OnPeriodicUpdate entry points in all scripts, one at a time. Start with the main display script. Then execute the display object scripts in the order in which the objects were placed into the display.

If the thread fails to complete execution of all OnPeriodicUpdate entry points before the next timer event occurs, that event will have been queued by the system. There is no preemption if the periodic update thread falls behind. There can be at most one timer event on the queue at a time. If there is already a timer event in the queue when a new timer event arrives, the new event will be discarded.

3.17.4 Scripts blocked when any are executing an extension routine

The scripting package is conservative in its support for multithreading. When any script is executing an extension function, all other scripts are blocked from running until the extension function returns. Implementers of extension functions can choose to override this behavior if they determine their functions to be multithread safe.

3.18 Script Capabilities Based on Display Mode

This section discusses the capabilities of scripts in the various modes of operation available in GUS Displays. The two display modes in which scripts operate are

- Display is in build mode
- Display is operating in run mode

The primary difference in capability between the modes is the accessibility of LCN data.

3.18.1 Script capabilities when display is in build mode

When the display is in build mode (that is, you are editing a display), scripts can be executed from the script edit window. Use the toolbar buttons or menu items to control script execution. See the information in Script Edit Window for more details. When scripts are executed in this way they function normally in most respects, but they cannot access actual LCN data. When an access is attempted, either an error is displayed or the script will execute with artificial data.

The behavior of a script when accessing an LCN value while debugging depends on whether or not the display has been validated:

- If the display has not been validated, an error will occur when you attempt to access LCN data. The error is reported in a dialog box created by the debugger, and the script terminates. The error message states that an unknown property of method of an object was accessed. This is a trappable runtime error (number 423).
- If the display has been validated (either off line or on line), an attempt to access LCN data while in the debugger does not cause an error. If you attempt to write to LCN data, the statement does nothing. If you attempt to read an LCN parameter, an integer value is returned that is a count of the number of attempted reads since the display edit session began. Any attempt to read a property of a parameter returns an undefined integer value.

The script debugger is currently useful only for debugging single functions that do not access LCN data.

3.18.2 Script capabilities when display is in run mode

A display in run mode processes all event types (user-interface, periodic update and data change) and can access LCN data if connected to an LCN data server. In other words, all script capabilities are available when a display is in run mode.

3.19 Break Handling

This section discusses the use of <Ctrl-Break> by the operator to terminate script execution.

3.19.1 Break handling in the script edit window of the display builder

When using the script debugger in a script edit window of the display builder, entering <Ctrl-Break> will stop script execution. If the script is displaying a dialog, the dialog is removed from the screen.

Use the <Ctrl-Break> sequence if you have inadvertently written a script containing an infinite loop.

3.19.2 Break handling when running a display

When a display is in run mode, break handling is only supported during display shutdown (that is, after the operator has selected the Close button). Entering <Ctrl-Break> at this time causes an immediate termination of any scripts that may be running, and the immediate closing of the display's window.

Typically, only the display shutdown scripts are running during display shutdown. An operator may need to use <Ctrl-Break> if a display inadvertently contains an infinite loop in its scripts, or if the display shutdown sequence is taking too long and the operator must terminate prematurely.

3.20 Scope of Rules in GUS Display Scripts

This section discusses the scope rules for data, subroutines, and functions declared in GUS display scripts. Scope rules determine the visibility of these items across and within scripts.

3.20.1 Variables

GUS Basic does not extend or modify the scope rules supplied by BasicScript for Make the following selections from the Configuration Utility menu variables. You can use the BasicScript Private and Public statements to declare private and public variables.

3.20.2 Subroutines

GUS Basic does not extend or modify the scope rules supplied by BasicScript for subroutines. The standard BasicScript rules for declaring public and private subroutines apply.

3.20.3 Display Object names

Display object names are global to all scripts. No DIM statements are needed or allowed to access the name of a display object. All display object names are automatically added to the global name space. Display object names must conform to the following lexical rules:

- Must begin with an alphabetic character
- Can include alphabetic characters, numbers, exclamation points, or underbars
- Cannot be greater than 24 characters in length

These rules are enforced by the Display Builder when you enter a name for the object in the Properties dialog box.

3.21 Expression Evaluation and Type Checking

This section summarizes the rules for expression evaluation and type checking in GUS Basic.

Note that GUS Basic is a loosely typed language, as compared to other Honeywell programming language products such as CL.

GUS Basic expression evaluation corresponds to standard BasicScript expression evaluation.

3.21.1 Expression Evaluation

GUS Basic allows expressions to involve data of different types. When this occurs, the two arguments are converted to the same type by promoting the less precise operand to the same type as the more precise operand. For Example, GUS Basic will promote the value of int1 to a Double in the following expression:

```
dim int1 as integer
dim dbl1 as double
result# = int1 * dbl1
```

In some cases, the data type to which each operand is promoted is different than that of the most precise operand. This is dependent on the operator and the data types of the two operands, as defined for each BasicScript operator.

If an operation is performed between a numeric expression and a String expression, the String expression is usually converted to the same type as the numeric expression. For example, the following expression converts the String expression to an Integer before performing the multiplication:

```
result = 10 *'2'          | ‘Result is equal to 20
```

This rule does not apply to all BasicScript operators.

GUS Basic performs numeric type conversion automatically. Automatic conversions can result in overflow errors, as shown in the following example:

d# = 45354

i% = d#

In the above example, an overflow error will be generated because the value specified for d# is greater than the maximum size of an Integer.

3.22 Access to Process Data from Scripts

This section presents an end user view of process data access from GUS Basic.

Subjects discussed are

- Process data access in GUS Basic uses an object paradigm
- Properties and methods of data access objects
- The Display Database (DispDB)
- Enumeration Type Parameters
- Access to TPS Network Collector Data
- Immediate and Cached Data
- Validation of Point References: Off-line versus On-line Building
- Bad Status Examples
- Enumerations in TPS Network pictures vs. GUS displays

3.23 Process data access in GUS Basic uses an object paradigm

Process data access in GUS Basic follows an object paradigm. For GUS basic programmers this means that points are treated like built-in objects, and parameters are handled as properties on those objects. A major benefit of this approach is that the syntax for accessing point data in GUS Basic is similar to the syntax found in other Honeywell products, such as CL or the TPS Network picture editor. The fundamental difference between GUS Basic and previous products is the greater extent to which the object paradigm is applied. The differences are

- There are additional objects in GUS Basic,
- The objects have additional properties, and
- Some data that would be accessed via functions in other products is accessed via properties or methods of an object.

The benefits of this approach are

- The end user has a more consistent view of process data, and
- LCN data access, with its large name space, can be cleanly integrated into the name space provided by Basic.

3.23.1 Built-in objects provide access to LCN data

GUS Basic provides the following built-in objects to provide access to LCN data:

- LCN - This object represents an LCN. Points can be accessed only as properties of the LCN data object.
- DispDB - This object represents the display database.
- Collector - This object represents the functionality of the TPN Picture's collectors, which provide access to certain system data.

These objects are visible in all scripts. Each of these objects will be discussed in more detail below.

3.23.2 Points are accessible as properties of the LCN object

LCN points are represented as objects in GUS Basic. Point objects are directly accessible only as properties of the LCN object (and indirectly accessible as properties of the DispDB object). Furthermore, parameters are represented as objects and can be accessed only as properties of a point object or DispDB object. For example, to read the value of a point and parameter named ‘A100.PV' and assign it to the text property of a text object named text1, use the LCN object as follows:

```
text1.text = lcn.a100.pv
```

Because a point is an object in GUS Basic, it can have properties. Generally, the only properties of a point are its parameters. Because a parameter is treated as an object in GUS Basic, it too can have properties. The most commonly accessed property of a parameter will be its value. Additional properties include its status, type and various value representations for enumeration-type parameters. In general, the parameter object is where GUS basic differs most from previous languages. Where previously a function was provided to access data about a parameter, GUS basic provides a property.

3.23.3 Default properties simplify object references

The default property of an object is the property that will be used when the user does not explicitly specify a property when referencing the object. For example, a parameter has several properties, including the value property. The value property is the default. A reference of the form LCN.A100.PV, accessing the PV parameter of the point A100, will access the value property of the PV parameter. This reference is a short cut for LCN.A100.PV.VALUE because it takes advantage of the default property of the parameter object.

Default properties are chosen to simplify references to the most common property of an object.

3.23.4 Using LCN names

When you are using LCN name forms, for example, lcn.a100.pv, and then referencing a name form (such as lcn.pv.pv2), where'pv.pv2' is the parameter name, will produce a'type mismatch' syntax error. Use the [ ] escape mechanism to resolve this error. For example, using lcn.a100.[pv.pv2] will not produce an error.

3.23.5 Accessing LCN names with invalid characters or that are duplicates of built-in properties

Point and parameter names can contain characters that are not valid in GUS Basic. Usage of these names as property names on either the LCN object or a point object will result in a compilation error. GUS Basic provides an escape mechanism that allows for property names that contain invalid characters. To reference LCN names that include invalid characters, enclose the name in brackets. This example accesses a point named $a100 with a parameter named !parm1:

me.text = lcn.[$a100].[!parm1]

This escape mechanism can also be used to access LCN point or parameter names that are duplicates of the built-in properties of the GUS basic data access objects. This example accesses the name parameter of a point named A100:

me.text = lcn.a100.[name]

3.23.6 Correct use of evaluate operator in LCN references

When using the evaluate operator to enable access from scripts to points and parameters containing special characters, be careful to enclose only one piece of the name form in the operator at a time. Enclosing both the point and parameter name in one evaluate operator will sometimes result in errors on stores. For example:

lcn.[$nml1b31].fl(64) ‘Correct usage

lcn.[$nml1b31.fl(64)] ‘Incorrect usage

3.23.7 Evaluate property

The bracket syntax described here is an alternative method for accessing a distinguished object property called *evaluate.*

Evaluate is a property that accepts one string argument. Evaluate is treated as a reference to a property of the object with the name contained in the string. The bracket example shown above could be rewritten as:

me.text = lcn.a100.evaluate('name')

3.23.8 LCN data types mapping to Basic data types

Care should be taken when manipulating LCN data within scripts. LCN data types must in some cases be mapped into the appropriate Basic data types. This table describes in general how these types would map.

LCN Data Type	Basic Data Type
Integer	Integer
Real	Single or Double
Boolean	Boolean
String	String
Enumeration	String
SD Enumeration	String

LCN Data Type	Basic Data Type
Date/Time	Date
Unknown	String

3.23.9 Testing the configuration of an entity

An entity type parameter or DDB location can contain an invalid (non-existing) entity or can be empty. Empty means that it does not contain an entity reference. The reference to a parameter of either the invalid or empty entity will cause the script to invoke the error handler with an err.number code 1052. Most scripts do not need to differentiate between an empty or invalid entity reference.

For example:

```
Sub OndataChange
On error goto ErrHandler
Me.text = lcn.a100.entpar.pv
Exit sub
ErrHandler:
Select case err.number
Case 1052:
‘ Configuration error (entity is invalid or empty)
Me.text ='#####'
Case…:
.
.
.
Case else
Me.text ='?????'
End case
End sub
```

CAUTION

If for some reason a different action needs to be taken when the entity is empty, a test of the external property of the entity is possible. Ensure that the statement is executed by exception (for example, in an error handler). The drawback of this approach is that accessing the external property in a script will cause an immediate read of that value. Ensure that in the'normal' case at display startup, the entity is initialized with a valid entity. Proliferation of this approach throughout a display outside an error handler can cause a severe degradation of the call-up speed of the display.

An example of an error handler that would provide this functionality follows:

```
ErrHandler: Select case err.number Case 1052: ‘ Configuration
error (entity is invalid or empty) If lcn.a100.entpar.external =''
then … Else Me.text ='#####' End if Case…: . . . Case else Me.text
='?????' End case End sub
```

3.24 Properties and Methods of Data Access Objects

This section provides detailed information about the data access objects.

Related topics

3.24.1 LCN object

An *LCN* object represents an LCN. An LCN object, named LCN, is built-in to GUS basic. Points are accessible by referencing a point property of the built-in LCN object. The name of a point property is the name of the point on the LCN. If the point name includes a gateway name, the point object is located on a foreign LCN. The only properties of the LCN object are points.

3.24.2 POINT object

A *point* object is used to represent LCN data points. Point objects can appear only as a property of the LCN object. The only properties of a point object are the parameters that appear on that point. The name of a point object is the name of the point it represents on the LCN. If the point name does not conform to the lexical rules for identifiers in GUS Basic, use the *evaluate* property described earlier.

If a point is accessed by using a network gateway, its name must contain the network gateway identifier. to specify the network gateway identifier, precede the point name with the network gateway identifier, followed by a slash character (/). Because of the slash character, you must use the evaluate property, described earlier, to escape the restrictions on Basic identifiers. The following example shows how to assign the value of the *pv* parameter of point *a100* on network gateway *ng* to the text property of an object.

me.text = lcn.[ng\a100].pv

3.24.3 PARAMETER object

A *parameter* object is used to represent parameters of LCN points. These objects are accessible only as a property of a point object. For every parameter on the point represented by a point object, there is a property of the same name that is a parameter object.

A parameter object has several properties. One property makes available the value of the parameter, while others provide status and type information. The default property of a parameter is its value.

3.24.4 Enumeration Examples

Notice the'Enum' function potentially eliminates the confusion regarding the size of the Set and Member number but requires the user to know the LCN Set and Member name.

Some examples for accessing, modifying and comparing LCN point.parameters of type enumeration are shown below. They are not case-sensitive and should not be considered the only correct formats. In some cases, alternate formats are shown that accomplish the same task. We recommend you select the simplest form. Assume a text object is used.

NOTE: If you use the Value page of the text object, the Type field should be set to enumeration even if the access specification includes the'internal' property.

3.24.5 Standard and Custom Enumeration

Read Access using a text object and an LCN object

me.text = lcn.point.parameter

me.text = lcn.point.parameter.External

me.text = lcn.point.parameter.Internal

Using the Enum function to display all or a portion of a Standard or Custom Enumeration

This function removes some of the confusion regarding datatypes discussed previously however, the user must know the LCN SetName and MemberName to use this function.

Determining the SetNumber:

me.text = enum('SetName:MemberName').SetNumber

me.Text = enum('BoxColor:Red').SetNumber

Determining the SetNumber and Member Number:

me.text = enum('SetName:MemberName').Value

me.Text = enum('BoxColor:Red').Value

Determining the MemberNumber:

me.text = enum('SetName:MemberName').MemberNumber

me.Text = enum('BoxColor:Red').MemberNumber

DDB usage: (We do Not recommended that the .External form be used for the DDB)

To access the Internal form use:

DispDB.Enm01.Internal = lcn.point.parameter.Internal

me.text = DispDB.Enm01.Internal

To access the External form use:

DispDB.Enm01.Internal = lcn.point.parameter.Internal

me.text = DispDB.Enm01.External

Modifying an LCN enumeration value using an LCN object

lcn.point.parameter ='SetMember'

lcn.A100.pv ='Open'

lcn.point.parameter = enum('SetID:SetMember')

lcn.B100.pv = enum('BoxColor:RED')

lcn.point.parameter.Internal = enum('SetID:SetMember').Value

lcn.B100.pv.Internal = enum('BoxColor:RED').Value

lcn.point.parameter = enum('SetID:SetMember').Value

Comparison

if lcn.point.parameter ='SetMember' Then …

if lcn.a100.pv ='Open' Then …

DDB

if DispDB.Enm01.External ='SetMember' Then …

if DispDB.Enm01.External ='Open' Then …

if DispDB.Enm01 = lcn.point.param.External Then …

if DispDB.Enm01.External = lcn.point.param.External Then …

if DispDB.Enm01.External = lcn.point.param.Value Then …

if DispDB.Enm01.Internal = lcn.point.param.Internal Then …

3.24.6 Self Defining Enumeration

Read Access using a text object and an LCN object

me.text = lcn.point.parameter

me.text = lcn.SD100.pv

me.text = lcn.point.parameter.External

me.text = lcn.SD100.pv.External

DDB usage

Do NOT assign a Self Defining Enumeration to the DDB

Modifying an LCN enumeration value using an LCN object

lcn.point.parameter ='SetMember'

lcn.SD100.pv ='LOWER'

Comparison

if lcn.point.parameter ='SetMember' Then …

If lcn.SD100.pv ='UPPER' Then

if lcn.point.parameter.External ='SetMember' Then …

If lcn.SD100.pv.External ='UPPER' Then

3.24.7 Properties of a Parameter Object

The following table describes the properties of a parameter object:

Property	Return Type	Arguments	Description
Value(default)	Varies	None	This is the value of the parameter.
Status	Long Integer	None	The data access status of this parameter.
NodeType	Long Integer	None	The HOPC-type code for the parameter.
Type	Long Integer	None	The HOPC-type code for the parameter reference. This will be the same as the Node Type for all parameters except those of type variable. For variable types, NodeType will return HOPC_TYPE_variableId (9), while Type will return the type of the variable referenced by the parameter.
External(default for Get accesses)	String	None	This property is valid only for parameters of enumeration or entity type. It references a string representing the enumeration set member for enumeration types, or the name of the entity for entity types. If this property is accessed on another type, the value property is returned.

Property	Return Type	Arguments	Description
Internal(default for Put accesses)	Long Integer - For standard and custom enumeration parameters. Short Integer - For self defining enumeration parameters. String for entity parameters.	None	This property is valid only for parameters of enumeration or entity type. It references a value containing an internal representation of the parameter. For standard and custom enumerations, the internal representation consists of the enumeration set number in the high word and the member number in the low word. For self-defining enumerations, it is an integer containing the member number. For entities, it is a string containing a packed binary representation of the internal entity ID. If this property is accessed on another type, the value property is returned.
Name	String	None	This property is valid only for parameters of variable type. It will return an error if accessed on any other parameter type. This property is the external name of the variable referenced by this parameter.
ID	String	None	This property is valid only for parameters of variable type. It will return an error if accessed on any other parameter type. This property is a string containing a packed binary representation of the internal variable ID of the variable referenced by the parameter.

3.25 Display Database (DispDB)

This section discusses how to access the display database from GUS Basic.

3.25.1 Writing to DispDB Values

GUS scripting software does not prevent users from writing into the dispdb values (such as $CONNUM) that are identified for READONLY use. These values are used to relay TPN configuration data to GUS scripts. These dispdb values were designed be used as reference data only, and changing them could cause misrepresentation of the nodes condition.

3.25.2 DispDB object provides support for translated displays

The DispDB object in GUS displays is provided to enable translation of TPS Network pictures into GUS displays. If you are not familiar with this TPS Network concept, refer to the Actors Manual for more information on the display database.

3.25.3 TPS Network DDB items are properties of the DispDB object

In the TPS Network picture editor, the DDB appeared to the end user as a collection of built-in variables. The naming of DDB variables indicated their type and contained a number to distinguish the name from others of the same type (for example, INT01 is local DDB integer number 1). This approach is used in GUS Basic, except that the collection of names appears as properties of the DispDB object, rather than in the global name space. For example, to reference the first integer DDB item in the local DDB, enter DispDB.INT01.

3.25.4 Added DDB items

In GUS displays, the DDB has been extended to handle more items than are available in TPS Network pictures. There are now 256 items, in both the global and local DDB, for each data type. The initialization policy, however, for the new items is different than it is for the items in the original range. The initial value of the items in the new range is undefined, while the items in the old range are initialized according to the rules described in the Actor's Manual.

3.25.5 Meaning of Local and Global for GUS displays

The DispDB retains the concepts of local and global DDB. Items with a trailing'g' in their names are in the global DDB.

In the TPS network, the terms local and global refer to the level of persistence possessed by a DDB item. The values in the local DDB of the TPS Network exist only during the life of the display. They are reinitialized for each new display. The values in the global DDB persist across picture invocations. They are initialized when the station starts up. The global DDB can be used to pass data from one picture to the next.

In both TPS the network and GUS pictures, the values of local DDB items persist only during the life of a display invocation. They are initialized by the data access system when the display starts. Refer to “Added DDB items” on page 62 for information on differences in initialization values between old and new range DDB items.

The global DDB in GUS is quite different than it is in the TPS Network because there can be more than one display running on the station. The global DDB in the GUS Display Builder is designed to allow data to be passed between the invoked display and the invoking display when using the *InvokeDisplay* statement. That is, the global DDB of a display invoked by using InvokeDisplay is initialized to the values in the DDB of the invoking display. Note that the global DDB is not shared across displays in the station, rather it is simply initialized differently when the display is invoked by using *InvokeDisplay*. Note also that *InvokeDisplay* relies

on the display manager to handle copying DDBs. If the display manager is not active, *InvokeDisplay* does not process the global DDB.

3.25.6 DispDB Properties

The following table lists the available properties of the DispDB object:

Item	Type	Description
INT01 - INT256	Integer	Storage for integer values in the local storage
INT01G - INT256G	Integer	Storage for integer values in the global storage
REAL01 - REAL256	Float	Storage for floating point numbers in local storage
REAL01G - REAL256G	Float	Storage for floating numbers in global storage
BOOL01 - BOOL256	Integer	Storage for Boolean values in the local storage
BOOL01G - BOOL256G	Integer	Storage for Boolean values in the global storage
STRING01 - STRING256	string	Storage for string values in the local storage
STR01G - STR256G	string	Storage for string values in the global storage
ENT01 - ENT256	point	Storage for a reference to a point in local storage.
ENT01G - ENT256G	point	Storage for a reference to a point in global storage.
ENM01 - ENM256	enumeration	Storage for enumeration values in the local storage. Only standard and custom enumerations can be stored here.
ENM01G - ENM256G	enumeration	Storage for enumeration values in the global storage. Only standard and custom enumerations can be stored here.
VAR01 - VAR256	variable	Storage for a reference to a point.parameter in local storage. Refer to “Variable-type DISPDB indirection” on page 64 for more details.
VAR01G - VAR256G	variable	Storage for a reference to a point.parameter in global storage. Refer to “Variable-type DISPDB indirection” on page 64 for more details.
DATIME1 - DATIME256	time	Storage for time values in the local storage. See Note 1 below.
DATIME1G - DATIME256G	time	Storage for time values in the global storage. See Note 1 below.
$AL_ENTY	variable	An entity variable that provides the name of a point that is selected on the Alarm Summary, Unit Alarm Summary, Alarm Annunciation, or Organizational Summary displays. See **Attention** below.
$ALMCOLR	variable	An integer that specifies the alarm priority color option selected in the NCF: 0 = two color option (red, yellow), 1 = 3 color option (user selected colors)
$ARAID	variable	An 8-character string name for the current area.
$ARAID01 - $ARAID10	variable	An 8-character string name for each indicated area as configured in the NCF.
$ARADSC	variable	A 24-character string description of the current area.
$ARADS01 - $ARADS10	variable	A 24-character string description for each indicated area as configured in the NCF.
$CONDSC	variable	A 24-character string description of the current console.
$CONDS01-$CONDS10	variable	A 24-character string description of the indicated console as configured in the NCF.
$CONNUM	variable	An integer that represents the current console number.

Item	Type	Description
$CZ_ENTY	variable	An entity variable for displaying a Change Zone. For more information, refer to Change Zones in the *Display Builder User's Guide*.
$EALMCLR	variable	An integer that specifies the Emergency Alarm priority color option selected in the NCF.
$GRPBASE	variable	An entity ID that can be used to access the group definitions.*
$HALMCLR	variable	An integer that specifies the High Alarm priority color option selected in the NCF.
$KEYLEVL	variable	An enumeration that contains the internal keyswitch level for this US. The values are: VIEW, OPR, SUP, and ENGR.
$LALMCLR	variable	An integer that specifies the Low Alarm priority color option selected in the NCF.
$MY_AREA	variable	An integer that represents the area number on which the custom display is running.
$MY_PNA	variable	An integer that represents the node number on which the custom display is running.
$REDYEL	variable	An enumeration that contains the minimum alarm priority indicated by the color red. It contains the members LOW, HIGH, and EMERGENCY.
$STNNUM	variable	An integer that represents the station number on which the custom display is running.

Note 1: Note that the DATIMEn and DATIMEnG objects are different and consequently provide different times. The local DDB (DATIMEn) contains the startup time from the display that invoked it. The global DDB (DATIMEnG) contains the station startup time.

Attention
The GUS Alarm Summary is not supported on Experion nodes.

3.25.7 DISPDB indirection

DISPDB indirection in GUS Basic is similar to TPS Network Pictures. In the TPS Network picture editor, the concept of accessing data using references is known as indirection.

There are two types of DISPDB indirection:

- Variable
- Entity

3.25.8 Variable-type DISPDB indirection

Variable-type DISPDB items (such as DISPDB.VAR01) are used to contain a reference to a parameter on a particular point. A reference is like an alias. For example, if DISPDB.VAR01 is set to reference A100.PV, reading the value of DISPDB.VAR01 is the same as a reading the value of A100.PV. Generally, variable-type DispDB items have the properties of parameter objects. Refer to' “Properties of a Parameter Object” on page 60' in Section “Properties and Methods of Data Access Objects” on page 58 for more details.

When a VAR item appears in an expression or alone on the left-hand side of an assignment statement, the contents of the VAR item are de-referenced: it accesses the contained point.parameter.

Variable IDs are now generally bound at buildtime. There are multiple user syntax expressions available for Variable IDs. The preferred expression is as follows:

SET Dispdb.var01 = lcn.a100.pv

In this form, the set keyword identifies that a reference to the parameter is being made, and the right hand side of the expression is the object being referenced.

The functional form of the reference is also provided.

The *GetVar* function can be used to generate a reference that can be stored in a VAR item using a SET statement. The function is defined as follows:

Dim GetVar(Name as string) as object

The argument Name is the name of the variable, for example A100.PV.

The function returns an object that a DispDB.Var object can be set to reference.

If the argument is a constant string such as'A100.PV' the reference is bound at buildtime.

If the argument is a variable string as in

GetVar(Askbox$('EnterVariable'))

the reference is bound at runtime.

When setting a VAR item in a SET statement, use the GetVar function on the right-hand side of the assignment to generate the reference to be stored in the VAR item.

The GetVar function is useful when you wish to access a point.parameter whose name was input from the operator.

The following example shows the use of the set statement with VAR items:

set DispDB.var01 = GetVar ('a100.pv')	‘ make DispDB.var01 reference a100.pv

The following examples show the use of the VAR items as references to a point.parameter

‘ Assume for the following that DispDB.var01 is set to a100.pv

DispDB.var01 = 3DispDB.var01 = 3	‘ assign a value to a100.pv
me.text = DispDB.var01	‘ Text item will show the value of a100.pv

Additional examples of variable-type expressions follow:

set dispdb.var01 = getvar('A100.num1')	OK
set dispdb.var01 = lcn.A100.num1	OK
set dispdb.var01 = lcn.A100.foo	Buildtime error: Invalid Name
set dispdb.var01 = display.params.p1	where p1 is of variable type, OK
set dispdb.var01 = display.params.p1	where p1 is of entity type, Runtime error 13 - type mismatch
dim o as object; set o = lcn.A100.num1; x = o	OK (does not trigger a data change script)
set dispdb.var01 = lcn.A100.num_par(i)	OK - Runtime bound (slower) if i is not constant
set dispdb.var01 = dispdb.var02	OK
set dispdb.var01 = GetVar(askbox('enter variable ID'))	OK - Runtime bound (slower)

Attention

- A known deficiency is that an indexed indirect reference like the following is not supported:
  - set dispdb.var01 = lcn.A100.ent_par(i).num1
- Support for the function GetVarID has been deleted.
- Unless you add appropriate script, selecting the CANCEL and CLOSE buttons on an AskBox will return a'0' for a numeric entry or a'null' string for a text entry. This could cause a runtime error that closes your display, or send a bad value to the process that could cause a serious process problem.

3.25.9 Entity-type DISPDB indirection

Entity IDs are now generally bound at buildtime. There are multiple user syntax expressions available for Entity IDs. The preferred expression is as follows:

SET Dispdb.ent01 = lcn.a100

In this form, the set keyword identifies that a reference to the entity is being made, and the right hand side of the expression is the object being referenced.

The functional form of the reference is also provided. The GetEnt function is new to be consistent with the GetVar function. In this form, the lcn identifier string is provided as an argument to a function that returns the referenced object, as follows:

SET Dispdb.ent01 = GetEnt('a100')

If the argument is a constant string such as'a100' the reference is bound at buildtime.

If the argument is a variable string as in

GetEnt(Askbox$('EnterEntity'))

the reference is bound at runtime.

Additional examples of entity-type expressions follow:

Set dispdb.ent01 = GetEnt('A100')	OK
Set Dispdb.ent01 = lcn.A100	OK
set dispdb.ent01 = lcn.A100.num1	Runtime Error 1058
set dispdb.ent01 = lcn.foo	Buildtime error: Invalid Name
set lcn.A100.ent_par(2) = GetEnt('A100')	OK
set lcn.A100.ent_par(Dispdb.int01) = GetEnt('A100')	OK
dim o as object; set o = lcn.A100; x = o.pv	OK (does not trigger data change script)
set dispdb.ent01 = lcn.A100.ent1	OK
dispdb.ent01 = lcn.A100.ent1	OK (ent01 and ent 1 reference the same entity)
set dispdb.ent01 = display.params.p1	where p1 is entity type, OK
set dispdb.ent01 = display.params.p1	where p1 is variable type, Runtime error 13 - type mismatch
set dispdb.ent01 = dispdb.ent02	OK
set dispdb.ent01 = GetEnt(askbox('enter entity ID'))	OK Runtime bound (slower)
set dispdb.ent01 = lcn.A100.ent_par(i)	OK Runtime bound (slower) if i is not constant

Double indirection is also possible: an entity-type parameter may contain a reference to an entity-type parameter. For example:

SET lcn.a100.ent1 = lcn.a101	A100 has an ent1 parameter of type entity, set it up to reference a101.
SET Dispdb.var01 = lcn.a100.ent1	Set the ent item to reference a100.
me.text=dispdb.ent01.ent1.num10	The value of a101.num10 is displayed.

Attention

The old syntax for entity IDs is also still supported:

Dispdb.ent01 ='a100' or Dispdb.ent01.external ='a100'

This syntax is not checked nor bound at build time and will run slower! It is recommended that this form not be used for new displays and that old displays using this form be re-written to use the new expression.

CAUTION

Unless you add appropriate script, selecting the CANCEL and CLOSE buttons on an AskBox will return a'0' for a numeric entry or a'null' string for a text entry. This could cause a runtime error that closes your display, or send a bad value to the process, which could cause a serious process problem.

3.26 Enumeration-type Parameters

This section presents how to access enumeration-type parameters in GUS Basic. Additional information on this topic is available in Enumerations in TPS Network pictures vs. GUS displays elsewhere in this document.

3.26.1 Properties of the PARAMETER object provide access to enumeration data

GUS Basic provides full access to the data contained in an enumeration-type parameter. This is accomplished via properties of a PARAMETER object that are available only for enumeration-type parameters. The following properties provide access to enumeration data:

- External - Returns a string representing the enumeration set member for the current value of the parameter;
- Internal - Returns a number representing the enumeration value. If the enumeration is a standard or custom enumeration, this is a long integer containing the set id in the upper half of the number, and the member id in the lower half of the number. If it is a self defining enumeration, it contains only the member number;
- Value - The value property is synonymous with the External property on Get references and Internal property for Put references.

3.26.2 Behavior of Assignment involving enumeration-types

Assignment to the properties of enumeration-type parameters has the following behavior:

- Internal property - The behavior of assignment depends on the kind of enumeration:
  - Standard enumerations - the assigned value is treated as a long integer. The high-order half of the integer is interpreted as the enumeration set ID. If the assigned enumeration set does not match that of the parameter-type, a runtime error that can be trapped will be raised. The lower order half of the word is treated as the member number. If it is not a valid member number for the enumeration set, a tradable runtime error will be raised.
  - Custom Enumerations - the assigned value is treated the same as for standard enumerations.
  - Self-defining enumerations - the assigned value is treated as an integer containing the member number. If it is not a valid member number for the parameter, a tradable runtime error will be raised.
- External property - The type of this property is String. The assigned value must contain a valid member name for the enumeration set of the parameter. A tradable runtime error will be generated if the assigned string is not valid for the enumeration set. Because of LCN limitations, a runtime error will always be raised if the external property of a self-defining enumeration is written to.

3.26.3 Behavior of Comparisons involving enumeration-types

Comparisons for equality involving the properties of enumeration-type parameters have the following behavior:

- Internal property - The comparison is simply an integer comparison. Because the number contains both the set ID and the member ID (only the member ID for self-defining), the comparison is safe.
- External property - The type of this property is String and so the comparisons behave in the normal manner for strings. Because the comparison does not include the set ID, it is somewhat less safe than using the Internal property.

3.26.4 Enumeration constants for standard and custom enumerations

GUS Basic provides a built-in function, *enum that* returns an enumeration value given a string input argument. An *enumeration value* is defined as a long integer containing a set number in the upper-half of the word and a member number in the lower half. The input argument of the enum function is a string of the form:'<setname>:<membername>', where *setname* is a standard or custom enumeration set name, *membername* is a member of the specified set, and ‘:' is a delimiter between the names.

The returned value from the *enum* function can be used in assignments and comparisons with the internal property of standard- or custom-type enumeration parameters. For example:

if lcn.a100.mode.internal = enum('mode:auto') then

The enum built-in function actually returns an enumeration constant object whose default property is an enumeration value. The enumeration constant object returned from *enum* has the following properties:

Property	Type	Description
Value (default)	long	An enumeration value that can be used in assignments and comparisons with the internal property of a standard- or custom-type enumeration parameter.
SetNumber	short	The set number of the enumeration value.
MemberNumber	short	The member number of the enumeration value.
SetName	string	A string containing the enumeration set name.
MemberName	string	A string containing the enumeration set member name.

The following is an example showing usage of some of the properties of enum constant object:

msgbox'setnumber for mode is' & enum('mode:auto').setnumber

Use string literals for self-defining enumerations

Self-defining enumerations do not have a special literal form or enum function. Simply use a string literal in assignments or comparisons with the External property of the parameter. For example:

if lcn.a100.sdparm ='on' then...

3.27 Access to TPS Network Collector Data

This section describes how to access TPS Network collector data. Collector data is provided by the LCN data server in addition to the point and DispDB data described in the Display Database (DispDB). Collector data includes alarm and history information. Refer to the TPS Network Picture Editor Reference Manual for more information on collectors.

3.27.1 Collector function provides access to collector data

Collector data is accessible from GUS displays by using the *collector* function. It accepts one argument that specifies the desired collector data, and returns the value of the specified collector. The input argument must be a string literal containing a collector and its argument list as specified in Appendix H of the TPS Network Picture Editor Reference Manual.

Notes:

1. This string literal cannot contain any spaces, otherwise a validation error will occur.
2. Use the double quote (") character as an escape character when a double quote character is required within the string literal. For example

   collector('$UNITSTS("01")')

   NOTICE: This is equivalent to the TPS Network syntax: $UNITSTS ('01')
3. You cannot parameterize the argument of a collector. For example, because the type of the return value depends on the specified collector, you cannot script the following:

   collector('ACKSTAT(dispdb.params.pointr)')

   The following example shows the proper use of the ACKSTAT collector,

   ackvalue = collector('ACKSTAT(A100)')
4. The TPS Network references within the collector string literal must not contain GUS prefixes DispDB or lcn.

   CORRECT:

   WRONG:

   collector('$PNTSTS(A100)')

   collector('$PNTSTS(lcn.A100)')

   CORRECT:

   WRONG:

   collector('$ADDBCNT(STRING01,ENM01)')

   collector('$ADDBCNT(DispDB.STRING01,DispDB.ENM01)')

   CORRECT:

   WRONG:

   collector('HOUR_V(A100.PV,3,2,1)')

   collector('HOUR_V(lcn.A100.PV,3,2,1)')
5. Use quotes (with each quote escaped with another quote) around all references to Primod Names, Unit Ids, and Annunciator Group Titles.

   collector('$PRIMCNT("PRIMPNT",$ALRMSTS:UNACKHI)')

   collector('$PRIMSTS("PRIMPNT")')

   collector('$UNITCNT("01",$ALRMSTS:ACKEM)')

   collector('$ANNSTS("GRP001")')

6. **Do not put collectors in the OnDisplayStartup script.** A runtime error will occur if a collector is used in an OnDisplayStartup script.
7. Parameter Start Time - Some History Collectors have a parameter start time. The value entered for the Parameter Start Time is not checked for its validity. Therefore, it can display wrong data.

   Valid values for Parameter Start Time are 0-3.

3.27.2 CollectHistory function initiates processing of history collectors

To initiate collection of history data, you must call the CollectHistory function from within a script. This is typically done in a script handling an operator event, after the operator has entered all the entities and parameters that the history collectors need. Refer to the “CollectHistory (statement)” on page 91.

3.28 Immediate and Cached Data

This section explains immediate and cached data access.

3.28.1 Cached and Immediate Data Access Defined

Immediate data access is performed directly to the LCN. A request for immediate data access waits for completion before performing the next action in the script. Immediate data is always current, but there are substantial performance implications because there can be a large time delay as the data is read or written to the LCN. The concept of immediate data access applies to both data reads and data writes.

Cached data access is performed to a cache maintained by the data access subsystem on the NT side of the system. Cached data access applies only to data reads.

3.28.2 Type of data access is determined by type of event

The type of data access that is performed for a given data read or write is determined by the type of event being handling by the executing script. Cached data access is always performed when the script is handling an OnDataChange or OnPeriodicUpdate event. Immediate data access is performed when the script is handling all other events.

3.28.3 Data writes not recommended during OnDataChange or OnPeriodicUpdate events

Data writes should be avoided in a subroutine that handles an OnDataChange or OnPeriodicUpdate events. Because data writes are always immediate, they are too time-consuming when handling these events.

3.29 Validation of Point References: Off-line versus On-line Building

This section describes the concept of data access validation.

3.29.1 Two phases of validation

There are two phases of data access reference validation. The first phase validates the syntax of the script, including the syntax of data access references. During the first phase, the display builder records information about all data access references made in the script. This information is used in the second phase of validation. In the second phase, the existence of all referenced LCN data is validated and the system performs work to ensure the fast call up of the display at runtime.

The first phase of validation occurs when any of the following actions are performed by the end user:

- Close a script edit window;
- Select the syntax check menu item when using the script edit window;
- Run the script by using the debugger in the script edit window.
- Execute any of the commands that do the second phase.

The second phase of validation occurs when the user selects one of the following menu items:

- File/validate and save
- File/validate and save as
- Display/validate

When one of the validation menu items is selected, all scripts in the display are compiled before performing the validation. Compilation consists of syntax checking and generation of an efficient internal representation for script execution. This provides two benefits:

- The user is enabled to find scripts that may have been saved with errors.
- Changes to the objects in the display may have invalidated existing scripts. These errors are reported at validation time, rather than runtime.

3.29.2 Error reporting during the second phase of validation

When the user selects one of the menu items that initiate display validation, the system may detect errors. The errors are listed in a dialog that provides an indication of the object in error, a message describing the error and the location in the script where the error occurred. This list of errors will contain errors that occur both during the compilation of the scripts and during the LCN data reference validation.

3.29.3 Full or Incremental Validation

If the Full Validation entry in the Display menu is checked, all LCN data references are recompiled using the internal values in the connect TPN system. If this entry is not checked, the validation accepts any previously validated references without recompilation. This can provide a significant reduction in validation time when small changes are made to an existing display. However, if the display was validated against a different TPN system or if some of the points referenced in the display were deleted and rebuilt, incremental validation can result in a display with improper data access references.

3.29.4 Unvalidated display containing data access references cannot be run

A display that contains data access references and has not successfully completed phase two of validation cannot be run.

3.29.5 Off-line versus On-line building

Displays can be built either off-line or on-line. Off-line and on-line refer to the availability of a connection to the LCN. *On-line* means that the node on which the display is being edited is connected to the LCN on which the display will run. *Off-line* means that the display is not connected to the target LCN.

When building in off-line mode, the user can refer to LCN objects in scripts and the first phase of validation can be performed. The user must be on-line in order to perform the second phase of display validation.

The menu item display/validate is always available. When the menu item is selected while editing in off-line mode, all scripts are compiled, but no LCN data access validation is performed. Any pre-existing LCN data access references are maintained, so a display can be revalidated and saved from the off-line builder as long as no LCN data access references have been added. When selected in on-line mode, both compilation and validation occur. Compilation can be useful when in off-line mode to verify that no scripts in the display contain syntax errors.

3.30 Default Error Handling

Default error handling is provided on Reads of LCN data (whether referenced in scripts or in Value properties). In these cases, the Data Access Error, is reported in the'.status' attribute of the referenced value (ex: lcn.a100.pv.status), and the displayed value is updated appropriately.

The following status codes may appear:

	.status	Displayed Numeric Value	
HOPC_NO_ERROR	0	actual LCN value	
HOPC_COMMUNICATION_ERROR	1	?????	[data owner shut down]
HOPC_CONFIGURATION_ERROR	2	@@@@@	[if bad entity id.]
		!!!!!	[if bad parameter id.]
HOPC_VALUE_ERROR	3	-----	[BadValue or NaN]
HOPC_STORE_ERROR	4	!!!!!	[Node initializing or indirection to entity that does not have the matching parameter]
HOPC_DYNAMIC_INDEX_ERROR	6	^^^^^	[index not resolved yet]

Note: The above do not apply to Display Data Base (dispDB), calculated, or public variables.

The following default error handling applies to formatting numeric values for display (regardless of source):

Displayed Numeric Value	
#####	[calculation error]
*****	[overflow in formatting]

3.31 Explicit Error Handling for Data Access

Scripts may provide explicit error handling (using the ON ERROR statement) to trap and resolve runtime problems with data values. Within the Error handling routine, the error code is identified in the ERR.NUMBER variable. The following ERR.NUMBER codes may be encountered while running a correct script, depending on the values of referenced data:

ERR.NUMBER	Description	Sample Cause
5	Invalid Procedure Call	Invalid date input from user
6	Overflow	Calculated value outside range for variable
11	Division by Zero	Denominator in calculation = 0
13	Type Mismatch	Variable input from user doesn't match intended data type
1051	COMMUNICATION	Data owner not accessible
1052	CONFIGURATION	Invalid Entity or Variable input from user
1053	VALUE_ERROR	Enumeration.Internal specifies invalid member number
1054	STORE_ERROR	Data Owner in the process of initializing
1056	DYNAMIC_INDEX	Array Index not resolved
1057	NO_ERROR_NO_DATA	Enumeration.Internal specifies invalid set number
1058	Varied by cause	Store block by LCN validation/security handling
-214352566	OLE Automation Error	Integer Overflow (int01 = 123456)
-2147319765	OLE Automation Error	dispdb.ENT02.external ='BADPOINT'

3.32 Bad Status Examples

This section discusses bad status handling in GUS Basic.

3.32.1 Options: Direct access to parameter status property or use an error handler

In GUS Basic, you have two options for dealing with parameters that contain bad status. You can access the parameters status property directly and program some action based on that information or you can use an error handler to trap references to parameters that contain bad values. The approaches can be combined based on the end user's needs. Each method is described in more detail below.

3.32.2 Parameter Status defined

Status is a property of a parameter value obtained from the LCN. Status indicates the success of the data access and the goodness of the data that was obtained. Status is said to be *bad* when the status value indicates that the parameter data is not valid or is compromised some way.

3.32.3 Handling bad status option 1: Direct access to parameter status

Bad status can be handled by directly accessing the status property of a parameter. For example, to obtain the status of A100.PV the end user would write A100.PV.STATUS. The data-type of the *status* property is a long integer. Refer to “Common Status Errors” on page 107 for status values to compare with actual status values. The following code fragment shows an example of access to parameter status in a script associated with a text object:

```
if lcn.a100.pv.status = HOPC_NO_ERROR then
me.text = lcn.a100.pv
else
me.text ='??????'
end if
```

3.32.4 Handling bad status option 2: Bad Status as a Runtime Error Condition

The Basic language provides a general facility for handling runtime error conditions. GUS Basic enables end users to use this facility for handling access to parameter data that has bad status. It does this by raising a trappable runtime error when an access is made to a parameter value that has a bad status. The end user can deal with trappable runtime errors by writing an error handler. An error handler is also known as an error trap. An error handler is a component of a subroutine.

An error handler is set up by using the *on error* statement. This statement activates and deactivates an *on error* handler. The error handler itself is simply a labeled block of code in the body of the subroutine in which the *on error* statement occurs. The standard BasicScript On Error (statement) functionality applies.

The following example shows an OnDataChange subroutine containing an error handler:

NOTE: an offset of 1050 is required to access error messages.

```
Sub OnDataChange()
On Error Goto Error_Handler
me.text = dispdb.ent01.pv
Exit Sub
Error_Handler:
select case err.number
case 1051
‘communication error
me.text ='=====''
case 1052
‘configuration error
me.text ='#####'
case 1053
‘value error
```

```
me.text ='-----'
case 1054
‘store error
me.text ='+++++'
case else
Msgbox Err.Description +' Error Number is ' + CStr(Err.Number)
end select
End Sub
```

3.32.5 Summary: Bad status always raises a trappable error

A reference to the value property of a parameter with bad status will always raise a trappable runtime error. You can avoid the error by first checking the status property before accessing the value property, or you can program for the occurrence of the trappable error by placing an error handler in the subroutine.

3.33 Enumerations in TPS Network pictures vs. GUS displays

3.33.1 Summary of TPS Network enumeration-type

The TPS Network supports three types of enumeration: Standard, Custom, and Self-Defining. Standard and Custom enumerations are handled identically from the end user perspective.

An enumeration-type is a set. It has two components: a set name and the set members. An enumeration value specifies a particular enumeration set name and a particular set member within that set.

Standard and self-defining enumerations are different in several ways:

1	Standard and custom enumerations are not associated with a point. These types are known by the system based on their enumeration set name. Self-defining enumerations are associated with particular Digital and Flag points and are defined by certain parameter values on those points.
2	Standard and custom enumerations are essentially numeric in nature. When a comparison between enumerations of this type is performed, it is usually a numeric comparison. Self-defining enumerations are essentially strings. When a comparison between enumerations of this type is performed, they are typically string comparisons. These distinctions are not fixed. The end user can write scripts that access enumerations as either as strings or numbers, but this seems to be the natural way the different enumeration-types are handled in the TPS Network picture editor.

3.33.2 Summary of enumeration handling in TPS Network Pictures

The TPS Network picture editor supports comparisons between enumeration-types and assignments to enumeration-type parameters (in Actors). Note that there are substantial differences between enumeration handling in actor language compared to the other scripts that can appear in a TPS Network picture (conditions, variants, expressions, etc.).

The TPS Network picture provides syntax for specifying an enumeration constant. An enumeration constant is a literal that represents an enumeration value. It has the form'set name:set member', (such as MODE:AUTO). In some contexts, the set name and colon can be omitted. The set name in an enumeration constant for a self-defining enumeration is always'SD'.

3.33.3 Comparison of TPS Network Picture Enumerations with GUS BasicScript

The following is list of differences between enumerations in TPS Network pictures and GUS BasicScript:

- Enumeration literals
  - GUS BasicScript does not provide an enumeration literal. It supports this functionality via the *enum* built-in function.
  - There is no special syntax or access function for a literal of type self-defining enumeration in GUS BasicScript. Use a literal string for these values (for example A100.PV = SD.OPEN becomes LCN.A100.PV ='Open' in GUS BasicScript)
- Access to enumeration information

  The TPS Network uses intrinsic functions and actor functions to expose various components and views of enumeration-types. This is accomplished by using the following parameter object properties in GUS BasicScript.
  - The ‘External' built-in function in the TPS Network becomes the external property of the parameter in GUS BasicScript. For reads and stores, the external property is synonymous with the value property.
  - The ‘Internal' built-in function in the TPS Network becomes the internal property of the parameter in GUS BasicScript.
- Indexing by enumeration is not supported in GUS BasicScript.

3.34 Scripting of Embedded Displays

This section presents topics related to scripting in embedded displays. Subjects discussed are

- General Embedded Display Information
- Embedded Display Parameters
- General Guidelines for Value Type Display Parameter Usage

3.35 General Embedded Display Information

3.35.1 Embedded displays are like user-defined objects

The embedded display feature of GUS Displays enables the user to effectively construct new kinds of display objects. In other words, when a display is embedded, it acts like an object. It's like an object because it encapsulates reusable functionality and because it exposes a public interface via properties. In this case, the properties are the user-defined display parameters. Also, an embedded display is like an object because its internal components are not visible to the containing display, nor are the containing display's objects visible to the embedded display. This kind of encapsulation enhances reusability.

3.35.2 Scope rules

The scope of an embedded display's object names, public variables, or public functions is confined to the immediately containing display. When a display is embedded, its objects, variable, and functions are not visible to scripts in the containing display. Likewise, the objects, variables, and functions in the containing display are not visible to scripts in an embedded display.

3.35.3 What happens to main display script when display is embedded

A display can have a main display script, as described in the Multi-threaded Script Execution section of this document. The main display script is associated with the display itself, rather than with a particular display object. When a display is embedded, its main display script is retained, but it is not visible from the containing display. Entry points of the main display script of an embedded display are executed in the scope of the embedded display.

The entry points associated with the operator keyboard are run only in the outermost main display script. Other entry points, such as OnPeriodicUpdate and OnDataChange are run in all display scripts, including the main display script of an embedded display.

3.35.4 Script thread usage

The scripts in embedded displays share the threads of the main display. Scripts are dispatched for execution in the same manner regardless of whether they are from a containing display or an embedded display.

3.35.5 Validation

Display validation includes the validation of embedded displays. That is, embedded displays are validated when the containing display is validated, even if they were validated before embedding.

3.36 Embedded Display Parameters

3.36.1 Display parameters are user-defined display properties

The user can define the parameters of a display via the *Define Parameters* dialog. Although the word *parameter* implies that these items behave like the parameters of a subroutine, they really behave more like the properties of an object.

Although Display Parameters were developed for use with embedded displays, they may also be used in a top-level display. When used in a top-level display, these parameters are similar to public variables

3.36.2 Overview of parameter usage

Display parameters are used to convey data between the containing display and the embedding display. They can be used to convey static data at display startup (e.g. to use in displaying labels or setting limits), or they can convey dynamic data as the display executes (e.g. to use in setting a needle position in a dial). Execution of scripts and dynamics in an embedded display can be driven by changes in the value of display parameters. Specialized display parameter data-types (Entity and Variable) are provided to support building of embedded displays that manipulate LCN points generically and that are bound to specific LCN points when the display is embedded. Another specialized display parameter type (Inline) uses a build-time text substitution approach.

Some kinds of display parameters (Value kind, described below) can also be used to store data that is accessible throughout the display. They can be used an alternative to public variables and some DDB values, with the following added benefits:

- Visible in all scripts (according to the scope rules described below), without the need to use a dim statement everywhere as is necessary with public variables;
- Visible in the expression field of dynamics;
- More efficient than DDB usage, with immediate firing of the data change event.

3.36.3 Parameter kinds

When defining a parameter (by using the Define Parameters dialog), the user must select a data-type for the parameter. The data-types can be divided into three categories or kinds. The kinds are distinguished by the way in which the parameters are bound to actual values (as described below). The three categories are:

- Value parameters. These are parameters having the conventional Basic language data types. They behave like user defined display properties. Select one of Integer, Long, Single, Double, Date, Currency, String, or Object on from the Type drop down menu on the Add Parameter dialog of the Define Parameters dialog to define a value type parameter.
- Reference parameters. These represent LCN entities or variables. These parameters have special processing that binds them to the entity or variable specified on the Enter Parameters dialog. They behave as aliases for the item to which they are bound. Select either Entity or Variable on the Type drop down menu on the Add Parameter dialog of the Define Parameters dialog to define a reference type parameter.
- Inline parameters. These support a text substitution type of parameter passing mechanism. An occurrence of an Inline parameter in a display is replaced by its actual value when the script is validated. Select Inline on the Type drop down menu on the Add Parameter dialog of the Define Parameters dialog to define an Inline parameter.

3.36.4 How to access value and reference type display parameters from scripts

The reference and value type parameters of an embedded display are accessible to scripts from both inside and outside of the embedded display. These parameters are always accessed as properties of the *params* property of

a display object. The name of a parameter is the name assigned by the user in the *Define Parameters* dialog. When accessed from the containing display, use the *params* property of the embedded display object. When accessed from inside an embedded display, use the *params* property of the built-in *display* object.

For example, if you wish to display a parameter named *dialposition* of an embedded display named *embeddeddisplay1* from the containing display you could use

msgbox embeddeddisplay1.params.dialposition

For example, if you wish to set the angle property of a rectangle from its attached script to the value of an embedded display parameter named *dialposition* from inside the embedded display you could use

me.angle = display.params.dialposition

The only constraints on the assignment of values to display parameters apply to the following parameter types:

- Object
- Variable
- Entity

Because these parameters are implemented as pointers they must be assigned to point to an actual specific Object, Variable or Entity. Specifically, the following constructs will not work:

- display.params.myent = getent('a100')
- display.params.myvar = getvar('a100.pv')
- display.params.myobject = nothing

Additionally, OnDataChange scripts referencing these parameters will fire only when their assignment is changed and NOT when the values they point to change.

3.36.5 Limitations when using value type display parameters

Value type display parameters (such as display parameters whose type corresponds to a basic language data type) do not have status information like that of an LCN parameter. This means that conditions such as Bad Value cannot be directly represented using parameters of this type. Use reference or inline type parameters when the primary purpose of a display parameter is to bind to an LCN parameter.

3.36.6 How to access Inline type display parameters

Inline parameters are accessed simply by using the parameter name in a script, expression field of a dynamic page on a properties dialog, or in the expression of an enter parameters dialog. **Do not** prefix the name of an Inline display parameter with ‘display.params'. References to Inline parameters follow normal delimiter rules for identifiers.

Inline type parameters can only be accessed in the display that defines them. They cannot be accessed in the containing display after the owning display is embedded.

Inline parameters allow you to parameterize aspects of a display that don't work well with the ‘property' model followed by value and reference type display parameters. Inline parameters behave like a C language #define preprocessor statement in that they are substituted texturally at buildtime for their actual value.

Note: If a parameter is Inline because of substitution at validation time, the Inline parameter must be embedded for validation to work.

For example, if you wish to display an Inline parameter named **Temperature** in a msgbox you could use

msgbox Temperature

At validation time, **Temperature** will be substituted for the **text entered by the user** in the Enter Parameters dialog. Note that the reference to Temperature is not prefixed by ‘display.params'.

For example, if you wish to parameterize the parameter used in an LCN reference you could define an inline parameter named ‘param' using the Define Parameters dialog. You could then reference ‘param' in the script

me.text = lcn.a100.param

If the user entered ‘pv' in the enter parameters dialog for the param parameter, this statement would validate as

me.text = lcn.a100.pv

Two special operators are provided to enable usage of Inline parameters inside of string literals. The operators are entered as ‘\p(paramname)' or ‘\pe(paramname)', where paramname is the name of an Inline display parameter. The ‘\p' operator performs a simple substitution at validation time. The ‘\pe' performs a conversion of the actual text during the substitution. It converts a name form from that used in BasicScript to a traditional LCN name form.

For example, during substitution of a \pe reference, a value that appears as ‘LCN.A100.[NAME]' in the enter parameters dialog is converted to ‘A100.NAME' during the substitution of a \pe operator that occurs during validation.

Tip
If a display with Inline parameters uses strings (such as this example pathname:'C:\Picture\a.pct') where this SUBSTITUTION SHOULD NOT OCCUR, use a second backslash (such as this example pathname:'C:\\Picture \a.pct') to prevent the substitution.

The \pe string substitution operator is useful when you wish to parameterize a collector reference. In general, you should design your embedded display such that the user of the embedded display is requested to enter the BasicScript name form for LCN references on the Enter Parameters dialog. When you script the embedded display, use the \pe operator to parameterize collector references.

For example, the following script could be used in an embedded display having an inline parameter named TempValue. This example shows a collector reference and a non-collector reference to the same display parameter. In this example, the user could enter a name such as ‘LCN.A100' in the enter parameters dialog when they embed this display. The embedded display would also work correctly if they entered ‘Dispdb.ent01'.

```
Sub OnDataChange
Me.text = collector('ackstat(\pe(TempValue))')
valuetext.text = TempValue.pv‘ Valuetext is a text object in this display
End Sub
```

Note that Inline parameter substitution does occur in the expression field of the Enter Parameters dialog. This allows you to propagate inline parameter values into nested embedded displays.

Inline parameters that are not given a value by the user in the enter parameters dialog are substituted with a null string during validation.

3.36.7 Overview of value-type parameters

Parameters whose type is one of the Basic language data-types are simply user-defined data items into which values can be stored and from which data can be read. Because they are visible in scripts in both the containing and embedded display, you can see how these kinds of parameters can be used to pass data between the two contexts.

3.36.8 Overview of reference-type parameters

Parameters whose type is either *variable* or *entity* are specialized for access to LCN variables and entities. These parameters are similar to DDB VAR and ENT items because they can contain a reference or alias for an LCN object. The reference is set up by using the *Enter Parameters* dialog, as described in the next section. A usage of these parameters behaves as if you are accessing the entity or variable to which they have been bound. That is, you can reference the parameters of a point indirectly through an entity-type display parameter and you can reference a network variable (such as a point.parameter) indirectly through a variable-type display parameter.

Unlike DDB VAR and ENT items, whose reference can be reset, you cannot change the point to which a reference-type parameters is bound. This reference is established by the *Enter Parameters* dialog and cannot be changed programmatically.

3.36.9 Enter Parameters dialog: Binding of value-type parameters

The *Enter Parameters* dialog allows you to specify a value for a parameter. This section discusses the use of this dialog for value-type parameters (such as parameters whose type is a Basic language type like *long*).

You can enter two expressions for a value-type parameter, one for the initial value (labeled *Initial Value Expression* on the dialog), and one to be executed as the data it references changes (labeled *OnDataChange Expression* on the dialog). It is useful to think of the *Enter Parameters* dialog for value-type parameters as resulting in the generation of assignment statements in a script subroutine. You can think of the initial value expression as the right-hand side of an assignment statement in an OnDisplayStartup subroutine. You can think of the OnDataChange expression as the right-hand side of an assignment statement in an OnDataChange subroutine.

The actual value of the parameter will be the result of evaluating the expression. Therefore, string constants must be enclosed in quotes (scripting will attempt to evaluate a non-quoted string as the name of a variable).

The following are guidelines for when to use the *Initial Value Expression* vs. the *OnDataChange* expression:

- Use the initial value expression when the data assigned to the parameter is static. This may be useful for display parameter values used to display a label in the embedded display. Note that any LCN object references in this expression are immediate-type accesses that are evaluated once at display startup.
- Use the OnDataChange expression when the value of the expression is dynamic based on an LCN object reference in the expression. Note that the OnDataChange expression is only executed when the LCN data it references changes. It follows the rules for execution of an OnDataChange entry point. This means that if you put an expression containing no LCN or DispDb reference, this expression will never be evaluated and the value of the display parameter will not be assigned. There is generally no need for an initial value expression if you use the OnDataChange expression because the initial data scan will result in an initial execution of this expression.

3.36.10 Enter Parameters dialog: Binding of reference-type parameters

The *Enter Parameters* dialog for reference-type parameters lets you bind a display parameter to a particular LCN or DISPDB variable or entity. After binding, the display parameter becomes an alias for the LCN point. Enter an LCN variable or entity in the edit box labeled *LCN Reference*. Enter the reference as you would a reference to an LCN or DISPDB object in a script. For example, to bind an entity-type parameter to the point A100, enter: lcn.a100. To bind a variable-type parameter to A100.PV enter'lcn.a100.pv'.

Attention

Limitation: Binding a variable-type parameter to an expression with dynamic indirection is not supported. If'lcn.custom1.str(dispdb.int01)' was bound to a variable-type parameter, the result would NOT update with changes to the value of dispdb.int01.

3.36.11 Enter Parameters dialog: Binding of inline-type parameters

The Enter Parameters dialog for inline-type parameters lets you specify the text to be substituted during validation for a particular display parameter. The validation process performs a simple text substitution, replacing an occurrence of a parameter in the script text, with the text specified in the enter parameters dialog. Normally, you should not surround the text you enter in the enter parameters dialog with quotes, unless you wish to bind the parameter to a string literal.

3.36.12 OnDataChange reacts to changes in parameter values

During display execution, changes in the value held by value and reference type display parameters result in the execution of OnDataChange entry points in the embedded display. In other words, if you reference a display parameter in an OnDataChange subroutine, the subroutine will be executed whenever the value of the parameter

changes. A reference in an OnDataChange subroutine using any of the possible name forms and scopes for display parameters results in this behavior. For example, for a display having a String type parameter named str1, the following name forms in an OnDataChange subroutine will result in data change events as the value changes:

text1.text = me.params.str1	‘ In the main display script
text1.text = embeddedpicture1.params.str1	‘ In a script in the containing display
text1.text = display.params.str1	‘ In a script in the embedded display itself

For a value-type parameter, this means that dynamic embedded displays can be developed by writing scripts that reference display parameters in OnDataChange subroutines or that simply reference a display parameter in an expression of a dynamic. Scripts in the container (or other scripts in the embedded display) can assign values to a parameter, resulting in execution of respective OnDataChange entry points.

Reference-type display parameters behave like aliases for the LCN objects to which they are bound. A use of one of these parameters in an OnDataChange subroutine in an embedded display as a property of the'display.params' object behaves exactly like a direct reference to the bound LCN object. If the bound object is a point, it is valid to reference any valid parameter of the point. If the LCN object is a variable, it is valid to use any of the properties of a parameter object to access LCN data.

3.37 General Guidelines for Value Type Display Parameter Usage

This section concentrates on the behavior of value type parameters during display execution.

- Because inline parameters are substituted for their actual value at build time, they do not have special runtime behavior and so are not discussed in this section.
- Because reference type parameters are simply an alias for the entity or variable to which they are bound and so have the same behavior as variables and entities, they also are not discussed in this section.

Listed in this section are several guidelines for use of value type display parameters in embedded displays and for users of embedded displays when they are filling in the'Enter Parameters' dialog. Following these guidelines will make it easier to produce embedded displays that give the user of the display the results they expect.

1	The author of a display that is intended to be embedded in another display should always write scripts that reference display parameters under the'OnDataChange' event or mouse events. Scripts under the'OnDisplayStartup' event should never reference display parameters.
2	The user who is embedding a display into another display should select the'on data change (dynamic)' for the update frequency of a parameter if the entered expression for that parameter contains LCN or DDB variables. The'OnDisplayStartup (static)' selection should be made if the expression contains only literal values. This is needed because literal values do not cause OnDataChange events.
3	If the expression entered for a parameter of an embedded display contains LCN and/or DDB variables that are known to be static, such as configuration data, and the user wants to improve display performance by not having them periodically collected, he should set the desired variable collection rates appropriately from the Data Collection dialog. The display parameter update frequency should still be set to'OnDataChange (dynamic).'
4	Usage of'OnDisplayStartup' expression on the'Enter Parameters' dialog will result in execution, at display startup, of'OnDataChange' entry points in the embedded display if those entry points reference the display's parameters. This is convenient because embedded display developers don't have to differentiate between data change events for initialization and data change events during normal display operation. However, if the OnDataChange entry point in the embedded display mixes references to LCN data and display parameters, there will be a transient condition in which the OnDataChange is initially executed with LCN data access errors immediately followed by an OnDataChange event with valid data.

Use an error handler to respond to the initial data access error if you

- Choose to mix reads of display parameters and LCN data in the same OnDataChange entry point, or
- Inform the users of the embedded display not to use the'OnDisplayStartup' expression on the Enter Parameters dialog for the parameters that are mixed with LCN references.

3.37.1 Expression-type on'Enter Parameters'

The following table contains guidelines for how to use combinations of the parameter reference location in embedded displays and parameter bindings in the'Enter Parameters' dialog.

Location of Display Parameter Reference in Embedded Display	On Data Change (Dynamic)	On Display Startup (Static)
on data change	Used when entered parameter value is an expression, containing LCN and/or DDB variables.	Used when the entered parameter value is a literal. The scripts in the embedded display that mix display parameter references with LCN or DDB references in expressions may evaluate before the initial collection of LCN and DDB variables. This results in a transient condition in which the display behavior does not factor in the LCN and DDB variables.

Location of Display Parameter Reference in Embedded Display	On Data Change (Dynamic)	On Display Startup (Static)
on display startup	Never do this!! The value of the parameter will never be passed to the embedded display.	Never do this!! If the embedded display script that references the embedded display's parameters is in the OnDisplayStartup subroutine, the user of the embedded display can only define the parameter to be passed on display startup (static). There is no advantage in allowing the user of an embedded display to define the parameters as static and it causes confusion when the user defines the parameter as dynamic and the embedded picture does not behave as expected.

3.38 Access to Workspace Functions from Scripts

This section presents the end-user view of workspace functionality accessible from the scripting language.

A workspace object is predefined in GUS BasicScript. All of the properties and methods of the workspace are available on this object. Refer to Workspace Organization in the *SafeView User's Guide* for details on how workspaces are managed by SafeView.

3.39 Built-in Functions and Statements

In addition to the standard BasicScript functions, a list of the functions and statements added by Honeywell for GUS BasicScript is as follows:

- CollectHistory (statement)
- GetEnt (function)
- GetVar (function)
- GetGUSMode (function)
- InvokeDisplay (statement)
- MakeColor (function)
- IsDouble (function)
- IsSingle (function)
- IsInteger (function)
- IsLong (function)
- IsNan (function)
- RaiseCustomEvent

3.40 CollectHistory (statement)

Related topics

“CollectHistory Syntax” on page 91
“CollectHistory Description” on page 91
“CollectHistory Comments” on page 91

3.40.1 CollectHistory Syntax

CollectHistory InitFlag[,ReferenceTime]

3.40.2 CollectHistory Description

This statement initiates the processing of history collectors.

3.40.3 CollectHistory Comments

The CollectHistory statement requires the following parameters:

Parameter	Description
InitFlag	A Boolean flag that determines whether old history values are discarded before collecting new values. If this parameter is TRUE, the value of all history collectors will be invalid until the new collection is complete. Furthermore, a data change event occurs for all history collectors referenced in OnDataChange subroutines indicating that the history collector data is not valid. If this parameter Is FALSE, the value of all history collectors is not changed, and no data change event occurs until valid, new values are collected.
ReferenceTime	An optional second parameter that specifies a reference Date/Time. This type of parameter is Date. If this parameter is not present, the current time and date will be used. ReferenceTime establishes a date/time for the history collector. Arguments to the collector can offset the collection period from the reference time.

The CollectHistory statement is typically done in a script handling an operator event, after the operator has entered all the entities and parameters needed by the history collectors.

The following examples show several uses of the CollectHistory statement:

```
rem Use GetVar to establish a reference in a dispdb VAR item
CollectHistory TRUE
CollectHistory TRUE, DispDB.DATIME1
```

3.41 GetEnt (function)

Related topics

3.41.1 GetEnt Syntax

GetEnt (expression$)

3.41.2 GetEnt Description

Returns an object that references the LCN entity specified in the expression.

3.41.3 GetEnt Comments

The GetEnt function takes one parameter. The parameter must evaluate to a string containing an LCN entity name form. An LCN entity name has the form *'ponit'*.

The object returned by the function behaves the same as a DispDB ENT item. An object created by GetEnt can be used to establish a reference in a DispDB ENT item. That is, these objects can appear on the right-hand side of the equal sign in a SET statement, where the item on the left-hand side of the equal sign is a DispDB ENT item.

Note that when a GetEnt generated object is used in an OnDataChange entry point, it does *not* set up OnDataChange processing for the referenced LCN entity.

3.41.4 GetEnt Example

```
rem Use GetEnt to establish a reference in a dispdb ENT item
Set dispdb.ent01 = GetEnt('A100')
```

3.42 GetVar (function)

Related topics

3.42.1 GetVar Syntax

GetVar (expression$)

3.42.2 GetVar Description

Returns an object that references the LCN variable specified in the expression.

3.42.3 GetVar Comments

The GetVar function takes one parameter. The parameter must evaluate to a string containing an LCN variable name form. An LCN variable name has the form *'point.parameter'*, where point is the name of an LCN point and parameter is the name of a parameter on the point.

The object returned by the function behaves similarly to a DispDB VAR item. The *value* property of the object accesses the referenced LCN variable. When the value property of the object is read, it returns the value of the referenced variable, and when it is assigned a value it writes to the variable. The *value* property is the default property of the object. The reads and writes performed by this object are immediate type accesses.

An object created by GetVar can be used to establish a reference in a DispDB VAR item. That is, these objects can appear on the right-hand side of the equal sign in a SET statement in which the item on the left-hand side of the equal sign is a dispDB VAR item. The simplest way to establish a reference in a DispDB VAR item is to use the GetVar function as the assigned value in a SET statement.

Note that when a GetVar generated object is used in an OnDataChange entry point, it does not set up OnDataChange processing for the referenced LCN variable.

3.42.4 GetVar Example

These examples show several uses of the GetVar function. The first shows GetVar being used to establish a reference to an LCN variable in a DispDB VAR item. The second shows GetVar being used to create a variable reference object.

rem Use GetVar to establish a reference in a dispdb VAR item

```
Set dispdb.var01 = GetVar('A100.PV')
```

rem Use GetVar to create an object that behaves like a VAR item

```
dim obj as object
set obj = GetVar('A100.PV')
obj = 100
msgbox'The value of A100.PV is ' & obj.value
```

3.43 GetGUSMode (function)

Related topics

3.43.1 GetGUSMode Syntax

GetGUSMode

3.43.2 GetGUSMode Description

The GetGUSMode scripting function provides runtime information about the current GUS LCN connection type.

3.43.3 GetGUSMode Comments

There are *no* input parameters for this function. The return value is one of the following three enumerated types:

- GUSLocal
- GUSRemote
- GUSDesktop

GUSLocal signifies that the GUS runtime accesses LCN data through an LCNP card on the same computer.

GUSRemote signifies that the GUS runtime accesses LCN data through a remote machine's LCNP card.

3.43.4 GetGUSMode Example

```
Sub OnLButtonClick()
If GetGUSMode = GUSRemote Then
MsgBox'You are running a remote GUS'
End If
End Sub
```

3.44 InvokeDisplay (statement)

Related topics

3.44.1 InvokeDisplay Syntax

InvokeDisplay displaypath$ [,Window$][,Disp_parameter]

3.44.2 InvokeDisplay Description

Invokes the specified GUS display.

3.44.3 InvokeDisplay Comments

The InvokeDisplay statement takes the following parameters:

Parameter	Description
displaypath$	A string containing the file path for a GUS display file. This can be a partial path, in which case the display search path will be used to locate the display. NOTE: The maximum length of the displaypath$ parameter is 260 characters.
Window$	An optional second parameter that specifies a window specification name within a SafeView configuration to be used to manage the resulting display window. SafeView must be running and loaded with a configuration having the window specification for this parameter to be effective. Normally, this parameter is not used. It is recommended that you allow SafeView to select the window specification based on the match criteria of the current configuration and the current state of the workspace. Use of this argument can be confusing to the operator because it overrides the output focus concept of the workspace. This parameter, however, can be useful when populating a workspace with displays in consistent locations.
Disp_parameter	An optional third parameter of type variant that can be used to pass a data value to the invoked display. The invoked display can access this value via the GetDisplayParameter function.

The InvokeDisplay statement causes the specified GUS display file to be displayed in run mode. The display will appear in a new window. InvokeDisplay uses the display manager. Regardless of whether or not the display manager is active, the specified display will be invoked.

3.44.4 InvokeDisplay Example

The following examples show uses of the InvokeDisplay statement to cause displays to be shown.

InvokeDisplay'Overview1.pct'

InvokeDisplay'd:\mydisplays\Overview2.pct','','Area02'

InvokeDisplay'Summary.pct','SummaryWindow'

dim *disp* as string

disp = AskBox ('Enter desired display')

InvokeDisplay disp

3.45 GetDisplayParameter (function)

Related topics

3.45.1 GetDisplayParameter Syntax

GetDisplayParameter

3.45.2 GetDisplayParameter Description

Returns the {variant} value of the Disp_parameter that was included in the InvokeDisplay statement that invoked this display.

3.45.3 GetDisplayParameter Comments

If the display is started by Runpic, the SCHEM actor or an InvokeDisplay statement that did not include a Disp_parameter, then GetDisplayParameter will return a NULL string.

3.45.4 GetDisplayParameter Example

This example sets the tagname in a Faceplate by using GetDisplayParameter.

```
Sub OnDisplayStartup()
If GetDisplayParameter <>'' then ‘NULL string
fp.tagname = GetDisplayParameter
end if
End Sub
```

3.46 MakeColor (function)

3.46.1 MakeColor Syntax

MakeColor (red, green, blue)

3.46.2 MakeColor Description

Constructs a color value.

3.46.3 MakeColor Comments

The MakeColor function takes the following parameters:

Parameter	Description
red	A short integer containing the red value of the color. This value must be between zero and 255.
green	A short integer containing the green value of the color. This value must be between zero and 255.
blue	A short integer containing the blue value of the color. This value must be between zero and 255.

The MakeColor function constructs a PALETTERGB-type color value for use in assignment to various display object properties requiring a color value. The desired RGB values can be determined by using the color selection dialog displayed when setting various color properties of display objects from the properties dialog. The color dialog displays the current Red-Green-Blue intensity values for the select color.

3.46.4 MakeColor Example

This example sets the value of a fillcolor property by using MakeColor.

rectangle1.fillcolor = MakeColor(0, 122, 255)

3.47 IsDouble (function)

Related topics

3.47.1 IsDouble Syntax

IsDouble (expression$)

3.47.2 IsDouble Description

Returns True if the string expression can be converted to a number of type double or if the string contains the TPS Nan indicator, returns False otherwise.

3.47.3 IsDouble Comments

The following syntax is recognized as a valid number:

[whitespace] [sign] [digits] [.digits] [{e | E}[sign]digits]

A TPS Nan indicator consists of a sequence of three or more consecutive hyphens (e.g.'---').

3.47.4 IsDouble Example

```
Sub Main()
Dim s$ As String
s$ = me.text
If IsDouble(s$) Then
MsgBox'You did good!'
Else
MsgBox'You didn't do so good!'
End If
End Sub
```

3.48 IsSingle (function)

Related topics

3.48.1 IsSingle Syntax

IsSingle (expression$)

3.48.2 IsSingle Description

Returns True if the string expression can be converted to a number of type single or if the string contains the TPS Nan indicator, returns False otherwise.

3.48.3 IsSingle Comments

The following syntax is recognized as a valid number:

[whitespace] [sign] [digits] [.digits] [{e | E}[sign]digits]

A TPS Nan indicator consists of a sequence of three or more consecutive hyphens (e.g.'---').

3.48.4 IsSingle Example

```
Sub Main()
Dim s$ As String
s$ = me.text
If IsSingle(s$) Then
MsgBox'You did good!'
Else
MsgBox'You didn't do so good!'
End If
End Sub
```

3.49 IsInteger (function)

Related topics

3.49.1 IsInteger Syntax

IsInteger (expression$)

3.49.2 IsInteger Description

Returns True if the string expression can be converted to a number of type integer, returns False otherwise.

3.49.3 IsInteger Comments

The following syntax is recognized as a valid number:

[sign]digits

3.49.4 IsInteger Example

```
Sub Main()
Dim s$ As String
s$ = me.text
If IsInteger(s$) Then
MsgBox'You did good!'
Else
MsgBox'You didn't do so good!'
End If
End Sub
```

3.50 IsLong (function)

Related topics

"IsLong Syntax" on page 102
"IsLong Description" on page 102
"IsLong Comments" on page 102
"IsLong Example" on page 102

3.50.1 IsLong Syntax

IsLong (expression$)

3.50.2 IsLong Description

Returns True if the string expression can be converted to a number of type long, returns False otherwise.

3.50.3 IsLong Comments

The following syntax is recognized as a valid number:

[sign]digits

3.50.4 IsLong Example

```
Sub Main()
Dim s$ As String
s$ = me.text
If IsLong(s$) Then
MsgBox'You did good!'
Else
MsgBox'You didn't do so good!'
End If
End Sub
```

3.51 IsNan (function)

Related topics

3.51.1 IsNan Syntax

IsNan (expression$)

3.51.2 IsNan Description

Returns True if the string expression contains the TPS Nan indicator, returns False otherwise.

3.51.3 IsNan Comments

A TPS Nan indicator consists of a sequence of three or more consecutive hyphens (e.g.'---').

3.51.4 IsNan Example

```
Sub Main()
Dim s$ As String
s$ = me.text
If IsNan(s$) Then
MsgBox'You did good!'
Else
MsgBox'You didn't do so good!'
End If
End Sub
```

3.52 RaiseCustomEvent

Related topics

“RaiseCustomEvent Syntax” on page 104
“RaiseCustomEvent Description” on page 104
“RaiseCustomEvent Example” on page 104

3.52.1 RaiseCustomEvent Syntax

RaiseCustomEvent ID, Parameter

where;

ID = string uniquely identifying the custom event. Event ids prefixed with ‘_' are reserved for Honeywell

Parameter = [opt] Variant as defined by the user

3.52.2 RaiseCustomEvent Description

Used to fire the GUS built in event, OnCustomEvent.

3.52.3 RaiseCustomEvent Example

```
Sub OnLButtonDown
RaiseCustomEvent'SelectedTag','A100-Detail'
End Sub
```

3.53 Built-in Constants

This section lists the built-in constants added by Honeywell to the GUS Basic language. Built-in constants are identifiers that represent constant values. They are provided as a convenience to be used when assigning values to certain display object properties.

Related topics

"Color Values" on page 105
"Fill Patterns" on page 105
"Line Styles" on page 106
"Fill directions" on page 106
"Text Alignment" on page 107
"Common Status Errors" on page 107
"Example Usage" on page 107

3.53.1 Color Values

The following constants are provided for setting various color properties:

Name	Color	Intensity	Red	Green	Blue
TDC_BLACK	Black	None	0	0	0
TDC_RED	Red	Full	255	0	0
TDC_GREEN	Green	Full	0	255	0
TDC_YELLOW	Yellow	Full	255	255	0
TDC_BLUE	Blue	Full	0	0	255
TDC_MAGENTA	Magenta	Full	255	0	255
TDC_CYAN	Cyan	Full	0	255	255
TDC_WHITE	White	Full	255	255	255
TDC_HALF_RED	Red	Half	128	0	0
TDC_HALF_GREEN	Green	Half	0	128	0
TDC_HALF_YELLOW	Yellow	Half	128	128	0
TDC_HALF_BLUE	Blue	Half	0	17	170
TDC_HALF_MAGENTA	Magenta	Half	128	0	128
TDC_HALF_CYAN	Cyan	Half	0	128	128
TDC_HALF_WHITE	White	Half	128	128	128

3.53.2 Fill Patterns

Values used to set the FillPattern property. All are of type integer:

Name	Pattern	Integer Value	Fill property must be set equal to. . . (see Note below)
HS_HORIZONTAL	Horizontal line	0	TRUE
HS_VERTICAL	Vertical line	1	TRUE
HS_FDIAGONAL	Forward diagonal line	2	TRUE

Name	Pattern	Integer Value	Fill property must be set equal to. . . (see Note below)
HS_BDIAGONAL	Backward diagonal line	3	TRUE
HS_CROSS	Cross	4	TRUE
HS_DIAGCROSS	Diagonal cross	5	TRUE
HS_SOLID	Solid Fill	15	TRUE
HS_SOLID	Transparent Fill	15	FALSE

Note: The'.fill' property must be true to be able to see any of the fill patterns. If the'.fill' property is false, the object will have a transparent fill regardless of the value assigned to'.fillpattern'.

In other words, the'.fill' property overrides any fill pattern selected.

Therefore, the .fill property needs to be set equal to true for ALL fill patterns except Transparent fill.

Examples:

1. To set the fill pattern to a cross pattern:

   me.fillpattern=HS_CROSS

   me.fill=true
2. To set the fill pattern to a cross pattern:

   me.fillpattern=4

   me.fill=true
3. To set the fill pattern to a solid pattern:

   me.fillpattern=HS_SOLID

   me.fill=true
4. To set the fill pattern to a transparent pattern (regardless of fill pattern selected):

   me.fillpattern=HS_SOLID

   me.fill=false
5. To set the fill pattern to a transparent pattern (regardless of fill pattern selected):

   me.fillpattern=HS_HORIZONTAL

   me.fill=false

3.53.3 Line Styles

Values used to set the LineStyle property:

Name	Purpose
PS_SOLID	Solid Line
PS_DASH	Dashed Line
PS_DOT	Dotted Line
PS_DASHDOT	Dash-Dot-Dash-Dot
PS_DASHDOTDOT	Dash-Dot-Dot-Dash-Dot-Dot

3.53.4 Fill directions

Values used to set the FillDirection property:

Name	Purpose
FILL_LR	Fill from Left to Right
FILL_RL	Fill from Right to Left
FILL_BT	Fill from Bottom to Top
FILL_TB	Fill from Top to Bottom

3.53.5 Text Alignment

Values used to set the TextAlignment property:

Name	Purpose
DT_LEFT	Align Text to Left Edge
DT_RIGHT	Align Text to Right Edge
DT_CENTER	Align Text to Center

3.53.6 Common Status Errors

Values used in comparisons with the status property of parameters.

Name	Purpose
HOPC_NO_ERROR	No errors
HOPC_COMMUNICATION_ERROR	Communication on the LCN is disrupted or a device has failed.
HOPC_CONFIGURATION_ERROR	Can usually be corrected with the LCN system. Check for database mismatch errors.
HOPC_VALUE_ERROR	Value error—bad data or value out of range.
HOPC_STORE_ERROR	Internal error—database or software problem.
HOPC_DYNAMICINDEXERROR	The data is not available because the array index has not been calculated.
HOPC_NOT_YET_SUPPORTED	Internal error—the data access request is not valid.
HOPC_NO_ERROR_BUT_NO_DATA	During Display Validation, a point.parameter using indirection (for example, A100.P1.PV) cannot be verified accurately.
0x8002802B	Element Not Found or Full tag name (point parameter) entered as an Entity name
0x8002000A	Out of Present Range or Floating point value assigned outside the range of a Single to an LCN real (like 2E38)
1054	Store Error

3.53.7 Example Usage

The following shows an example of a fill pattern constant

rectangle1.fillpattern = HS_CROSS

The following shows an example of a color value constant

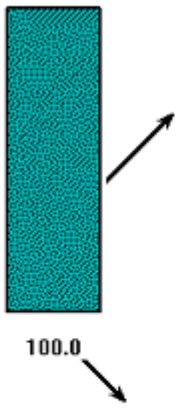
```
rectangle1.fillcolor = TDC_GREEN
```

3.54 Example Display with Scripts

The following is a simple example containing two display objects, a Rectangle-Bar object and a Text object. The Rectangle-Bar object behaves like a combination of a bar chart and a target. When the dynamics of the point'a100.pv' are set up in the Rectangle-Bar's property page, its height changes as the value of the point changes.

These script examples show functionality you might add.

- The Rectangle-Bar object's script changes its color from green to red when the point's high-temperature (PVHITP) is exceeded.
- The Text object's script changes its numeric value as the value of'a100.pv' changes.




```
Sub OnDataChange
    If lcn.a100.PV> lcn.a100.PVHITP Then
        me.fillcolor = TDC_RED
    Else
        me.fillcolor = TDC_GREEN
    End if
End sub
```

```
Sub OnDataChange
        me.text = lcn.a100.PV
End sub
```

3.55 Trend Property Constants

The following built-in constants are provided to be used with the Trend OLE control OnPropertyChange() event. Each constant corresponds to a property, which will equal the value of the nProperty argument returned in the OnPropertyChange() event.

Trend or Trace Property	Built-in Constant
AxesColor	TREND_AXESCOLOR
BackgroundColor	TREND_BACKGROUNDCOLOR
Correlation Coefficient	TREND_CORRELATION
DataSource	TREND_DATASOURCE
Description	TREND_DESCRIPTION
DisplayRelative	TREND_DISPLAYRELATIVE
GridColor	TREND_GRIDCOLOR
HairlineColor	TREND_HAIRLINECOLOR
HairlineCursor	TREND_HAIRLINECURSOR
HairlineCursorActive	TREND_HAIRLINECURSORACTIVE
HairlineReadout	TREND_HAIRLINEREADOUT
InheritProperties	TREND_INHERITPROPERTIES
NumTraces	TREND_NUMTRACES
Paused	TREND_PAUSED
RuntimeScrolling	TREND_RUNTIMESCROLLING
ScrollBackTime	TREND_SCROLLBACKTIME
ScrollForwardTime	TREND_SCROLLFORWARDTIME
ShowAxes	TREND_SHOWAXES
ShowGrid	TREND_SHOWGRID
TimeBase	TREND_TIMEBASE
TimeBaseIndex	TREND_TIMEBASEINDEX
TimeLimitLeft	TREND_TIMELIMITLEFT
TimeLimitRight	TREND_TIMELIMITRIGHT
TraceColor	TREND_TRACECOLOR
VariableID	TREND_VARIABLEID
XCursorTime	TREND_XCURSORTIME
XCursorTimeDate	TREND_XCURSORTIMEDATE
YCursorReadout	TREND_YCURSORREADOUT
YCursorReadoutReal	TREND_YCURSORREADOUTREAL
YdisplayTicPercents	TREND_YDISPLAYTICPERCENT
YNumMinorGridLines	TREND_YNUMMINORGRIDLINES
YnumTicMarks	TREND_YNUMTICKMARKS
YRangeHigh	TREND_YRANGEHIGH
YRangeLow	TREND_YRANGELOW
YScaleHigh	TREND_YSCALEHIGH

Trend or Trace Property	Built-in Constant
YScaleLow	TREND_YSCALELOW

3.56 Trend Status Constants

The following built-in constants are equivalent to the valid values of the Trend OLE control property YCursorReadoutStatus.

Name	Purpose
TREND_STATUS_OK	A good value has been received from the LCN for this timestamp.
TREND_STATUS_ERR	No data has been received from the LCN for this timestamp.
TREND_STATUS_BAD	A bad status has been received from the LCN for this timestamp.
TREND_STATUS_NA	The trace is not active.

4 Using OPC Server Automation Interfaces

Related topics

4.1 Overview of Automation Interfaces

- Section Overview
- Not all OPC server methods are directly accessible from GUS displays
- GUS scripting does not support the'Variant containing an array' data type
- Using the accessor functions provided is recommended
- OPC methods not directly accessible from GUS displays
- Built-in functions are provided to overcome data type limitations
- GUS displays and secondary interfaces
- Using OPC asynchronous I/O
- Coding strategies when using automation methods of OPC objects

4.1.1 Section Overview

This section describes how to use the Automation interfaces of OPC servers from a GUS display. For information on OPC interfaces, refer to the HCI/OPC Data Access User's Guide. This section describes extensions required in BasicScript to fully support the OPC methods and lists some suggested coding practices.

4.1.2 Not all OPC server methods are directly accessible from GUS displays

In general, OPC automation methods are accessed using BasicScript's built-in support for OLE automation. Unfortunately, BasicScript does not support all the possible data types that can be passed in an automation method. The result is that not all automation methods are directly accessible from GUS Basic. A set of built-in functions is provided to overcome this limitation. These functions are described in more detail below.

The standard functionality of Objects as defied for BasicScript applies.

GUS scripting does not support the Variant containing an array data type

The data type used in many of the OPC methods that BasicScript does not support is'Variant containing an array.' Be careful not to confuse this with an array of type'Variant,' which is fully supported by BasicScript. The following BasicScript code illustrates the difference between a variant containing an array and an array of variants.

```
Dim v as Variant
‘ Declare a variant
Dim varr(5) as Variant
‘ Declare an array of variant. This is fully supported by GUS Basic
Dim intarr(5) as integer
‘ Declare an array of integer
v = intarr
‘ Assign the whole array to the variant. The variant is now a variant
‘ containing an array. This is NOT supported in BASICSCRIPT.
‘ You will get the error'whole array assignment not supported.'
```

4.2 Using the accessor functions provided is recommended

You may find that all OPC methods can be directly accessed on some servers by using an array of a base type in place of a variant containing an array. You may also find that this works for in-process servers, but not when the same server is run locally or remotely. This is due to variations in the implementation of servers with respect to their ability to convert actual parameter data types to those needed by the server. Because of this variability, it is recommended that you use the accessor functions described below to provide portability across different servers and different server locality.

Typically, the error caused by the lack of support for variants containing arrays is either the OLE error'Invalid Callee' or the BasicScript error'Type Mismatch.'

4.3 OPC methods not directly accessible from GUS displays

The following is a list the OPC automation methods that are not directly accessible from GUS displays. (These methods can be indirectly accessed via the accessor functions described below). The notation lists the method name preceded by the interface name:

IOPCItemMgtDisp::AddItems
IOPCItemMgtDisp::RemoveItems
IOPCItemMgtDisp::ValidateItems
IOPCItemMgtDisp::SetActiveState
IOPCItemMgtDisp::SetClientHandles
IOPCItemMgtDisp::SetDataTypes
IOPCSyncIODisp::OPCRead
IOPCSyncIODisp::OPCWrite
IOPCASyncIODisp::OPCRead
IOPCASyncIODisp::OPCWrite

4.4 Built-in functions are provided to overcome data type limitations

GUS displays provide a set of built in functions that allow you to access those OPC methods that would otherwise be inaccessible due to lack of support for variants containing arrays in BasicScript. In all cases the arguments of the accessor functions are the same as that of the corresponding OPC method except as follows:

- There is an additional argument of type object added to the front of the argument list that is used to pass the OPC group object into the function.
- Where the OPC method calls for an argument of type'Variant containing an array,' use an array directly. This typically occurs in the'out' parameters where the method is returning an array of items.

The following table lists OPC methods and the corresponding accessor function that must be used to access this method from GUS scripting:

For this OPC Method…	Use this accessor function
IOPCItemMgtDisp::AddItems	OPCAddItems
IOPCItemMgtDisp::RemoveItems	OPCRemoveItems
IOPCItemMgtDisp::ValidateItems	OPCValidateItems
IOPCItemMgtDisp::SetActiveState	OPCSetActiveState
IOPCItemMgtDisp::SetClientHandles	OPCSetClientHandles
IOPCItemMgtDisp::SetDataTypes	OPCSetDataTypes
IOPCSyncIODisp::OPCRead	OPCSyncRead
IOPCSyncIODisp::OPCWrite	OPCSyncWrite
IOPCASyncIODisp::OPCRead	OPCAsyncRead
IOPCASyncIODisp::OPCWrite	OPCAsyncWrite

The following is a list of the accessor subroutines including their argument lists, and with notation showing the optional arguments:

OPCAddItems Group, NumItems, ItemIds, ActiveStates, ClientHandles [, ServerHandles, Errors, ItemObjects, AccessPaths, RequestedDataTypes, Blobs, CanonicalDataTypes, AccessRights]

OPCSyncRead Group, Source, NumItems, ServerHandles, Values [, Qualities, TimeStamps, Errors]

OPCSyncWrite Group, NumItems, ServerHandles, Values [, Errors]

OPCRemoveItems Group, NumItems, ServerHandles, Errors, Force

OPCValidateItems Group, NumItems, ItemIds, Errors [, AccessPaths, RequestedDataTypes, BlobUpdates, Blobs, CanonicalDataTypes, AccessRights]

OPCSetActiveState Group, NumItems, ServerHandles, ActiveState, Errors

OPCSetClientHandles Group, NumItems, ServerHandles, ClientHandles, Errors

OPCSetDataTypes Group, NumItems, ServerHandles, RequestedDataTypes, Errors

The following is a list of the accessor functions including the argument list, and with a notation that shows optional arguments.

OPCASyncWrite Group, NumItems. ServerHandles, Values [, Errors] returns TransactionID

OPCASyncRead Group, Source, NumItems, ServerHandles [, Errors] returns TransactionID

4.5 GUS Displays and Secondary Interfaces

Some objects expose more than one interface; for example, the OPC group object exposes multiple interfaces such as IOPCItemMtgDisp, IOPCSyncIODisp, etc. In a GUS display, all objects are stored in variables of type object, which refers to the object's IDispatch interface. The IDispatch resolves a method invocation to its correct interface base on the name and the actual arguments. However, the OPC specification defines some methods with the same name and possibly the same number of arguments (due to optional arguments). Make the following selections from the Configuration Utility menu bar. One way this is resolved in VB5 is by defining the objects to point to the specific interface. You can do this by referring to the object's type library and then declaring the object as the desired interface. Refer to the HCI/OPC Data Access User's Guide for information. The problem of resolving the secondary interfaces in GUS display is solved by the above extensions because each accessor function has a unique name.

4.6 Using OPC asynchronous I/O

The OPC specification supports asynchronous operations. This is accomplished when the client provides a callback function that is called by the OPC server when the asynchronous operation has completed. Any object in a GUS display can be used as an OPC callback object by defining the desired callback method (OnDataChange or OnAsyncWriteComplete) in the script of the object, and by passing an interface to that object to the OPC group via the AddCallBackReference method.

The OnDataChange subroutine in the object's script must appear as follows:

```
Sub OnDataChange(GroupHandle as Long, Count as Long, ClientHandles() as Long, _
Values() as Variant, Qualities() as Long, GroupTime as Date, _
TimeStamps() as Date)
‘ubroutine body that does something with the data…
End Sub
```

The OnAsyncWriteComplete subroutine in the object's script must appear as follows:

```
Sub OnAsyncWriteComplete(GroupHandle as Long, Count as Long, _
CallbackHandles() as Long, ItemResults() as Long)
End Sub
```

To register the callback, you must call the AddCallbackReference method on the desired group, passing it the value of the object's methods property. For example:

group.AddCallbackReference 1, text1.methods

where text1 is the name of an object in the display having callback methods with the correct argument lists, as shown above. Refer to Section “Primitives - Properties, Methods, and Events” on page 133,' “Primitives - Properties, Methods, and Events” on page 133' and Section “OLE Controls - Properties, Methods, and Events” on page 151,' “OLE Controls - Properties, Methods, and Events” on page 151' for details on the use of *methods* in scripting GUS display objects.

You can define methods on any display object by implementing the methods as a subroutine in the object's script. These user-defined methods can then be invoked using an object's'methods' property. For example, to define a method called MyMethod on object named text1, add a subroutine called MyMethod to the script attached to text1.

```
Sub MyMethod(arg1 as string, arg2 as string)
msgbox arg1 & arg2
End Sub
```

To call the method MyMethod on text1 do the following:

text1.methods.MyMethod'This is my ','method'

4.7 Coding Strategies When Using Automation Methods of OPC Objects

The following are several coding strategies that you can employ to make using the OPC automation interfaces simpler in a GUS display.

- Use public variables for data items that are shared across objects. Typical data items that are shared are group objects and the array of item handles. An alternative is to use display parameters instead of public variables for these items. The benefit of using display parameters is that you need declare the data item only once (using the'Define Parameters' dialog). Note that in this case the term'display parameter' is misleading because you are not using these items as a means of passing data across displays. They are simply being used as a resource used within a display. You can set the'Appears on enter parameters' option on the'Define Parameters' dialog to false if you chose to use display parameters in this way.
- You can encapsulate the OPC automation calls within one object (such as the main display object) and use display parameters to distribute data across the display. In particular, you can take advantage of the ability of display parameters to drive OnDataChange events by assigning values to display parameters in the callback method registered with the OPC server. As data is assigned into the display parameters, the references to those display parameters will be updated through the OnDataChange event mechanism. You can also have data items represented by a pair of display parameters, one representing the data and the other representing the status. Again, this is a use of display parameters as a global resource that drives data change events, rather than as a parameter passing mechanism across displays. You can set the'Appears on enter parameters' option on the'Define Parameters' dialog to false if you chose to use display parameters in this way. Refer to the RAC directory for an OPC example.
- Use display parameters to pass key OPC objects into embedded displays. This would be a useful way to minimize the number of connection made to a single server. Make the connection once in the main display, and then pass the server or group object to embedded displays using display parameters. Refer to the RAC directory for an OPC example.

5 GUS Scripting Application Specifics

Related topics

5.1 Display Object Properties

The following table is a complete list of display object properties and short descriptions of each.

Properties	Read/ Write	Type	Description
Angle	RW (see note)	Double	Sets/Reads clockwise rotation angle in degrees. Note: For the main Display Object, the angle property is *read-only*.
ArcShape	RW	Long	Sets/Reads Arc Shape: <1 = (not defined) 1 = Open Arc 2 = Closed Arc 3 = Pie >3 = (not defined)
AutoParamPrompts	R	Boolean	True if object's Auto Param Prompt state is on, false if not.
BackgroundColor (Display Only)	RW	Long	Sets/Reads the Background Color expressed in its 8-digit RGB value. Note: Do not use with OnDisplayStartup script.
Blink	RW	Boolean	True if object's Blinking state is on, false if not. Can be set with script.
Category (Display Only)	RW	String	Sets/Reads the displays Category string.
CharSet	RW	Integer	Can be set at runtime by scripting assignment. Note: At buildtime, CharSet defaults to 0 (ANSI_CHARSET) and automatically changes to 2 (SYMBOL_CHARSET) when the user selects one of the'Wingding' fonts. It changes back to 0 when any other font is selected.
Closed	RW	Boolean	True if object's Closed state is on, false if not. Can be set with script.
Description	RW	String	Sets/Reads the object's Description string.
DisplayHeight (Display Only)	R	Long	Reads the Display Height Size Property in Pixels.
DisplayWidth (Display Only)	R	Long	Reads the Display Width Size Property in Pixels.
Fill	RW	Boolean	True if object's Filled state is on, false if not filled. Can be set with script.
FillColor	RW	Long	Sets/Reads the Fill Color expressed in its 8-digit RGB value.
FillDirection	RW	Long	Sets/Reads the Direction that object fills: <1 = (not defined) 1 = Fill from Left to Right 2 = Fill from Right to Left 3 = Fill from Bottom to Top (Default) 4 = Fill from Top to Bottom >4 = (not defined)
FillFollowsRotation	RW	Boolean	True if object's Fill Follows Rotation is on, false if not. Can be set with script.

Properties	Read/ Write	Type	Description
FillPattern	RW	Long	Sets/Reads the Fill Lines on an object: <0 = (not defined) 0 = Horizontal Line Fill 1 = Vertical Line Fill 2 = Top Left to Bottom Right Line Fill 3 = Top Right to Bottom Left Line Fill 4 = Horizontal and Vertical Line Fill 5 = Crosshatch Line Fill 6-14 = (not defined) 15 = Solid Fill >15 = (not defined)
FillPercent	RW (see note)	Integer	Sets/Reads the amount the object is filled. Expressed in percent. Note: For the main Display Object, the FillPercent property is *read-only*.
Font	RW	String	Sets/Reads the text object's Font Type string.
Fontsize	RW	Integer	Sets/Reads the text object's Font Size internal value. Note: This will normally be a negative number. Use the Pointsize property to operate on the Text Size in the units shown on the property page.
GridColor (Display Only)	RW	Long	Sets/Reads the Grid Color expressed in its 8-digit RGB value. NOTE: Grid is not displayed in the runtime display.
GridHeight (Display Only)	RW	Long	Sets/Reads the Grid Height expressed in its 8-digit RGB value. NOTE: Grid is not displayed in the runtime display.
GridWidth (Display Only)	RW	Long	Sets/Reads the Grid Width expressed in its 8-digit RGB value. NOTE: Grid is not displayed in the runtime display.
HalfIntensity	RW	Boolean	When True, the text color is altered to appear as half-intensity.
Inherit	RW	Boolean	True if object's Inherit state in on, false if not. Can be set with script.
LineColor	RW	Long	Sets/Reads the object's Line Color expressed in its 8-digit RGB value.
LineStyle	RW	Long	Sets/Reads the Line Style: <0 = (not defined) 0 = Solid line 1 = Dashed line 2 = Dotted line 3 = Dash-Dot-Dash-Dot line 4 = Dash-Dot-Dot-Dash-Dot-Dot line >4 = (not defined)
LineWidth	RW	Long	Sets/Reads the object's Line Width Size in Pixels.
MultiLine	R	Boolean	True if object's Multiline state is on, False if not.
Name	R	String	Reads the object's Name string.
Pointsize	RW	Integer	Sets/Reads the text objects Point size property. Changes to this property are also reflected in the value of Fontsize.
ReferenceName	R	String	Reads the object's Name procedure string.

Properties	Read/ Write	Type	Description
ReferencePath	R	String	Reads the object's Path string.
Reverse	RW	Boolean	When True, the text object is drawn with the text in the configured FillColor and the filled background in the configured TextColor
ScaleX	RW (see note)	Double	Sets the object's Width in the horizontal direction. Enlargement or reduction is about the center of the object. Scale is 1=100%, so 1.5 changes width to 150%. Note: For the main Display Object, the ScaleX property is *read only*.
ScaleY	RW (see note)	Double	Sets the object's Height in the vertical direction. Enlargement or reduction is about the center of the object. Scale is 1=100%, so 1.5 changes width to 150%. Note: For the main Display Object, the ScaleY property is read only.
Selectable	R	Boolean	True if object's Selectable state is on, False if not.
ShowGrid	R	Boolean	True if object's Show Grid state is on, False if hidden.
SnapToGrid	R	Boolean	True if object's Snap To Grid state is on, False of not.
Text	RW	String	Sets/Reads the object's Text string.
TextAlignment	RW	Long	Sets/Reads the Text alignment: <0 = (not defined) 0 = Left Alignment 1 = Center Alignment 2 = Right Alignment >2 = (not defined)
TextBold	RW	Boolean	True if object's Bold state is on, False if not.
TextColor	RW	Long	Sets/Reads the object's Text Color expressed in its 8-digit RGB value.
TextHeight	RW	Long	Sets/Reads the object's Text Height expressed in pixels (or points).
TextItalic	RW	Boolean	True if object's Italic state is on, False if not.
TextStrikeOut	RW	Boolean	True if object's StrikeOut state is on, False if not.
TextUnderline	RW	Boolean	True if object's Underline state is on, False if not.
TransX	RW (see note)	Double	Moves the object in the horizontal direction. Moves right for positive numbers and left for negative numbers. Displacement is in pixels. NOTE: This command is always in relation to the object's ORIGINAL X direction.
TransY	RW (see note)	Double	Moves the object in the vertical direction. Moves up for positive numbers and down for negative numbers. Displacement is in pixels. NOTE: This command is always in relation to the object's ORIGINAL Y direction.
Visible	R	Boolean	True if object's Visible state is on, False if not.
ZoomToFitOption	R (see note)	Boolean	True if object's Zoom To Fit state is on, False if not.

Note: For the main Display Object, the TransX and TransY properties are *Read Only*. Also, the ZoomToFitOption property is Read/Write for'(Display only)'.

The following is a complete list of properties defined for objects in GUS Displays. The table contains an indication of which properties are read only (1). The x's with () indicate that these properties will be removed at some time in the future and definitely should not be used in scripts.

Notes:

1	When the angle property is read in a script, the angle is reported as a negative value if the object has been rotated counterclockwise and reported as a positive value if the object has been rotated clockwise.
2	The grid on the background of a display is only displayed at build-time. If you create a script to show the grid, the script will execute, but the grid will not appear at runtime.
3	Changing the Angle, ScaleX, ScaleY, TransX and/or TransY of a Group or Embedded Display will not alter the location or size of any OLE or ActiveX components that it contains.

Properties	Property Type	Display	Embedded Display	Line	Rectangle	Rounded Rectangle	Ellipse	Polyline	Polygon	Open Bezier Curve	Closed Bezier Curve	Arc	Text	Bitmap	Group	OLE
Activation Verb																x
Angle	Double	(1)	(3)	x	x	x	x	x	x	x	x	X	x	x	(3)	
ArcShape	Long											X				
AutoParam Prompts	Boolean	(1)	(1)													
Background Color	Long	x														
Blink	Boolean	(x)	x	x	x	x	x	x	x	x	x	X	x	x	x	
Category	String	(1)														
CharSet	Integer												x		x	
Closed	Boolean							x	x	x	x					
Description	String	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	
Display Height	Long	(1)														
Display Width	Long	(1)														
Fill	Boolean	x	x		x	x	x	x	x	x	x	X	x	(x)	x	
FillColor	Long	x	x		x	x	x	x	x	x	x	X	x	(x)	x	
FillDirection	Integer	(x)	x		x	x	x	x	x	x	x	X	x	(x)	(x)	
FillFollows Rotation	Boolean	(x)	(x)		x	x	x	x	x	x	x	X	x	(x)	(x)	
FillPattern	Long	x	x		x	x	x	x	x	x	x	X	x	(x)	x	
FillPercent	Integer	(x)	(x)		x	x	x	x	x	x	x	X	x	(x)	(x)	
Font	String												x			
GridColor	Long	(1)														
GridHeight	Long	(1)														
GridWidth	Long	(1)														
Inherit	Boolean	(1)	x	x	x	x	x	x	x	x	x	X	x	(x)	x	
LineColor	Long	x	x	x	x	x	x	x	x	x	x	X	x	(x)	x	

Properties	Property Type	Display	Embedded Display	Line	Rectangle	Rounded Rectangle	Ellipse	Polyline	Polygon	Open Bezier Curve	Closed Bezier Curve	Arc	Text	Bitmap	Group	OLE
LineStyle	Long	x	x	x	x	x	x	x	x	x	x	X	x	(x)	x	
LineWidth	Long	x	x	x	x	x	x	x	x	x	x	X	x	(x)	x	
MultiLine	Boolean												x			
Name	String	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	
Reference Name	String	(1)	(1)												(x)	
Reference Path	String	(1)	(1)													
ScaleX	Double	(1)	(3)	x	x	x	x	x	x	x	x	X	x	x	(3)	
ScaleY	Double	(1)	(3)	x	x	x	x	x	x	x	x	X	x	x	(3)	
Selectable	Boolean	(x)	x	x	x	x	x	x	x	x	x	X	x	x	x	
ShowGrid	Boolean	(1)														
SnapToGrid	Boolean	(1)														
Text	String												x			
Text Alignment	Long												x			
TextBold	Boolean												x			
TextColor	Long												x			
TextHeight	Long												x			
TextItalic	Boolean												x			
TextStrike Out	Boolean												x			
Text Underline	Boolean												x			
TimeStamp (1)	Long	(1)	(1)													
TransX	Double	(1)	(3)	x	x	x	x	x	x	x	x	X	x	x	(3)	
TransY	Double	(1)	(3)	x	x	x	x	x	x	x	x	X	x	x	(3)	
Version Nbr(1)	Single	(1)	(1)													
Visible	Boolean	(x)	x	x	x	x	x	x	x	x	x	X	x	x	x	
X																x
Y																x
Zoom Factor		x														
ZoomToFit Option	Boolean	x	(x)												(x)	

5.2 Using the Drag and Drop Function with DropZone Control

5.2.1 Drag-and-drop support in GUS Displays

GUS R320 and later supports drag-and-drop of text from a GUS display target to an ActiveX drop target. The ActiveX drop target may be contained in a GUS display, a web page, or any ActiveX container (that is, a VB application). To support the drag-and-drop function, GUS

- Has a DragText scripting function that a display author may include on the OnLButtonDown or OnRButtonDown script of a GUS target to initiate drag mode, and
- Has implemented and installed an ActiveX Drop Zone Control called Honeywell GUS DropZone Control. Upon receiving the dropped text, the DropZone control raises an event, OnDropText(sText As String), allowing the container to respond to the drop of text on this control.

5.2.2 Drag-and-drop operation in GUS Displays

5.2.3 Drag Enter

Use GUS scripting extension function'DragText' in one of the mouse-down events, OnLButtonDown or OnRButtonDown, to enable drag entry.

5.2.4 Drop Target

Insert the Honeywell-supplied ActiveX drop-target control, Honeywell GUS DropZone Control, or any other ActiveX drop-target control capable of accepting clipboard formats, CF_TEXT or CF_UNICODETEXT, into an ActiveX container.

5.2.5 Sequence of events in the drag-and-drop operation

No.	Event
1	Press the left mouse button over a GUS display object that has a script handling the OnLButtonDown event. The OnLButtonDown script executes the'DragText' function.
2	While the button remains pressed, the'DragText' function attempts to enter drag mode based on a defined drag-entry criterion.
3	When drag mode is entered, the cursor shape changes per the drag-mode convention.
4	Move the mouse over the drop-target control and release the left button.
5	The drop-target control accepts the text and generates an event to the container. In the case of the Honeywell GUS DropZone Control, the event is OnDropText(sText As String).
6	The container script handles the OnDropText event.
7	The'DragText' function returns, and the invoking script on the GUS display object continues.

5.2.6 Criteria for entering drag mode

Drag mode will be entered based on parameters defined in the'DragText' function. The following two conditions are considered:

- Cursor movement area (for example, NArea parameter)

- Time (for example, NTime parameter)

In the default setting, the drag mode will enter after the mouse cursor moves out of the area of the GUS display object that was the target of the mouse button-down event.

5.2.7 Canceling the drag operation

Cancel the drag operation by pressing the Escape key or by pressing an'opposite' mouse button. For example, if drag has been started when holding the left mouse button down, clicking on the right mouse button will cancel the drag operation.

5.2.8 Handling mouse button - up events

If drag mode has been entered and is not cancelled, the corresponding mouse button-up event that terminates the operation will not be available to the originally hit target or any other application or target.

If a GUS display object has a script executing the'DragText' function on a mouse button-down event (for example, OnLButtonDown) and also has a script defining action on a mouse button-up event (for example, OnLButtonUp or OnLButtonClick), the mouse button-up event will not fire.

Use the'DragText' function return value to schedule execution of the mouse button-up event via scripting, if required. See “Example 4” under' “DragText examples” on page 129' later in this section.

5.2.9 Added GUS built-in function

DragText (statement)

Syntax

NResult = DragText(SText [, NFormat [,NArea [,NTime]]])

Description

Enables a GUS target to enter drag mode based on given drag entry criteria

Values of NResult

The returned value of the DragText function may have one of the following values.

Value of NResult	Description
0	drag mode did not enter or has been cancelled
1	successful drop
-1	no drop occurred; mouse cursor is currently positioned over the area defined by the originally hit display object
-2	no drop occurred; mouse cursor is outside of the area defined by the originally hit display object

Parameters of DragText

Parameter	Description
SText	A string containing the text-of-interest of the drag-and-drop operation

Parameter	Description
NFormat	This optional parameter defines the data format of the text. -1 text is offered in ANSI form (CF_TEXT). [default value] 0 text is offered in Unicode (CF_UNICODETEXT) Note: The Honeywell GUS DropZone Control currently does not accept Unicode text.
NArea	This optional parameter defines the area that the mouse-down cursor may stay within without causing drag mode to be entered. -1 The area is determined by the boundary of the display object that was the target of the initial mouse button-down event [default value] 0 [reserved] >0 The area of a square with the center at the current mouse position. The length of the square's side = 2* NArea pixels.
NTime	This optional parameter defines the length of time before drag mode will be entered automatically even without mouse movement. This parameter is defined in msecs. -1 Time out disabled [default value] 0 Enter drag mode immediately >0 Enter drag mode after this delay

Scripting using Parameters, NArea and NTime

Both NArea and NTime define a criterion for when to enter drag after a user has done a mouse-down action on a scripted drag target. Before authoring drag targets, define a standard concerning the criterion (NArea or NTime) and its value for when to enter drag for all drag targets on GUS displays. This standard causes consistent behavior and minimizes confusion for the operator.

Note: if both criteria are given (NArea and NTime), drag mode enters at the earliest opportunity (that is., as soon as one of the conditions allows it).

5.2.10 DragText examples

Example 1

A simple drag operation from a text object; drag mode will enter after cursor leaves text area

```
Sub OnLButtonDown()
Dim N as integer
N = DragText(me.text)
End Sub
```

Example 2

Drag operation using mouse right button

```
Sub OnRButtonDown()
Dim N as integer
N = DragText(me.text)
End Sub
```

Example 3

Drag text in Unicode; drag mode enters immediately

```
Sub OnLButtonDown()
Dim N as integer
N = DragText(text5.text, 0, -1, 0)
End Sub
```

Example 4

An object has scripted both mouse button-down and button-up events. If the drag operation has entered but did not complete and the mouse cursor is currently positioned over the object, schedule execution of the (otherwise missed) mouse button-up event.

```
Sub OnLButtonDown()
Dim N as integer
N = DragText(oleobject1.stringdataval)
If (N = -1) then OnLButtonClick
End Sub
Sub OnLButtonClick()
Msgbox' got a left button click'
End Sub
```

5.2.11 Honeywell GUS DropZone control

The Honeywell GUS DropZone control acts as a drop target for dragged text. Upon receiving dropped text, the control raises an event, OnDropText(sText As String), allowing the container to respond to the dropping of text on the control.

The Honeywell GUS DropZone control may be inserted into any ActiveX container (for example, a GUS display or web page).

An additional runtime feature enables the user to right-click on the control to see a single option, **Show Contents**. This option opens a dialog displaying the dropped text.

5.2.12 Buildtime and runtime appearance for DropZone control

The dropzone control appears the same at build-time and runtime as shown below.

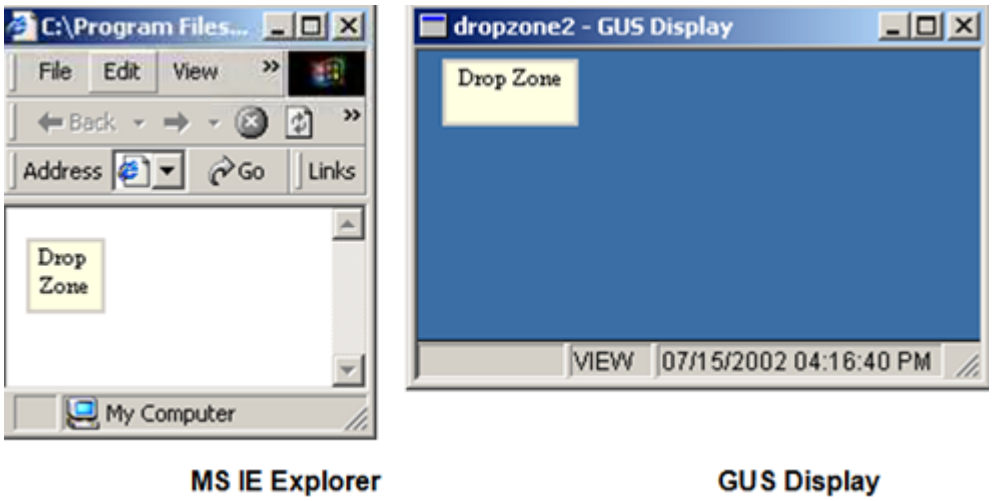


Figure 1: Example dropzone.ocx in Microsoft Internet Explorer and GUS Display

DropZone Scriptable Properties

Property	Type	Description
Caption	String	Defaults to'Drop Zone' and is the text displayed, centered, on the control. It is a read/write property that can be scripted.
BackColor	OLE_COLOR	Defaults to a ‘pale yellow' and is the background color of the control. It is a read/write property that can be scripted.

DropZone Property Pages

The dropzone provides two property pages, one provides build-time access to'Caption' property. The other provides standard access to the'Background' (color) property. They are pictured in Figures 2 and 3 below.

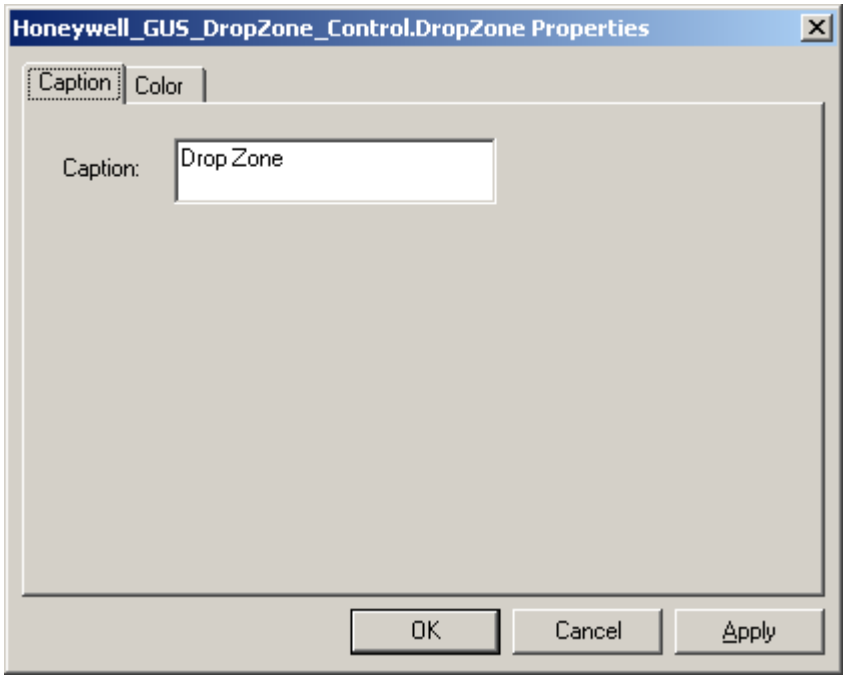


Figure 2: DropZone Caption Property Page

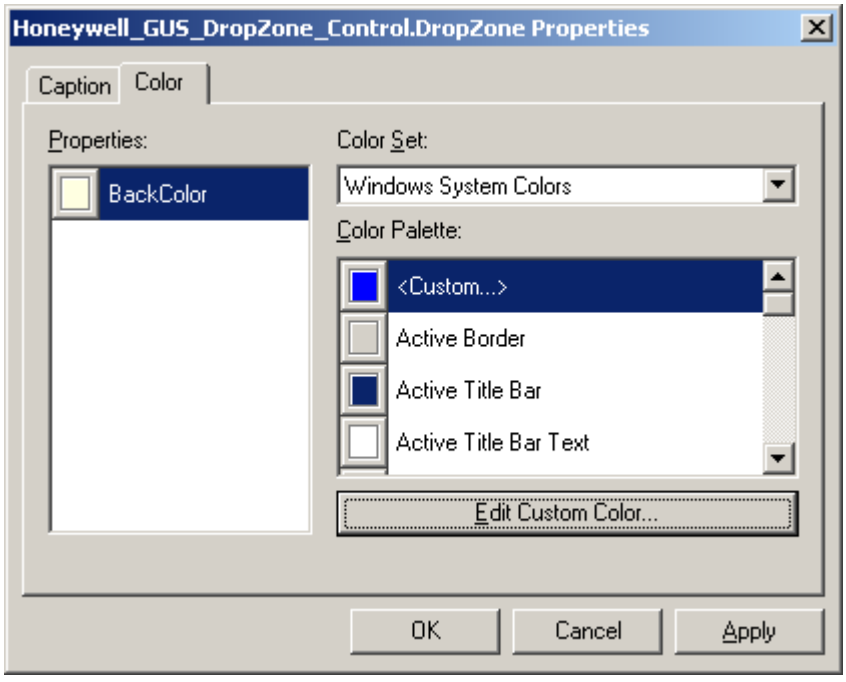


Figure 3: DropZone'(Back)Color' Property Page

5.2.13 DropZone Methods

No methods are provided.

5.2.14 OnDropText Event

The DropZone Control event, OnDropText, notifies the user when text has been dropped on the control. The user may define behavior via a script when this event is raised. An example of behavior that may be scripted for this event is to add a trace to the running trend for the tag name defined in the dropped text.

The syntax for this event is

```
OnDropText(sText As String)
```

where sText is a string containing the dropped text.

An example GUS script handling this event is as follows:

```
Sub OnDragText(sText as String)
Msgbox sText
End Sub
```

Note: The Honeywell GUS DropZone Control currently does not accept Unicode text.

5.2.15 DropZone control availability

The DropZone Control is installed when either the GUS Display Builder or the GUS Display Runtime is installed.

When installed, the dropzone.ocx control is available in the GUS **Insert Control** dialog as **Honeywell_GUS_DropZone_Control.DropZone**.

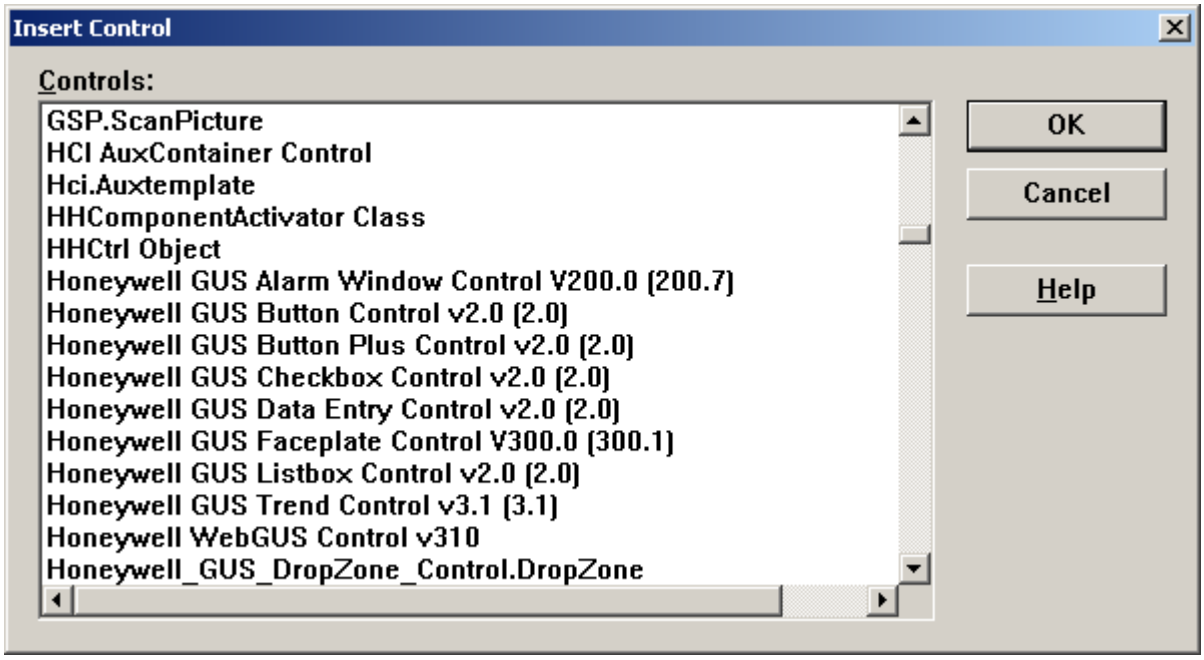


Figure 4: DropZone Control in GUS Insert Control

5.3 Primitives - Properties, Methods, and Events

This section includes tables that list the properties, methods, and events of the Primitives used in the Display Builder.

The following Primitives are described:

- Button Primitive
- ComboBox Primitive
- Data Entry Primitive
- Listbox Primitive

5.4 Button Primitive

Related topics

5.4.1 Overview of Button Primitive

Refer to “Button OLE Control” on page 151 for more information. You can also refer to Button Object General Properties,Command Button OLE Control Properties, and Using Honeywell OLE input controls and scripting in the *Display Builder User's Guide*.

Note: The Button Primitive is the suggested method for this functionality. If you have old OCXs, it is recommended you replace them with the Button Primitive.

5.4.2 Button specific events

Standard events as with other GUS primitives (e.g. OnLButtonUp, OnHelp, OnGotFocus etc.) These events may not be associated with a particular pushbutton if NumberButtons > 1.

Event	Parameters	Notes
OnSelectionOn	nIndex = 0-11	nIndex = 0-11 on selection of a particular pushbutton; visually'pushed-in' button.
OnSelectionOff	nIndex = 0-11	nIndex = 0-11 on de-selection of a particular pushbutton.
OnSelectionClick	nIndex = 0-11	nIndex = 0-11 for momentary buttons only; following a click on a pushbutton.

The events are generated depending on the button type and the current state of the button. For button types that maintain state, events are generated only if button state changes.

Pressing the left mouse button down over a particular pushbutton will cause a focus rectangle to appear on that button; momentary type button will appear as selected (pushed-in).

Mouse click over a particular pushbutton may generate the **OnSelectionClick**() (momentary type) or the **OnSelectionOn/Off**() (for other button types) events. The mouse left button down and up events must occur within the same pushbutton, otherwise no click will be recognized.

The events can also be generated by explicitly changing pushbutton state in the script through the

ButtonState(nIndex) property.

The events will not be generated if the pushbutton's **ButtonSelectable** property is set to FALSE.

Momentary buttons do not maintain state. The **OnSelectionOn**() and **OnSelectionOff** events are generated any time the visual state of the pushbutton changes; i.e. on initial mouse button press-down, and subsequent release; and after initial mouse button press down if the mouse is moved in and out of the pushbutton area while holding the button down. A mouse click over a pushbutton will generate the **OnSelectionClick**() event. The **ButtonState** property is not available.

Interlocked buttons maintain their state. Only one pushbutton may be in the ON state at a time. Clicking on a pushbutton that is in the OFF state will cause the currently selected pushbutton to become deselected (issuing the **OnSelectionOff**() event), and the clicked-on pushbutton becomes selected (issuing the **OnSelectionOn**() event). Clicking on a pushbutton which is already in the ON state will produce no change of state (and no

event). The **ButtonState** property may be used to change the state of the pushbutton (following the single-selected pushbutton rule), with **OnSelectionOn/Off**() events generated.

Latched buttons maintain state. Multiple pushbuttons may be set in the ON state. A mouse click on a pushbutton toggles the pushbutton state, issuing an appropriate event. The **ButtonState** property may be used to change the state of the pushbutton, with **OnSelectionOn/Off**() events generated.

5.4.3 Specific properties of Primitive Button

Unless otherwise specified, all properties are accessed through the (internal) Get() and Set() methods.

Property	Type	Comments
Appearance	long	0 - 3D, 1 - flat
ButtonNormalColor	long	Pushbutton color when not selected.
ButtonSelectedColor	long	Pushbutton color when selected.
NumberButtons	long 1-12	Read only.
ButtonIndex	long	Index of the selected pushbutton; -1 if none selected. The lowest index if latched button with multiple selections.

The following properties require providing an index(0-11) of the particular pushbutton. This is required even if the NumberButtons is 1.

Property	Type	Comments
ButtonText(index)	string	Text label on a particular pushbutton.
ButtonVisible (index)	BOOL	Visible state (0 = hidden) of a particular pushbutton.
ButtonSelectable (index)	BOOL	Selectability of a particular pushbutton; the non-selectable pushbutton will not react to the mouse left button down/up events and the OnSelectionOn/Off/Click events will not be generated. Text color is half-intensity of the current color of the buttonface.
ButtonState(index)	BOOL	Current state (0 = off) of the particular pushbutton. On set, if state changes (and ButtonSelectable is TRUE) then the OnSelectionOn() or OnSelectionOff() event will be generated. The ButtonState property does not apply to the momentary buttons.

5.4.4 Examples of Button Primitive

Set state ON for the third pushbutton, OFF for the others. Button type- interlocked (single selection only)

Buttonname. ButtonState (2) = true

[alternate method]

Buttonname.ButtonIndex = 2

Set state ON for the first and second buttons, OFF for the third. Button type- latched (multiple selection)

Buttonname.ButtonState(0) = true

Buttonname.ButtonState (1) = true

Buttonname.ButtonState (2) = false

[alternate method]

Buttonname.ButtonIndex = -1 'all buttons = OFF

Buttonname.ButtonState(0) = true

Buttonname.ButtonState (1) = true

Retrieve index and text of the currently selected pushbutton

```
CurrentlySelected = Buttonname.ButtonIndex
If CurrentlySelected = -1 then
Msgbox'none selected'
Else
Msgbox Buttonname.ButtonText(CurrentlySelected)
End if
```

Disable mouse input & grey second pushbutton

Buttonname. ButtonSelectable (1) = false

Hide second pushbutton

Buttonname. ButtonVisible (1) = false

On selection change, command a point if pushbutton text label shows:'OPEN'

```
Rem button type: latched or interlocked
Sub OnSelectionOn (nIndex as Integer)
If me.buttontext(nIndex) ='OPEN' then
Lcn.param.pv = OPENVALUE
End if
End Sub
```

5.5 ComboBox Primitive

Related topics

5.5.1 ComboBox Primitive Overview

The Combo box primitive is used to acquire operator input by means of selection or from a drop down list. Operation of the primitive is similar to the Visual Basic'Combo Box' control. The primitive is based on a standard Windows COMBOBOX control.

Multiple Combobox primitives may exist on a display.

At runtime, control that has input focus will receive keyboard input.

5.5.2 Combobox Button on the Graphic Primitives Toolbar

A new button is added to the graphics primitives toolbar; it depicts a Combobox control (Combobox primitive). When selected, the tool creates a single Combobox object in a fashion similar to the Text primitive tool. The control's window is visible, input focus may not be assigned to the control.

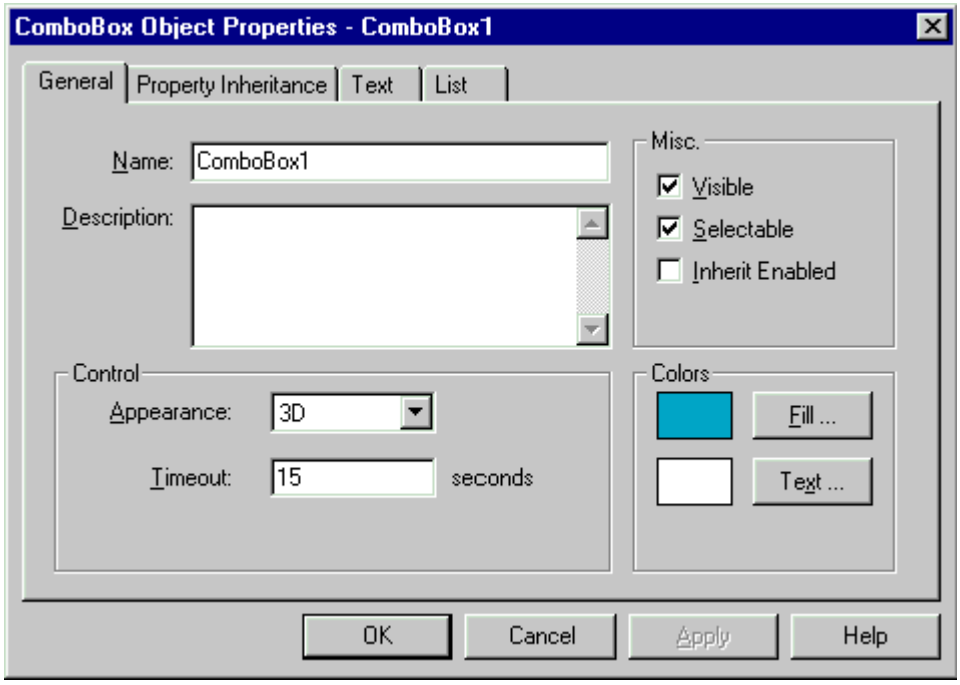
The tool creates a single selection Combobox control. Font size will not adjust automatically to fit the bounding rectangle. The height of the Combobox is determined by the font size of the Combobox and not its bounding box. Vertical scrollbars will appear on the Combobox.

5.5.3 ComboBox Button Specific Events

Standard events, as with other GUS primitives (OnLButtonUp, OnHelp, OnGotFocus, etc.), may not be associated with a particular pushbutton if NumberButtons > 1.

5.5.4 ComboBox Object Property Pages - The General Page

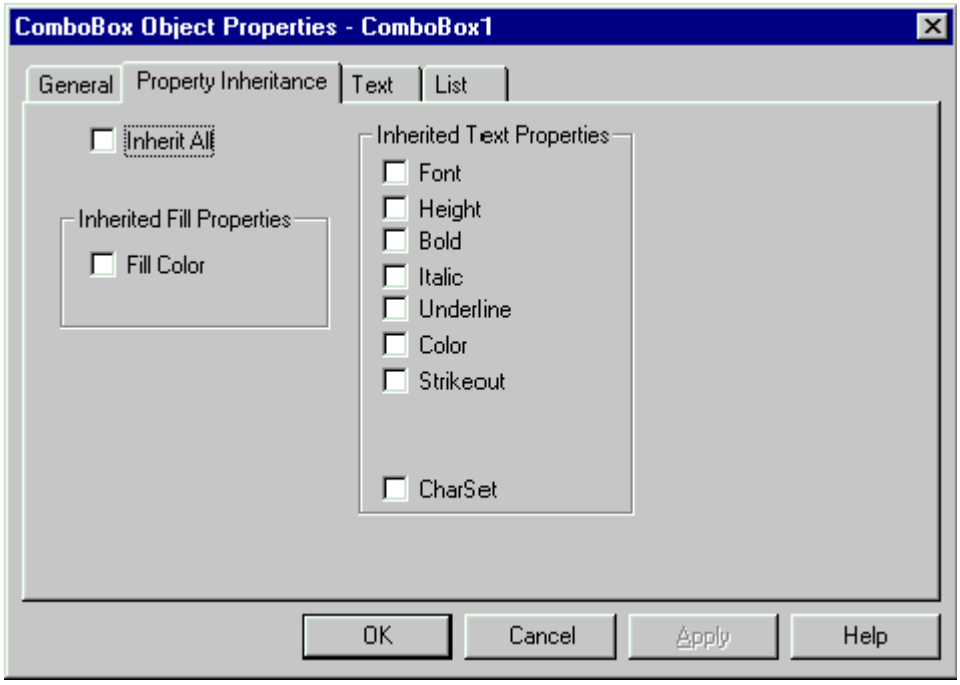
The page is shown below:



Property	Description
Name	[same as for other GUS drawing primitives]
Visible	[same as for other GUS drawing primitives]
Selectable	[same as for other GUS drawing primitives]
Inherit Enabled	[same as for other GUS drawing primitives]
Appearance	3D or Flat. Note: the flat style has the same appearance as the 3D style.
Timeout	Value in seconds. At runtime, timer must be activated by the script (using the property **TimeoutActive**). Timeout value = 0 disables timer activation.
Fill Color	Fill color for the edit and list portion of the Combobox.
Text Color	Text color for the edit and list portion of the Combobox.

ComboBox Object Property Pages - The Property Inheritance Page

The page is shown below:

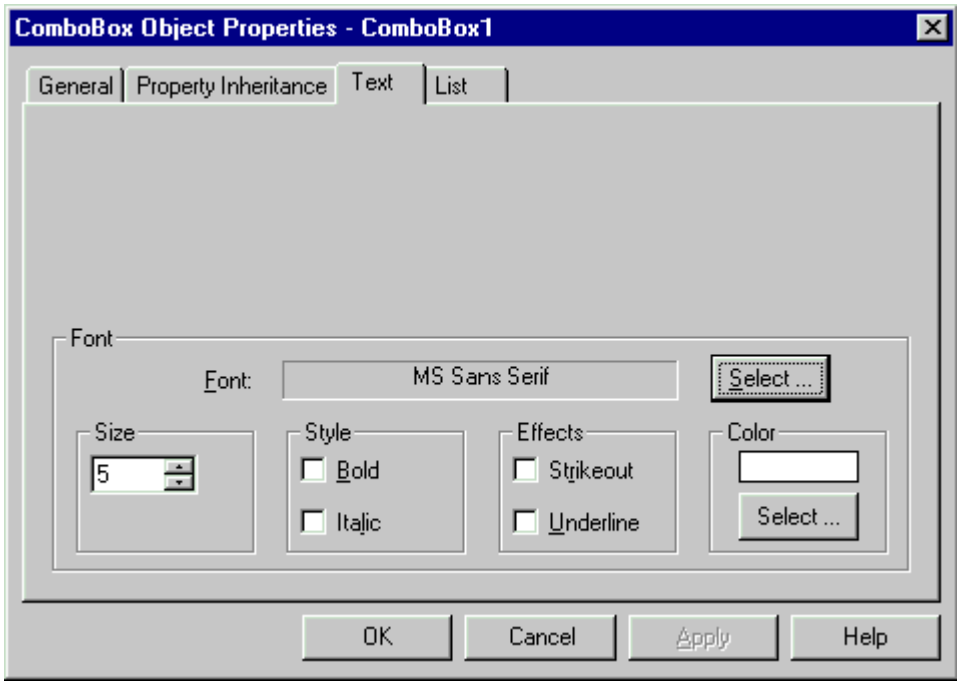


The Property Inheritance is not supported for the following properties:

Blink, Fill Pattern, Line Style, Line Color, Line Width, Alignment, Multiline, Reverse, Half Intensity. All other inheritable properties are the same as for other GUS drawing primitives.

5.5.5 ComboBox Object Property Pages - The Text Page

The page is shown below:



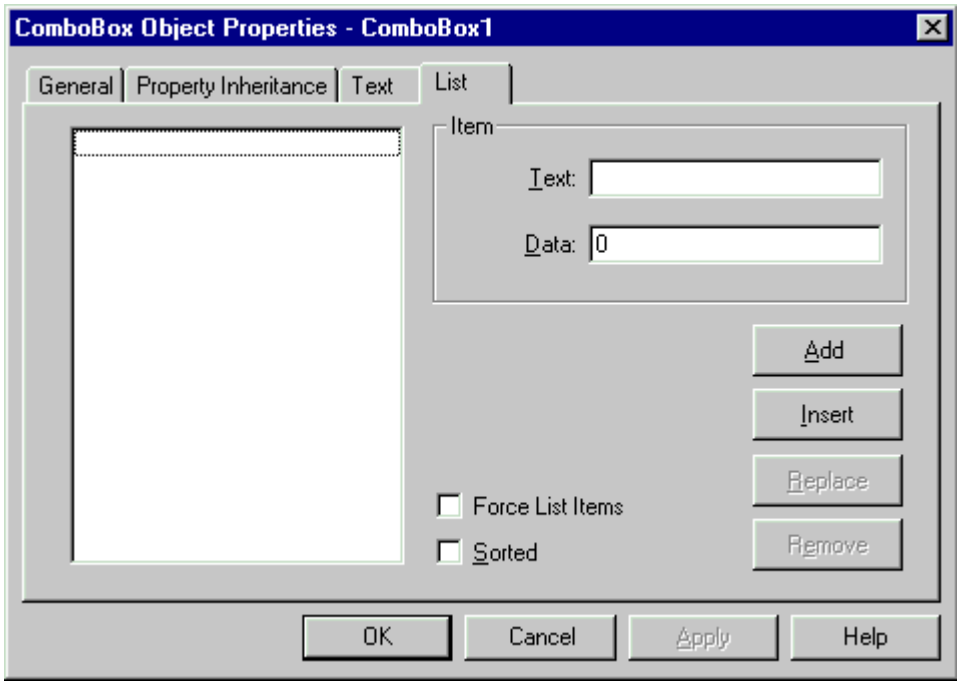
The Alignment, Multiline Text primitive properties are not supported.

List items are entered using the List property page.

Other text (font) properties are the same as for the Text primitive.

5.5.6 ComboBox Object Property Pages - The List Page

The page is shown below:



Property	Description
List	String list of all the currently defined strings that appear in the combobox list box.
Sorted	Sorts list items in alphabetical order if checked.
Force List Items	Force the user to choose an item in the list control of the Combobox at runtime. When this option is checked the user cannot enter information that does not appear in the Combobox list item. This property is not available at runtime.
Text	Text for the edit portion of the Combobox
Add	Command button. Will append currently defined item (**Item Text/Item Data**) to the Combobox. Adds to the end of the list, unless Combobox is sorted.
Insert	Command button. Will insert currently defined item (**Item Text/Item Data**) in front of the currently selected item in the Combobox. This command is not available if the Combobox is sorted.
Replace	Command button. Will replace currently selected item in the Combobox with currently defined item (**Item Text/Item Data**). This command is not available if the Combobox is sorted.
Delete	Command button. Will remove currently selected item from the Combobox.
Data	Integer value that may be associated with the particular text entry. At runtime, script may set and retrieve value of this field and take appropriate action without retrieving item's text.

All items in the list portion of the Combobox are left justified. The edit portion of the Combobox is left justified.

The **Item Text** and **Data** fields are reloaded each time new list item is selected.

5.5.7 ComboBox Control Behavior

The Combobox primitive provides number of built-in behaviors, independent of GUS scripting:

- Keyboard character entry selects an item that matches the first letter.
- Arrow keys change selection.

Selected item displays in reverse video. The background color is selectable through the system color settings dialog.

The Combobox acquires input focus through:

- A left button mouse click,
- The TAB key, and
- The Script SetFocus method.

Only a selectable and visible control can acquire input focus.

The control loses input focus when another GUS Display object or display background is selected.

The runtime Z-order position is always on top of other non-window based GUS primitives. Z-order position relative to other window-based objects (data entry, Combobox, ActiveX controls) may not be changed.

If timeout has been activated on a control, then any keyboard entry and mouse event while the control has input focus will restart the timeout timer.

5.5.8 Script Properties

The properties can be read and written to through the built-in Set/Get property methods. Unless otherwise specified, all properties may be set at runtime.

5.5.9 Common GUS object's properties

Visible

Selectable

FillColor = control background color

Text = currently selected item (Get method only)

TextColor

Font properties (e.g. Font, FontSize etc.)

Supported Inherit… properties.

TransX

TransY

Note: the Angle and ScaleX/ScaleY properties are not supported

5.5.10 Properties Specific to the Combobox Primitive

Property	Type	Comments
Appearance	long	0 - 3D, 1 - flat
ItemData(index)	long	Returns or assigns a numeric id value for an item in a Combobox ; index is 0-based.

Property	Type	Comments
List (index)	string	Returns text of an item in a Combobox; index is 0-based. NOTE 1: The Text property can be used to retrieve currently selected item. NOTE 2: Use AddItem/InsertItem/DeleteItem methods to manipulate items in a Combobox. NOTE 3: This is a READ ONLY property. Assigning a String to List(index) will result in an error.
ListCount	long	Returns number of items in a Combobox.
ListIndex	long	Returns or sets index of the currently selected item in a Combobox; index is 0-based. -1 indicates no selection.
NewIndex	long	Returns index of the item most recently added to the Combobox through the AddItem method (usually used in conjunction with ItemData, when the list is sorted). Index is 0-based.
Sorted	BOOL	Indicates if Combobox items are sorted in alphabetical order.
Text	string	Set's the edit controls text. Text is only valid if the'Force List Items' is not enabled.
Timeout	long	>= 0; in seconds. Requires activation through the TimeoutActive property. Value = 0 disables activation of the timer. Changing the Timeout resets the timer.
TimeoutActive	BOOL	Activates timeout timer if TRUE. Stops timeout timer if FALSE.

5.5.11 Script Methods

Method	Comments
AddItem string	A text item is added to the list (at the end of the list or sorted in, if Sorted property is true). The NewIndex property is updated with the index of the item on the list. Value of the NewIndex property can subsequently be used to set item data if required. See the ItemData(index) property in the “Properties Specific to the Combobox Primitive” on page 141 table.
Clear	Clears contents of the Combobox list control and edit control.
index = FindItem (string)	Returns index of the first item matching given text. String compare is case insensitive. Returns -1 if item has not been found.
InsertItem index, string	A text item is inserted at indicated position (index is 0 based). This method should be avoided if Sorted property is true; Combobox will not re-sort after inserting an item.
RemoveItem index	An item is removed from the list (index is 0 based).
SetFocus	After invoking the SetFocus method, user (keyboard) input is directed to the control. Control must be selectable and visible to receive input focus.

5.5.12 Script Events

Event	Comments
Arrow Key Movement	Movement of the arrow key (change of selection) produces the OnLButtonClick event.
Enter Key	The Enter key produces the OnLButtonDoubleClick event.
Mouse Events	The mouse events restart timeout timer, if it has been activated.
OnChange	Occurs when the edit portion of the Combobox changes.
OnDropDown	Occurs when the list portion of the Combobox is about to become visible.

Event	Comments
OnGotFocus	Occurs when control receives input focus - either by user action (click on the control, Tab key movement) or as a result of the SetFocus method. Only a visible and selectable control may receive input focus.
OnLButtonClick	Indicates possible change of selection.
OnLButtonDoubleClick	Indicates that the selection has been made. Stops the timeout timer.
OnLostFocus	Occurs when control loses input focus.
OnTimeout	Occurs when timeout has expired. See the Timeout and TimeoutActive properties in the “Properties Specific to the Combobox Primitive” on page 141 table.
Other GUS standard events	OnDataChange, OnPeriodicUpdate, etc.

Example 1

Update a display text (Text1) object with text from the currently selected item in a Combobox.

```
Rem Combobox 's script
Sub OnChange
text1.text = me.text
end sub
```

Example 2

Make sure none of the Combobox items is selected

```
Sub .......
me.listindex = -1
end sub
```

Example 3

Replace third item in a Combobox list (text only; itemdata is not used)

```
Sub .......
Combobox 1.removeitem 2 ‘ index is 0 - based....
Combobox 1.insertitem 2, text2
end sub
```

Example 4

On some object's script, reload Combobox. Combobox is sorted (builder). Each text item gets a numerical id assigned, which will be used when operator makes the selection (rather then retrieving selected text).

```
Rem some object's event
Sub ......
Combobox 1.clear ‘ clear the contents
Combobox 1.additem'item1text' ‘ add item 1
Combobox 1.itemdata(Combobox 1.newindex) = number1 ‘ must do right after additem
Combobox 1.additem'item2text' ‘ add item 2
Combobox 1.itemdata(Combobox 1.newindex) = number2 ‘ must do right after additem
......
end sub
```

5.6 Data Entry Primitive

Refer to “Data Entry OLE Control” on page 154 for more information. You can also refer to Data Entry Object General Properties Data Entry OLE Control Properties Using Honeywell OLE input controls and scripting in the *Display Builder User's Guide*.

Note: The Data Entry Primitive is the suggested method for this functionality. If you have old OCXs, it is recommended you replace them with the Data Entry Primitive.

Related topics

“DataEntry Script Properties” on page 144
“DataEntry Common GUS object's properties” on page 144
“Properties Specific to the Data Entry Primitive” on page 144
“DataEntry Script Methods” on page 145
“DataEntry Script Events” on page 145
“DataEntry Specific Events” on page 145

5.6.1 DataEntry Script Properties

The properties can be read and written to through the built-in Set/Get property methods. Unless otherwise specified, all properties may be set at runtime.

5.6.2 DataEntry Common GUS object's properties

Visible

Selectable

FillColor = control background color

Text = current contents of the control

TextColor

Font properties (e.g. **Font**, **FontSize** etc.)

Supported Inherit… properties.

TransX

TransY

Note: the **Angle** and **ScaleX**/**ScaleY** properties are not supported

5.6.3 Properties Specific to the Data Entry Primitive

Property	Type	Comments
Appearance	long	0 - 3D, 1 - flat
Timeout	long	>= 0; in seconds. Requires activation through the TimeoutActive property. Value = 0 disables activation of the timer.
TimeoutActive	BOOL	Activates timeout timer if TRUE. Stops timeout timer if FALSE.
MaxLength	long	>=0. Limits number of characters the control can hold. Value = 0 : no limit. If Text property set results in text longer than MaxLength, the control will allow deleting, but no adding text.

Property	Type	Comments
SelStart	long	Returns or sets the starting point of text selected; indicates the position of the insertion bar if SelLength is 0; changing value of the SelStart sets SelLength = 0. Internally, setting Text property positions the insertion bar at the end of the string (SelStart = length of text, SelLength = 0).
SelLength	long	Returns or sets number of characters selected (highlighted). Must be >= 0.

5.6.4 DataEntry Script Methods

SetFocus	After invoking the **SetFocus** method, user (keyboard) input is directed to the control. Control must be selectable and visible to receive input focus.

5.6.5 DataEntry Script Events

The GUS standard mouse left and right button events (e.g. OnLButtonClick etc.). Note: The mouse events restart timeout timer, if it has been activated.

Other GUS standard events (e.g. OnDataChange, OnPeriodicUpdate etc.)

5.6.6 DataEntry Specific Events

OnTimeout	Occurs when timeout has expired. See the Timeout and TimeoutActive properties in the “Properties Specific to the Data Entry Primitive” on page 144 table.	
OnEnter	Occurs when user hits the ENTER key. Stops timeout timer.	
OnGotFocus	Occurs when control receives input focus - either by user action (click on the control, Tab key movement) or as a result of the SetFocus method. Only a visible and selectable control may receive input focus.	
OnLostFocus	Occurs when control loses input focus.	
OnMaxLength	Occurs on each character when typing beyond specified MaxLength limit.	
OnKeyPress(KeyAscii as Integer)	Occurs on receiving character from the keyboard. Restarts timeout timer if it has been activated. The following keys produce the OnKeyPress event:	
	Any character key	
	BACKSPACE	0x08
	ENTER (carriage return)	0x0D
	ESC	0x1B
	SHIFT+ENTER (linefeed)	0x0A

DataEntry Example

Click on a object (button, text etc.) activates the DataEntry control. The control is normally invisible. Timeout set at 15 seconds. Click on any other selectable display object while the control is active results in hiding the control. Enter key is used to process input data.

Some Object's Script

```
Sub OnLButtonClick
…
dataentry1.selectable = true
dataentry1.visible = true
dataentry1.setfocus
```

```
…
end sub
```

The Data Entry Object Events

```
Sub OnGotFocus
me.text ='' ‘ clear
me.timeoutactive = true ‘ activate timeout
end sub
Sub OnLostFocus
me.selectable = false
me.visible = false
end sub
Sub OnEnter
Rem obtain input
Var = me.text
Rem convert/validate; if error notify the user; if OK etc.
……
end sub
Sub OnTimeout
Rem user took to long….
me.selectable = false
me.visible = false
end sub
```

5.7 Listbox Primitive

/Refer to “Listbox OLE Control” on page 155 for more information. You can also refer to ListBox /Object General Properties,.ListBox OLE Control Properties and Honeywell OLE input controls and scripting in the *Display Builder User's Guide*.

Note: The ListBox Primitive is the suggested method for this functionality. If you have old OCXs, it is recommended you replace them with the ListBox.

Related topics

“Listbox Control Behavior” on page 147
“ListBox Script Properties” on page 147
“Listbox Common GUS Object Properties” on page 147
“Properties Specific to the Listbox Primitive” on page 148
“Listbox Script Methods” on page 148
“Listbox Script Events” on page 149
“Listbox Specific Script Events” on page 149

5.7.1 Listbox Control Behavior

The Listbox primitive provides a number of built-in behaviors, which are independent of GUS scripting:

- keyboard character entry that selects an item that matches the first letter,

- arrow keys change selection,

- left button click that selects an item.

The selected item is displayed in reverse video. The background color is selectable through the system color settings dialog.

Mouse left button double-click is usually reserved to commit to the current selection.

The Listbox acquires input focus through:

- a left button mouse click,

- the TAB key,

- the Script SetFocus method.

Only a selectable and visible control may acquire input focus.

The control looses input focus when another GUS Display object or display background is selected.

The runtime Z-order position is always on top of other non-window based GUS primitives. Z-order position relative to other window-based objects (data entry, listbox OLE controls) cannot be changed.

If timeout has been activated on a control, then any keyboard entry and mouse event, while the control has input focus, restarts the timeout timer.

5.7.2 ListBox Script Properties

The properties can be read and written to through the built-in Set/Get property methods. Unless otherwise specified, all properties may be set at runtime.

5.7.3 Listbox Common GUS Object Properties

Visible

Selectable

FillColor = control background color

Text = currently selected item (Get method only)

TextColor

Font properties (e.g. **Font**, **FontSize** etc.)

Supported Inherit… properties.

TransX

TransY

Note: the **Angle**and **ScaleX**/**ScaleY** properties are not supported

5.7.4 Properties Specific to the Listbox Primitive

Property	Type	Comments
Appearance	long	0 - 3D, 1 - flat
ItemData(index)	long	Returns or assigns a numeric id value for an item in a listbox; index is 0-based.
List(index)	string	Returns text of an item in a listbox; index is 0-based. Note: the Text property can be used to retrieve currently selected item. Note: Use AddItem/InsertItem/DeleteItem methods to manipulate items in a listbox.
ListCount	long	Returns number of items in a listbox.
ListIndex	long	Returns or sets index of the currently selected item in a listbox; index is 0-based. -1 indicates no selection.
NewIndex	long	Returns index of the item most recently added to the listbox through the AddItem method (usually used in conjunction with ItemData, when the list is sorted). Index is 0-based.
Sorted	BOOL	Indicates if listbox items are sorted in alphabetical order.
Timeout	long	>= 0; in seconds. Requires activation through the TimeoutActive property. Value = 0 disables activation of the timer.
TimeoutActive	BOOL	Activates timeout timer if TRUE. Stops timeout timer if FALSE.

5.7.5 Listbox Script Methods

AddItem *string*	A text item is added to the list (at the end of the list or sorted in, if Sorted property is true). The NewIndex property is updated with the index of the item on the list. Value of the NewIndex property can subsequently be used to set item data if required. See the ItemData(index) property in the “Properties Specific to the Combobox Primitive” on page 141 table.
Clear	Clears contents of the listbox.
index = FindItem (*string*)	Returns index of the first item matching given text. String compare is case insensitive. Returns -1 if item has not been found.
InsertItem *index, string*	A text item is inserted at indicated position (index is 0 based). This method should be avoided if Sorted property is true; listbox will not re-sort after inserting an item.
RemoveItem *index*	An item is removed from the list (index is 0 based).
SetFocus	After invoking the **SetFocus** method, user (keyboard) input is directed to the control. Control must be selectable and visible to receive input focus.

5.7.6 Listbox Script Events

The GUS standard mouse left and right button events (e.g. OnLButtonClick etc.). The mouse events restart timeout timer, if it has been activated.

The OnLButtonClick event indicates possible change of selection.

The OnLButtonDoubleClick indicates that the selection has been made. Stops the timeout timer.

Selected keyboard entries are mapped into mouse events:

Movement of the arrow key (change of selection) produces the OnLButtonClick event.

The Enter key produces the OnLButtonDoubleClick event.

Other GUS standard events (e.g. OnDataChange, OnPeriodicUpdate etc.)

5.7.7 Listbox Specific Script Events

OnGotFocus	Occurs when control receives input focus - either by user action (click on the control, Tab key movement) or as a result of the SetFocus method. Only a visible and selectable control may receive input focus.
OnLostFocus	Occurs when control loses input focus.
OnTimeout	Occurs when timeout has expired. See the Timeout and TimeoutActive properties in the “Properties Specific to the Listbox Primitive” on page 148 table.

Example 1

Update a display text (Text1) object with text from the currently selected item in a listbox.

```
Llistbox Script
Sub OnLButtonClick
text1.text = me.text
end sub
Alternate Solution
Sub OnLButtonClick
text1.text = me.list(me.listindex)
end sub
```

Example 2

```
Ensures that none of the listbox items is selected.
Sub .......
me.listindex = -1
end sub
```

Example 3

```
Replace the third item in a listbox (text only; itemdata is not used).
Sub .......
listbox1.removeitem 2 ‘ index is 0 - based....
listbox1.insertitem 2, text2
end sub
```

Example 4

On some object's script, reload listbox. Listbox is sorted (builder). Each text item gets a numerical id assigned, which will be used when operator makes the selection (rather then retrieving selected text).

```
Some Object's Event
Sub ......
listbox1.clear ‘ clear the contents
listbox1.additem'item1text' ‘ add item 1
listbox1.itemdata(listbox1.newindex) = number1 ‘ must do right after additem
listbox1.additem'item2text' ‘ add item 2
```

```
listbox1.itemdata(listbox1.newindex) = number2 ‘ must do right after additem
.......
end sub
Listbox1 Event
Sub OnLButtonDblClick
rem retrieve numerical id and process depending on this value
itemid = me.itemdata(me.listindex).
end sub
```

5.8 OLE Controls - Properties, Methods, and Events

This section includes tables that list the properties, methods, and events of the OLE controls used in the Display Builder. The following controls are mentioned:

OLE Controls

- Button OLE Control
- ButtonPlus OLE Control
- Checkbox OLE Control
- Data Entry OLE Control
- Listbox OLE Control
- Trend OLE Control
- Trend Scripting Examples
- Scripting an Example of an Embedded Trend

5.8.1 Button OLE Control

Refer to Button Object General Properties ,Command Button OLE Control Properties, and, Using Honeywell OLE input controls and scripting in the *Display Builder User's Guide* for more information.

Note: The Button Object is the suggested recommendation for this functionality. If you have old OCXs, it is recommended you replace them with the Button primitive.

Properties	Type	Default Value	Valid Values
Description	string	empty string	valid ASCII characters.
Enabled	Boolean	true	0 - false (disabled) 1 - true (enabled)
Name	string	contains the name of the object	user-defined
Parent (read only)	Object ID		
Text	string	'Button'	valid ASCII characters.
Visible	Boolean	true	0 - false (not visible) 1 - true (visible)
Methods	Parameters	Return	
SetFocus()			sets the keyboard focus to the control
X	Long ???		sets/reads horizontal position of display's left side to the upper-left corner of OLE object or control expressed in pixels
Y	Long ???		sets/reads vertical position of display's top side to the upper-left corner of OLE object or control expressed in pixels
Events	Parameters		Notes
ButtonOn	none		mouse click & space bar activated
Click	none		mouse click activated
OnDisplayShutdown	none		

Properties	Type	Default Value	Valid Values
OnDisplayStartup	none		
OnHelp	none		
OnPeriodicUpdate	none		

5.8.2 ButtonPlus OLE Control

For more information, refer to ButtonPlus OLE Control Properties, and Using Honeywell OLE input controls and scripting in the *Display Builder User's Guide*.

ButtonPlus Properties	Type	Default Value	Valid Values
BackColor	long	ambient background color	valid color values
Border	Boolean	0 - false	0 - false 1 - true
BottomRightColor	long	black	valid color values
Description	string	empty string	any string
Enabled	Boolean	true	0 - false (disabled) 1 - true (enabled)
FillColor	long	ambient background color	valid color values
ForeColor	long	ambient foreground color	valid color values
Font	LPFONTDISP	ambient font	valid font values.
HorizontalSpacing	short	2	0 -50
NumberButtons	short	1	1 - 8
Orientation	short	0 - vertical	0 - vertical 1 - horizontal
SelectedFillColor	long	ambient background color	valid color values
SelectedTextColor	long	ambient foreground color	valid color values
Selection	short	0	0 - NumberButtons
TextColor	long	ambient foreground color	valid color values
TopLeftColor	long	white	valid color values
Type	short	0 - momentary	0 - momentary 1 - interlock 2 - latch
VerticalSpacing	short	2	0 - 50

ButtonPlus Events	Parameters	Return	Notes
ButtonOff	none		mouse click & space bar activated
ButtonOn	none		mouse click & space bar activated
Click	none		mouse click activated

5.8.3 ButtonPlus Selection Command

To initialize the member buttons in a ButtonPlus object, enter this script for each button and state:

Selection X, Y Where:

'X' represents the number of the button from 0 to 7 and

'Y' represents the initialized state where 1=on and 0=off.

For example, a display contains a ButtonPlus object named'Motor_Btn' with two member buttons used to turn a motor on or off.

The buttons are to be interlocked so button #1 turns the motor on and button #2 turns the motor off.

When the display starts, we assume:

- The state of the motor is given by the value of'Motor1.State' and
- The state of the button must reflect the state of the motor.

The following script initializes the state of the ButtonPlus object to reflect the state of the motor:

```
Sub OnDisplayStartup
if lcn.Motor1.State ='on' then
Motor_Btn.Selection 1, 1
‘Show Motor On
Motor_Btn.Selection 2, 0
Else
Motor_Btn.Selection 1, 0
‘Show Motor Off
Motor_Btn.Selection 2, 1
End if
End Sub
```

5.8.4 Checkbox OLE Control

For more information, refer to CheckBox OLE Control Properties and Using Honeywell OLE input controls and scripting in the *Display Builder User's Guide*.

Checkbox Properties	Type	Default Value	Valid Values
Description	string	empty string	valid ASCII characters.
Enabled	Boolean	true	0 - false (disabled) 1 - true (enabled)
FillColor	long	ambient background color	valid color values
Name	string	OLE Object#, where # represents an application-supplied number	user-defined
Parent (read only)	Object ID		
State	Boolean	0	0 - checkbox off 1 - checkbox on
Text	string	'Checkbox'	valid ASCII characters.
TextColor	long	ambient foreground color	valid color values
Visible	Boolean	true	0 - false (not visible) 1 - true (visible)

Checkbox Methods	Parameters	Return
SetFocus()	none	sets the keyboard focus to the control

Checkbox Events	Parameters	Notes
Click	none	mouse click activated
CheckboxOff	none	mouse click & space bar activated
CheckboxOn	none	mouse click & space bar activated
OnDisplayShutdown	none	
OnDisplayStartup	none	
OnHelp	none	
OnPeriodicUpdate	none	

5.8.5 Data Entry OLE Control

For more information, refer to Data Entry Object General Properties, Data Entry OLE Control Properties, and Using Honeywell OLE input controls and scripting in the *Display Builder User's Guide*.

Note: The Data Entry Object is the suggested recommendation for this functionality. If you have old OCXs, it is recommended you replace them with the Data Entry primitive.

Data Entry Properties	Type	Default Values	Valid Values
Description	string	empty string	valid ASCII characters
Enabled	Boolean	true	0 - false (disabled) 1 - true (enabled)
ErrorCondition	Boolean	false	0 - false (not in error condition state) 1 - true (in error condition state
ErrorFillColor	long	ambient background color	valid color values
ErrorTextColor	long	ambient foreground color	valid color values
Text	string	empty string	valid ASCII characters
Name	string	OLE Object#, where # represents an application-supplied number	user-defined
NormFillColor	long	ambient background color	valid color values
NormTextColor	long	ambient foreground color	valid color values
Parent (read only)	Object ID		
SelLength	integer	0	0 through the length of the text in the field
SelStart	integer	0	0 through the length of the text in the field
SelText	string	empty string	
Timeout	Boolean	true	0 - false (timer disabled) 1 - true (timer enabled) Note: An example of Timeout Usage follows this table.
TimeoutTime	integer	15	1 - 32767
Type	integer	string	0 - integer 1 - double integer 2 - real (32 bit) 3 - double-real (64 bit) 4 - string

Data Entry Properties	Type	Default Values	Valid Values
Visible	Boolean	true	0 - false (not visible) 1 - true (visible)

Timeout Usage Example:

Button1 is scripted to do the following OnLButtonClick:

OleObject1.setfocus

Add the following line afterwards to ensure that the timeout clock will start:

OleObject1.text ="

Data Entry Methods	Parameters	Description
SetFocus()	none	sets the keyboard focus to the control
BadEntry	string	Sent on the enter key. Note: String has no data type-checking.
Click	none	mouse click activated
GoodEntry	string	Sent on the enter key. Note: String has no data type-checking.
OnHelp	none	
OnPeriodicUpdate	none	
OnDisplayStartup	none	
OnDisplayShutdown	none	
Timeout	none	Timer time out

5.8.6 Listbox OLE Control

For more information, refer to ListBox Object General Properties, ListBox OLE Control Properties, and Using Honeywell OLE input controls and scripting in the *Display Builder User's Guide*.

Note: The ListBox Object is the suggested recommendation for this functionality. If you have old OCXs, it is recommended you replace them with the ListBox primitive.

Listbox Properties	Type	Default Values	Valid Values
BorderStyle	integer	1 - Normal border	0 - No border 1 - Normal border
Count	short	0	>= 0 (read only)
Description	string	empty string	don't know
Enabled	Boolean	true	0 - false (disabled) 1 - true (enabled)
FillColor	long	containers ambient back color	valid color values
Name	string	OLE Object#, where # represents an application-supplied number	user-defined
Parent (read only)	Object ID		
SelectedID	Integer	0	ID of selected enumeration member if the type of data in the list is enumeration

Listbox Properties	Type	Default Values	Valid Values
SelectedIndex	Integer	empty string	Index number of selected list item (0 - (no. of list items-1))
SelectedString	String	-1	Text of selected list item
SetId	Integer	-1	ID of enumeration set if the type of date in the list is enumeration
SetIndex	Integer	?	Index number of a list item (0 - (no. of list items-1))
Sort	Boolean	false	0 - false (disabled) 1 - true (enabled)
TextColor	long	containers ambient fore color	valid color values
Timeout	Boolean	true	0 - false (disabled) 1 - true (enabled)
TimeoutTime	integer	15	1-32267
Type	long	string	0 - string 1 - enumeration
Visible	Boolean	true	0 - false (not visible) 1 - true (visible)
Delete	short	Boolean	(short < Count) AND (short >= 0)
DeleteAll	none	Boolean	
DeleteFirst	none	Boolean	
DeleteLast	none	Boolean	
Get	short	string	(short < Count) AND (short >= 0)
GetFirst	none	string	
GetLast	none	string	
Insert	short, string	Boolean	(short < Count) AND (short >= 0)
InsertFirst	string	Boolean	don't know
InsertLast	string	Boolean	don't know
SetFocus()			sets the keyboard focus to the control
Listbox Events	Parameters		Description
Click	none		mouse click activated
DblClick	none		double-click
OnDisplayShutdown			
OnDisplayStartup			
OnHelp	none		
OnPeriodicUpdate			
SelectionChange	string, long		string selected in listbox
Timeout	none		timeout time exceeded before user interaction

5.8.7 Trend OLE Control

This section includes a number of tables that describe the properties and other parameters of the trending function used by the Display Builder. The following items are included:

- Extended Control Properties
- Properties
- Methods
- Events

5.8.8 Trend Properties

Most Trend Control properties have both read and write capability, and are available at buildtime and runtime. The following table describes the characteristics of each property.

User changes of most properties at runtime via scripting result in an immediate modification of the property. However, some properties require verification by the LCN of the requested change before the property value can be set by the Trend Control. Therefore, these properties are not immediately updated with the requested value.

Additionally, the LCN may reject the property change if it is an invalid request (e.g. setting the DataSource of a trend value to HM, when the point is not historized in the History Module). The user should use the OnPropertyChange() event to receive notification when the property has been changed. This is especially important for the property changes that are enforced by the LCN:

- DataSource
- ScrollBackwardTime
- ScrollForwardTime
- TimeBase
- TimeBaseIndex

Note: Properties designated with a parameter of nTraceID are trace specific and require this number when scripting; e.g.'Oleobject1.YRangeHigh(1) = 60' (where'1' represents an nTraceID of one.)

Refer to “Trend Scripting Examples” on page 173 for more information. You can also refer to Trend OLE Control Properties in the *Display Builder User's Guide*.

Trend Properties	Definition	Type	Default Value	Valid Values
AxesColor	Color of the X and Y axes. Available at both buildtime and runtime.	OLE_COLOR (long)	R/G/B=0/0/0 33554432 (BLACK)	GUS Color Palette
BackgroundColor	Color of the background of the trend. Available at both buildtime and runtime.	OLE_COLOR (long)	R/G/B= 187/187/187 45857723 (GRAY)	GUS Color Palette

Trend Properties	Definition	Type	Default Value	Valid Values
Chart Type	Specifies the type of chart	short	0	0 = Trend (x-axis is time; all of the traces show as trend lines) 1 = XY-Plot (x-axis is one of the configured traces; all of the other traces are plotted based on the value of the x variable at the time of the y value measurement.) 2 = Operation Point Plot (Similar to xy-plot, but only the most recent value of each Trace is plotted.)
Correlation Coefficient	Runtime property (read-only) specifies the calculated correlation of a given trace compared to current X trace. Set to FLT_MAX if no valid calculation is available. Note 1: The OnPropertyChange() event for this property is only fired for active traces. Note 2: Valid only for xy-plots.	double Parameter: short nTraceID	0.0	-1.0 to 1.0 or FLT_MAX if not valid where FLT_MAX =3.402823466e+38F
DataSource	Data source associated with a given trace. Available at both buildtime and runtime. When storing to this property, use the OnPropertyChange() event to confirm the property change has been accepted by the LCN and to receive the actual property value.	short Parameter: short nTraceID	0	0 = Real-Time, 1 = History Module, 2 = Hiway Gateway, 3 = Automatic Selection (Tries the History Module first, for all timebases except 1 minute and 2 minutes. For the 1 minute and 2 minute timebases, Real-Time is selected.)
Description	User description of the Trend Control object. Available at both buildtime and runtime.	Get = BSTR Set = LPCTSTR	empty string	0-1000 characters
DisplayRelative	Determines whether the time displayed is relative or absolute. When displaying relative time,'time now' is the most current value and is the rightmost value on the x-axis. Note: Ignored for xy-plots.	BOOL	TRUE	FALSE (Absolute time) TRUE (Relative time)
GridColor	Color of the grid. Available at both buildtime and runtime.	OLE_COLOR (long)	R/G/B= 255/255/51 36962303 (YELLOW)	GUS Color Palette

Trend Properties	Definition	Type	Default Value	Valid Values
HairlineColor	Color of the HairlineCursor. Available at both buildtime and runtime.	OLE_COLOR (long)	R/G/B= 255/255/51 36962303 (YELLOW)	GUS Color Palette
HairlineCursor	Determines if the hairline cursor functionality is available (enabled). When the HairlineCursor property is set to TRUE, the user may turn the hairline cursor on and off at runtime with the mouse. If the HairlineCursor property is set to FALSE, the hairline cursor will not be activated at runtime when the user selects the left mouse button. Available at both buildtime and runtime. Note 1: The hairline cursor is not automatically displayed when this property is set. The user can only activate the hairline cursor by using the mouse. Note 2: Ignored for Operation Point charts.	BOOL	TRUE	FALSE (disabled) TRUE (enabled)
HairlineCursorActive	Runtime property (read-only) specifies the active/inactive state of the hairline cursor. It is set to TRUE when the user has turned on the hairline cursor. Note: Ignored for Operation Point charts.	BOOL	FALSE	FALSE (inactive) TRUE (active)
HairlineCursorActive	Runtime property (read-only) specifying the active/inactive state of the hairline cursor. Set to TRUE when the user has activated the hairline cursor.	BOOL	FALSE	FALSE (inactive) TRUE (active)
HairlineReadout	Determines if the Y-values and the time at the hairline cursor are displayed on the Trend Control at runtime. These values are automatically displayed for the Plot Plus Labels trend style when the HairlineCursor property is set to TRUE. For the Plot Area Only trend style, the user must create text objects and scripts to display the properties equivalent to the hairline readout information. Available at both buildtime and runtime.	BOOL	TRUE	FALSE (disabled) TRUE (enabled)

Trend Properties	Definition	Type	Default Value	Valid Values
InheritProperties	Determines whether the object is going to inherit properties from the display's properties. The only property that the Trend Control inherits is background color if'Inherit Properties' is TRUE. Available at both buildtime and runtime.	BOOL	FALSE	FALSE (disabled) TRUE (enabled)
LimitLineColor	Specifies the color of the LimitLines.	OLE COLOR - long	BLACK	GUS Color Pallette
LimitLineLower	Specifies the percent of full-scale lower limit on the y-axis where a horizontal lower limit line is drawn. Note: Line is drawn only if the value is between YScaleLow and YscaleHigh. No line is drawn if the value is negative.	float	-1.0 = no line	-1.0 to 100
LimitLineUpper	Specifies the percent of full-scale upper limit on the y-axis where a horizontal upper Limit line is drawn. Note: Line is drawn only if the value is between YScaleLow and YscaleHigh. No line is drawn if the value is negative.	float	-1.0 = no line	-1.0 to 100
LinearIntercept	Runtime property (read-only) specifies the calculated y intercept for the linear regression equation of a given trace, compared to the current x trace. This is'b' in the equation: y = mx + b. Set to FLT_MAX if no valid calculation is available. Note 1: The OnPropertyChange() event for TREND_CORRELATION is only fired for active traces when this property changes. Note 2: Valid only for xy-plots.	double Parameter: short nTraceID	0.0	-1.0 to 1.0 or FLT_MAX if not valid where FLT_MAX =3.402823466e+38F

Trend Properties	Definition	Type	Default Value	Valid Values
LinearSlope	Runtime property (read-only) specifies the calculated slope for the linear regression equation, of a given trace compared to the current x trace. This is'm' in the equation: y = mx + b. Set to FLT_MAX if no valid calculation is available. Note 1: The OnPropertyChange() event for this property is only fired for active traces. Note 2: Valid only for xy-plots.	double Parameter: short nTraceID	0.0	-1.0 to 1.0 or FLT_MAX if not valid where FLT_MAX =3.402823466e+38F
MarkerSize	Specifies the size of the marking for each data point location.	short Note: default matches existing trend presentation.	0	0 to 9 0 = single dot 1 = small trace specific shape 2 through 9 are multiples applied to the size of the trace specific shapes.
Name	Object name that must be unique in the display. This is the name used in referencing the object in a script. Modified at buildtime, read-only at runtime.	BSTR	Empty string	1 through 24 characters per the following rules: 1. First character must be alpha. 2. Remaining characters must be alpha, numeric or underscore.
NumTraces	Runtime property (read-only) identifying the number of active traces in the trend.	short	0	0 through 8 0 = no active traces
Paused	Runtime property determines whether the chart is dynamically updated as new data is collected. Note 1: The plotted data remains static when TRUE. Note 2: When changed from TRUE to FALSE, the chart will be refreshed to the current time.	BOOL	FALSE	FALSE (Chart will update as new data is received) TRUE (No new values will be plotted)

Trend Properties	Definition	Type	Default Value	Valid Values
RuntimeScrolling	Determines whether runtime scrolling of the trend is enabled or disabled. When the RuntimeScrolling property is set to TRUE, the user can scroll backward and forward through the trace data using the ScrollBackTime and ScrollForwardTime properties. Scrolling capability does not apply to a trace with a data source of Real-Time. The RuntimeScrolling property is available at both buildtime and runtime.	BOOL	TRUE	FALSE (disabled) TRUE (enabled)
ScrollBackTime	Runtime property specifying the number of seconds to scroll back in time. This property is only available for storing when the RuntimeScrolling property is set to TRUE (scrolling enabled). To implement backward scrolling, set this property to the number of seconds to scroll back from the right time limit of the trend. Although any positive value for this property may be entered by the user, the LCN may adjust the amount of scrolling to an appropriate value based on attributes such as the current timebase (e.g. a scroll value of 30 seconds may be adjusted by the LCN to 60 seconds). Therefore, when storing to this property, use the OnPropertyChange() event to confirm the property change has been accepted by the LCN, and to receive the actual scroll amount. Scrolling capability does not apply to a trace with a data source of Real-Time.	long	0 seconds	0 to LONG_MAX, where LONG_MAX = 2147483647 (Note: If the user enters a negative number, its absolute value is taken.)

Trend Properties	Definition	Type	Default Value	Valid Values
ScrollForwardTime	Runtime property specifying the number of seconds to scroll forward in time. This property is only available for storing when the RuntimeScrolling property is set to TRUE (scrolling enabled). To implement forward scrolling, set this property to the number of seconds to scroll forward from the right time limit of the trend. After scrolling backward, to begin trending current data, scroll forward to the current time. Refer to “Trend Scripting Examples” on page 173. A request to scroll forward beyond the current time results in scrolling only to the current time, not into the future. Although any positive value for this property may be entered by the user, the LCN may adjust the amount of scrolling to an appropriate value based on attributes such as the current timebase (e.g. a scroll value of 30 seconds may be adjusted by the LCN to 60 seconds). Therefore, when storing to this property, use the OnPropertyChange() event to confirm the property change has been accepted by the LCN, and to receive the actual scroll amount. Scrolling capability does not apply to a trace with a data source of Real-Time.	long	0 seconds	0 to LONG_MAX, where LONG_MAX = 2147483647 (Note: If the user enters a negative number, its absolute value is taken.)
ShowAxes	Determines whether the axes are displayed on the trend. Available at both buildtime and runtime.	BOOL	TRUE	FALSE (disabled) TRUE (enabled)
ShowGrid	Determines whether a grid is displayed behind the traces. Available at both buildtime and runtime.	BOOL	TRUE	FALSE (disabled) TRUE (enabled)

Trend Properties	Definition	Type	Default Value	Valid Values
TimeBase	Specifies the timebase for the x-axis in seconds. For xy-plots, it determines the amount of history initially collected from the HM. Available at both buildtime and runtime. When storing to this property, use the OnPropertyChange() event to confirm the property change has been accepted by the LCN and to receive the actual property value.	long	1200 seconds (20 minutes)	60 seconds (1 minute) 120 seconds (2 minutes) 300 seconds (5 minutes) 600 seconds (10 minutes) 1200 seconds (20 minutes) 3600 seconds (1 hour) 7200 seconds (2 hours) 28800 seconds (8 hours) 57600 seconds (16 hours) 86400 seconds (24 hours) 172800 seconds (48 hours) 345600 seconds (96 hours) (Note: Adjusted to the nearest value if none of the above is entered.)
TimeBaseIndex	Specifies the timebase for the x-axis as a number from 1 through 12 where it corresponds to one of the twelve TimeBase selections. Available at both buildtime and runtime. When storing to this property, use the OnPropertyChange() event to confirm the property change has been accepted by the LCN and to receive the actual property value.	short	5 (20 minutes)	1 12 (for example, the number'5' sets the timebase to 1200 seconds).
TimeLimitLeft	Runtime property (read-only) which returns the time of collection of the oldest trace value displayed on the trend. In other words, the leftmost limit on the x-axis.	DATE	1/1/1970	1/1/1970 through current time
TimeLimitRight	Runtime property (read-only) which returns the time of collection of the newest trace value displayed on the trend. In other words, the rightmost limit on the x-axis.	DATE	1/1/1970	1/1/1970 through current time
TraceColor	Color of the trace. Available at both buildtime and runtime.	OLE_COLOR (long) Parameter: short nTraceID	R/G/B= 255/0/0 33554687 (RED)	GUS Color Palette

Trend Properties	Definition	Type	Default Value	Valid Values
TraceVisible	Runtime property specifies whether the values for a specific trace are plotted. Note: Changing this property does not affect data collection.	BOOL Parameter: short nTraceID	TRUE	FALSE (not plotted) TRUE (plotted)
TraceLine	Specifies how connecting lines are drawn on xy-plots. Note 1: The HairlineCursor is available only if TraceLine is 2 (Linear Regression). Note 2: Ignored for Trends and Operation Point charts.	short	0	0 = no line 1 = point to point 2 = Linear Regression
UserErrorHandling	Specifies how errors are reported.	BOOL	FALSE	FALSE (Errors are reported in a dialog box) TRUE (Errors are reported through an OnError event)
VariableID	Point name (i.e., Variable ID) for a given trace. Read-only property at runtime.	BSTR Parameter: short nTraceID	' .'	3 through 28 characters formatted per LCN Variable ID Syntax as follows: 1. Optional LCN ID of 2 alpha-numerics plus backslash separator. 2. Required 16-char max Point Name plus dot separator; 3. Required 8-char max Parameter Name. Each char in Point/ Parameter Name must be alpha, numeric, ! , $, or underscore. Names cannot begin or end in underscore, and cannot have two sequential underscores. Dot is not allowed. (Note 1: Custom Parameter Names can be internationalized. Therefore, only ASCII-based Parameter Names are syntax-checked in GUS Release 100.) (Note 2: Can only be updated in buildtime mode)
Visible	Determines the visibility of the object at buildtime and at display invocation. Available at both buildtime and runtime.	BOOL	TRUE	FALSE (disabled) TRUE (enabled)

Trend Properties	Definition	Type	Default Value	Valid Values
XCursorTime	Runtime property (read-only) specifying the time, as a data type string, referred to by the position of the hairline cursor. Only available when the HairlineCursor property is set to TRUE (hairline enabled). Note: Valid only for trends.	BSTR	Empty string	
XCursorTimeDate	Runtime property (read-only) specifying the time, as a data type DATE, referred to by the position of the hairline cursor. Only available when the HairlineCursor property is set to TRUE (hairline enabled). Note: Valid only for trends.	DATE	1/1/1970	1/1/1970 through current time
XnumMinorGridLines	Determines the number of ‘minor' grid lines to be displayed between the tic marks on the x-axis. Note: Ignored by trends.	short	1	0 through 20
XnumTicMarks	Determines the number of x-axis tic marks to be displayed evenly across the x scale range. Note: Ignored by trends.	short	5	1 through 20
XscaleHigh	Percent of full-scale high limit on the x-axis. Note: Ignored by trends.	float	100.0	0.01 through 100.0 (Note: XScaleHigh must be > XScaleLow)
XscaleLow	Percent of full-scale low limit on the x-axis. Note: Ignored by trends.	float	0.0	0.0 through 99.99 (Note: XScaleHigh must be > XScaleLow)
Xtrace	Runtime property specifies which variable is used for the x-axis. Note: Ignored by trends.	short	1	1 to 8 Must match a configured TraceID

Trend Properties	Definition	Type	Default Value	Valid Values
YCursorReadout	Runtime property (read-only) specifying the extrapolated value, as a data type long, of a given trace referred to by the position of the hairline cursor. Only available when the HairlineCursor property is set to TRUE (hairline enabled). This value is only valid when the corresponding YcursorReadoutStatus = TREND_STATUS_OK. When the corresponding YCursorReadoutStatus is not OK, the YCursorReadout value is set to LONG_MAX. Note: The OnPropertyChange() event for this property is only fired for active traces. This property is available upon request for both active and inactive traces. Note: On xy-plots, this is available only when TraceLine is set to Linear Regression, and the value is interpolated based on the Linear Regression equation. For OperationPoint charts, the plotted value of the trace is reported regardless of the HairlineCursor setting.	long Parameter: short nTraceID	LONG_MAX	(-1)*LONG_MAX to LONG_MAX, where LONG_MAX = 2147483647 Note: The YCursorReadout returns extrapolated values. Thus, the hairline cursor values displayed on the Trend Control can vary slightly from the same point.parameter if there is a difference in the range of the two traces. This variation occurs because each trace is collected from the data owner separately, even if the two traces have the same point.parameter. In this case, if the parameter changes rapidly, a slight difference in the value can occur when it is collected independently.

Trend Properties	Definition	Type	Default Value	Valid Values
YCursorReadoutReal	Runtime property (read-only) specifying the extrapolated value, as a data type Float, of a given trace referred to by the position of the hairline cursor. Only available when the HairlineCursor property is set to TRUE (hairline enabled). This value is only valid when the corresponding YcursorReadoutStatus = TREND_STATUS_OK. When the corresponding YCursorReadoutStatus is not OK, the YCursorReadout value is set to FLT_MAX. Note: The OnPropertyChange() event for this property is only fired for active traces. This property is available upon request for both active and inactive traces. Note: On xy-plots, this is available only when TraceLine is set to Linear Regression, and the value is interpolated based on the Linear Regression equation. For OperationPoint charts, the plotted value of the trace is reported regardless of the HairlineCursor setting).	float Parameter: short nTraceID	FLT_MAX	(-1)*FLT_MAX to FLT_MAX, where FLT_MAX = 3.402823466e+38F Note: The YCursorReadoutReal returns extrapolated values. Thus, the hairline cursor values displayed on the Trend Control can vary slightly from the same point.parameter if there is a difference in the range of the two traces. This variation occurs because each trace is collected from the data owner separately, even if the two traces have the same point.parameter. In this case, if the parameter changes rapidly, a slight difference in the value can occur when it is collected independently.
YCursorReadoutStatus	Runtime property specifying the status of the corresponding trace value of a given trace referred to by the position of the hairline cursor. Only available when the HairlineCursor property is set to TRUE (hairline enabled). Note: There is no OnPropertyChange() event for the YCursorReadoutStatus property. The value of the YCursorReadoutStatus property automatically changes when the value of the YCursorReadout property changes, yet there is no event indicating the status has changed. This property is available upon request for both active and inactive traces.	long Parameter: short nTraceID	3=TREND_ STATUS_NA	0 = TREND_STATUS_OK (A good value was received from the LCN for this timestamp) 1 = TREND_STATUS_ERR (No data has been received from the LCN for this timestamp) 2 = TREND_STATUS_BAD (A bad status has been received from the LCN for this timestamp) 3 = TREND_STATUS_NA (Trace not active)

Trend Properties	Definition	Type	Default Value	Valid Values
YdisplayTicPercents	Determines how the Y-axis (vertical) tic percentages are to be displayed. Available at both build time and runtime. This property is applicable only when the Plot Plus Labels trend style is selected.	short	0	0 = None (do not display percentages). 1 = Both Sides (display on both vertical axes). 2 = Left Side (display on left axis). 3 = Right Side (display on right axis).
YNumMinorGridLines	Determines the number of'minor' grid lines to be displayed between the tic marks. Available at buildtime and runtime.	short	1	0 through 20
YNumTicMarks	Determines the number of Y-axis tic marks to be displayed evenly across the Y scale range. Available at buildtime and runtime.	short	5	1 through 20
YRangeHigh	Engineering units high range on the y-axis for a given trace. Available at both buildtime and runtime.	float Parameter: short nTraceID	100.0	(-1)*FLT_MAX to FLT_MAX, where FLT_MAX = 3.402823466e+38F (Note: In Engineering Units, where YRangeHigh must be > YRangeLow)
YRangeLow	Engineering units low range on the y-axis for a given trace. Available at both buildtime and runtime.	float Parameter: short nTraceID	0.0	(-1)*FLT_MAX to FLT_MAX, where FLT_MAX = 3.402823466e+38F (Note: In Engineering Units, where YRangeHigh must be > YRangeLow)
YScaleHigh	Percent of full-scale high limit on the y-axis. Available at both buildtime and runtime.	float	100.0	0.01 through 100.0 (Note: YScaleHigh must be > YScaleLow)
YScaleLow	Percent of full-scale low limit on the y-axis. Available at both buildtime and runtime.	float	0.0	0.0 through 99.99 (Note: YScaleHigh must be > YScaleLow)

5.8.9 Trend Methods

Trend Methods	Definition	Parameters	Return
AddTrace	Method used in scripting to add a trace to a trend at runtime. The return parameter is the new trace number added. (-1) is returned for failure.	LPCTSTR sVariableID, OLE_COLOR (long)ClrTraceColor SvariableID is the name of the point.parameter (without the lcn. prefix) to be plotted. It must be a numeric, flag or enumeration. Flags are plotted with a value of 0 or 100. Enumerations are scaled based on the highest member number; thus an enumeration set with 4 members will have possible plotted values of 0, 33.3, 66.7, & 100.	short nTraceID
DeleteTrace	Method used in scripting to delete a trace from a trend at runtime. The return parameter is the trace number deleted. (-1) is returned for failure.	short nTraceID 0 = Delete all traces Note: If the x-axis trace is deleted, Xtrace is set to 0 and no data will plot for xy-plots or Operations Point chart.	short nTraceID
GetDataSourceString	Method used to retrieve a string representing the data source for a given trace. Returns are'RT' (Real-time),'HM' (History Module),'HG' (Hiway Gateway), or'AU' (Automatic).	short nTraceID	BSTR sDataSource
Refresh	Method used to clear the buffered data and collect a fresh set of values. The amount of data collected depends on the TimeBase property.	DATE endtime If a future endtime or a value of 0 is specified the current time is used.	BOOL indicating success or failure in retrieving the data

5.8.10 Trend Events

Trend Events	Definitions	Parameters
Click	Executes on both Left and Right button clicks. Left button click inside the plot area will activate hairline cursor (if enabled), followed by the actions scripted for the Click event. Left button click outside the plot area, for the Plot Plus Labels trend style, will deactivate the hairline cursor (if enabled), followed by actions scripted for Click event.	none
DblClick	Executes on both Left and Right button clicks. User will get Click event first followed by DblClick event.	none
OnDataChange		none
OnDisplayShutdown		none
OnDisplayStartup		none

Trend Events	Definitions	Parameters
OnHelp		none
OnPeriodicUpdate		none
OnPropertyChange	An event automatically executed when the value of a Trend Control property changes. The properties which activate events are selected at buildtime. Refer to “OnPropertyChange event” on page 173.	Integer nProperty, Integer nTraceNumber, Variant vValue
OnRButtonClick	Event to deactivate hairline cursor, if hairline cursor is enabled. Right button click any area of the Trend Control, for all trend styles, will deactivate the hairline cursor (if enabled), followed by the user's scripted actions for the OnRButtonClick event.	none

Tip

There are scenarios when a user click event executes more than one click script or a click script executes regardless of which mouse button is clicked. These scenarios are:

- When the left or right mouse button is clicked, Click and DblClick scripts are executed;
- When the right mouse button is clicked, the OnRButtonClick script is executed, followed by the Click script;
- When the left mouse button is double-clicked, the Click script is executed, followed by the DblClick script;
- When the right mouse button is double-clicked, the OnRButtonClick script is executed, followed by the Click script, and finally the DblClick script.

To prevent multiple scripts from executing when the mouse buttons are clicked, make sure to only script the Click event; do not script the OnRButtonClick or DblClick events.

5.9 Trend OnPropertyChange Event

Related topics

“Trend OnPropertyChange Description” on page 172
“Programmatic Interfaces” on page 172

5.9.1 Trend OnPropertyChange Description

The Trend Control event, OnPropertyChange(), notifies the user when the value of a property changes. The Plot Area Only trend style allows the user to present ancillary information such as axes labels and trace legend in their own fashion, via display objects and scripting. Therefore, the OnPropertyChange() event is needed by scripting to display accurate values for the ancillary information.

OnPropertyChange() is an event that is automatically executed on the Trend Control at runtime when any selected property of the trend or any selected property of a trace changes. The properties are selected on the Change Notification property page at buildtime. Property value changes can happen automatically, such as when the value changes at the hairline cursor. The change can also be user-instigated such as using the mouse to display the hairline cursor.

CAUTION
The user is advised to only select events that will be used through scripting at runtime. Failure to do this will result in unnecessary events and reduced performance of the entire display.

A change of a single property may cause multiple events of related properties, such as:

When the hairline cursor is moved several properties change, including the XCursorTime, and YCursorReadout for all traces.

When the hairline cursor is fixed and the values under the hairline cursor change due to new incoming trend values, several properties change, including the TimeLimitLeft, TimeLimitRight, XCursorTime, and YCursorReadout for all traces.

When the timebase is changed several properties change, including the TimeLimitLeft and TimeLimitRight. If the hairline cursor is active, the properties XCursorTime, and YCursorReadout for all traces change.

5.9.2 Programmatic Interfaces

The BasicScript syntax for this event is:

OnPropertyChange(nProperty As Integer, nTraceNumber As Integer, vValue As Variant) where:

- **nProperty** is an integer value which is an index indicating the specific property whose value has changed. Refer to “Trend Property Constants” on page 110 for additional information.
- **nTraceNumber** is an integer which indicates the trace number, if applicable. If the property is a trend property, and is not associated with a trace, then this value is 0;
- **vValue** is a variant which contains the new value of the property that changed at the time the event was sent. (Note: The user can alternatively reference the Trend Control explicitly to get the current value of the property.) The data type of vValue will be the same data type as the property.

5.10 Trend Scripting Examples

These examples contain scripts for the following Trend Control properties and events:

- OnPropertyChange
- AxesColor
- HairlineCursorActive
- TimeLimitRight and TimeLimitLeft
- YCursorReadoutStatus with YCursorReadout
- ScrollBackTime and ScrollForwardTime

5.10.1 OnPropertyChange event

The following is an example of the scripted OnPropertyChange() event on the Trend Control, using the index values for the Property.

Assume the display contains the following objects:

- A Trend Control object, Trend1, containing two traces.
- Two text objects, Text1 and Text2, which will display the value under the hairline cursor for each trace. It is not necessary for these two text objects to have an OnDataChange() script or VTO to display the value. The value is instead assigned in the OnPropertyChange script.

The Trend1 object has the following script:

```
Sub OnPropertyChange(nProperty As Integer, nTraceNumber As Integer, vValue As Variant)
Select Case nProperty
Case TREND_YCURSORREADOUT
If nTraceNumber = 1 then
‘ Get the value from the event (vValue)
Text1.text = vValue
elseif nTraceNumber = 2 then
‘ Can also get the value from the property:
Text2.text = Trend1.YCursorReadout(nTraceNumber)
End if
End Select
End Sub
```

AxesColor property

In BasicScript the user could use the following statement on a rectangle or text object to set the color of the axes of the Trend1 object to TDC_RED at runtime:

```
Sub OnLButtonClick()
Trend1.AxesColor = TDC_RED
End Sub
```

The user could also use the MakeColor function to chose a color:

```
Sub OnLButtonClick()
Trend1.AxesColor = MakeColor(1, 125, 255)
End Sub
```

HairlineCursorActive property

The following is an example of using the HairlineCursorActive property to determine if the hairline cursor time should be displayed through a text object.

Assume the display contains the following objects:

- A Trend Control object, Trend1, containing one trace.
- One text object, Text1, which will display the time under the hairline cursor for the trace only if the user is using the hairline cursor. It is not necessary for this text object to have an OnDataChange() script or VTO to display the value. The value is instead assigned in the OnPropertyChange script.

The Trend1 object has the following script:

```
Sub OnPropertyChange(nProperty As Integer, nTraceNumber As Integer, vValue As Variant)
Select Case nProperty
‘ The time under the hairline cursor has changed
Case TREND_XCURSORTIME
If Trend1.HairlineCursorActive = TRUE then
‘ Get the value from the event (vValue):
Text1.text = vValue
else
‘ Null out the text object:
Text1.text =''
End if
End Select
End Sub
```

TimeLimitRight and TimeLimitLeft properties

The following is an example of using the TimeLimitRight and TimeLimitLeft properties to display the timebase limits through two text objects.

Assume the display contains the following objects:

- A Trend Control object, Trend1, containing a trace, and is of the'plot area only' style.
- Two text objects, Text1 and Text2, which will display the timebase limits for the trend. It is not necessary for these text objects to have an OnDataChange() script or VTO to display the values. The values are instead assigned in the OnPropertyChange script.

The Trend1 object has the following script:

```
Sub OnPropertyChange(nProperty As Integer, nTraceNumber As Integer, vValue As Variant)
If nProperty = TREND_TIMELIMITLEFT then
‘ Get the value from the event (vValue):
Text1.Text = vValue
Else If nProperty = TREND_TIMELIMITRIGHT then
‘ Can also get the value from the property:
Text2.Text = Trend1.TimeLimitRight
End if
End Sub
```

YCursorReadoutStatus with YCursorReadout property

The following is an example of using the YCursorReadoutStatus in coordination with the YCursorReadout property to display the trace values through a text object.

Assume the display contains the following objects:

- A Trend Control object, Trend1, containing one trace, and is of the'plot area only' style.
- One text object, Text1, which will display the trace value or a bad status for the hairline cursor. It is not necessary for this text object to have an OnDataChange() script or VTO to display the value. The value is instead assigned in the OnPropertyChange script.

The Trend1 object has the following script:

```
Sub OnPropertyChange(nProperty As Integer, nTraceNumber As Integer, vValue As Variant)
If nProperty = TREND_YCURSORREADOUT then
If nTraceNumber = 1 then
If Trend1.YCursorReadoutStatus(nTraceNumber) = TREND_STATUS_OK then
‘ Get the good value from the event (vValue):
Text1.Text = vValue
elseif Trend1.YCursorReadoutStatus(nTraceNumber) = TREND_STATUS_ERR then
Text1.Text ='ERR'
elseif Trend1.YCursorReadoutStatus(nTraceNumber) = TREND_STATUS_BAD then
Text1.Text ='BAD'
elseif Trend1.YCursorReadoutStatus(nTraceNumber) = TREND_STATUS_NA then
Text1.Text ='N/A'
End if ‘ Status
End if ‘TraceNumber
End if ‘Property
End Sub
```

ScrollBackTime and ScrollForwardTime properties

The following is an example of using the ScrollBackTime and ScrollForwardTime properties in a variety of methods to allow the user to scroll back or scroll forward to view historized data.

Assume the display contains the following objects:

- A Trend Control object, Trend1, containing one trace whose data source is the History Module. The RuntimeScrolling property is enabled. In the Display builder, the following properties are set (on the Change Notification tab on the Properties page) to notify the user when they have changed: ScrollBackTime and ScrollForwardTime.
- One text object, Text1, which when left-button-clicked will scroll back in time, one time base. For example, if the time base is set to 20 minutes, when the Text1 object is left-mouse clicked, data from the previous 20 minutes will be displayed.
- One text object, Text2, which when left-button-clicked will prompt the user for the number of seconds to scroll forward in time and then scroll forward by the specified amount of time. If the specified amount of time results in a future time, forward scrolling will stop at the current time.
- One text object, Text3, which when left-button-clicked will move the user to the current time and start displaying current historized data.

The Trend1 object has the following script:

```
Sub OnPropertyChange(nProperty As Integer,nTraceNumber As Integer,vValue As Variant)
if nProperty = TREND_SCROLLFORWARDTIME then
msgbox'The new ScrollForwardTime is ' & vValue
elseif nProperty = TREND_SCROLLBACKTIME then
msgbox'The new ScrollBackwardTime is ' & vValue
end if
End Sub
```

The Text1 object has the following script:

```
Sub OnLButtonClick()
Trend1.ScrollBackTime = Trend1.TimeBase
End Sub
```

The Text2 object has the following script:

```
Sub OnLButtonClick()
dim u_input as string
dim request as long
‘request from the user the number of seconds to scroll forward
u_input = inputbox('Please enter the number of seconds you wish to scroll forward','Scroll Forward
Some Number of Seconds')
‘if the contents of the edit box of the input box are empty, then the user either entered nothing
in
‘the edit box or pressed the Cancel button on the input box. Therefore, exit this LButtonClick
event
if u_input ='' then
Exit Sub
end if
‘if the contents of the inputbox control are NOT numeric, then display a message box stating that
all
‘entries should be numeric and exit this LButtonClick event
if IsNumeric(u_input) = FALSE then
msgbox'Input requires a valid numeric entry - - Please try again'
Exit Sub
end if
‘must convert the data from the input box from a string to a long, the datatype for the
‘ScrollForwardTime property
request = CLng(u_input)
dim forward_date as date
‘calculate the new date based on the number of seconds requested by the user to scroll
‘forward
forward_date = DateAdd('s', request, Trend1.TimeLimitRight)
‘test to see if the new date is in the future beyond the current time. If so, display a
‘message box telling the user that the ScrollForwardTime Property will scroll forward
‘only to the current time.
if (forward_date > now()) then
msgbox'You have requested to move into the future - therefore we will only ScrollForward to the
current time.'
```

```
end if
‘Move forward to a time that is less than or equal to the current time. If the requested
‘forward time is some time in the future, then the forward scroll will only scroll to the
‘current time.
Trend1.ScrollForwardTime = request
End Sub
```

The Text3 object has the following script:

```
Sub OnLButtonClick()
dim toNow as double
‘calculate the number of seconds between the right time limit on the trend and the current
‘time
toNow = DateDiff('s',Trend1.TimeLimitRight,now() )
‘scroll forward in time, the number of seconds calculated in the previous statement.
‘This will set the TimeLimitRight property to the current time - thus the current data
‘will start being displayed
Trend1.ScrollForwardTime = toNow
End Sub
```

5.11 Scripting an Example of an Embedded Trend

This example is intended as an aid for Display authors.

There are three reasons why the embedded trend displays named **MainPic_EmbedTrendExample.pct** and **SubPic_EmbedTrendExample.pct** were developed.

1. To display how, at build time to specify point.parameter expressions for the YRangeHigh and YRangeLow properties for a trace rather than specifying a numerical value. This topic is discussed in Section 2 - How to Use a Point.Parameter Expression for the YRangeHigh and YRangeLow properties.
2. To demonstrate the new plot style called ‘Plot Area Only' and use of the new event OnPropertyChange( ).
3. To provide scripts to demonstrate how to add a trace, delete a trace and how to change the range of values being plotted at runtime, as well as how to populate the trend with trace data at display startup. These topics are discussed in section 6.2 - Components of the Embedded Display, SubPic_EmbedTrendExample.pct.

5.11.1 How to Use a Point.Parameter Expression for YRangeHigh and YRangeLow Properties

Currently, at buildtime, it is not possible to enter a point.parameter expression for the value of YRangeHigh or YRangeLow. Only numerical values can be entered for these two properties when a trend is being built.

However, by placing the trend in an embedded display and defining a parameter for the YRangeHigh and YRangeLow properties, the user can enter a point.parameter expression for each of these properties when the embedded display is inserted into a main display. Then, at runtime the value of the specified point.parameters are used as the values of the YRangeHigh and YRangeLow properties.

When creating the embedded display, the author should, for each trace, define a parameter for the following:

- The point.parameter whose value is being plotted
- The YRangeHigh property
- The YRangeLow property

For example, in the example embedded display, SubPic_EmbedTrendExample.pct the following parameters are defined for trace number 1:

- TraceName1 - represents the point.parameter being plotted in trace 1
- TraceHigh1 - represents the value of the YRangeHigh property of trace 1
- TraceLow1 - represents the value of the YRangeLow property of trace 1

The following table is a complete list of the parameters defined in the embedded display, SubPic_EmbedTrendExample.pct.

Parameter	Represents
TraceHigh1	Value of the YRangeHigh property of trace 1
TraceHigh2	Value of the YRangeHigh property of trace 2
TraceHigh3	Value of the YRangeHigh property of trace 3
TraceHigh4	Value of the YRangeHigh property of trace 4
TraceHigh5	Value of the YRangeHigh property of trace 5
TraceHigh6	Value of the YRangeHigh property of trace 6
TraceHigh7	Value of the YRangeHigh property of trace 7
TraceHigh8	Value of the YRangeHigh property of trace 8
TraceLow1	Value of the YRangeLow property of trace 1
TraceLow2	Value of the YRangeLow property of trace 2
TraceLow3	Value of the YRangeLow property of trace 3

Parameter	Represents
TraceLow4	Value of the YRangeLow property of trace 4
TraceLow5	Value of the YRangeLow property of trace 5
TraceLow6	Value of the YRangeLow property of trace 6
TraceLow7	Value of the YRangeLow property of trace 7
TraceLow8	Value of the YRangeLow property of trace 8
TraceName1	Point.parameter whose value is being plotted in trace 1
TraceName2	Point.parameter whose value is being plotted in trace 2
TraceName3	Point.parameter whose value is being plotted in trace 3
TraceName4	Point.parameter whose value is being plotted in trace 4
TraceName5	Point.parameter whose value is being plotted in trace 5
TraceName6	Point.parameter whose value is being plotted in trace 6
TraceName7	Point.parameter whose value is being plotted in trace 7
TraceName8	Point.parameter whose value is being plotted in trace 8

5.11.2 'Plot Area Only' and OnPropertyChange( ) Event

The user can select one of two trend styles at buildtime -'Plot Plus Labels' or'Plot Area Only'. The'Plot Plus Labels' plot style displays all the ancillary information (such as trace color, hairline cursor value or time of the last collected value) associated with a trend. This data is fixed and display authors cannot modify or delete any of the data that is displayed.

The'Plot Area Only' trend style and the OnPropertyChange( ) event were developed to provide display authors and users more flexibility with respect to displaying trend and trace data. This trend style displays only the plot area of the trend control. None of the ancillary information is displayed. This has two advantages:

- 'Plot Area Only' trend style takes up less real estate in a display.
- Authors can display, via scripting, only the data that is relevant for their needs

In the embedded display, SubPic_EmbedTrendExample.pct, the ‘Plot Area Only' trend style is used and, for example purposes, nearly all the ancillary information associated with the'Plot Plus Labels' trend style is displayed along with some additional information. Of course, authors can display part, all or completely different data - whatever is appropriate for their needs.

The following is a chart showing the data displayed in the embedded display, **SubPic_EmbedTrendExample.pct** and whether that data is displayed if the'Plot Plus Labels' trend style is selected.

Piece of Data	**Displayed in'Plot Plus Labels' trend style?**
Point.parameter being plotted for each trace	No
YRangeHigh value for each trace	No
YRangeLow value for each trace	No
Most recent time for the selected time base	Yes
Oldest time for the selected time base	Yes
Value at the hairline cursor, when the hairline cursor is active	Yes
Time at the hairline cursor, when the hairline cursor is active	Yes

How was it possible to display this data at runtime? For some of this data, it was very straightforward. For example, to display the color of a trace in a legend it was only necessary to fill a rectangle object with the color

of the trace and make that rectangle object visible when the trace was added to the trend. The following is the code segment that appears in the'Add A Trace' target for displaying the color of the trace:

- ‘fill rectangle named TraceColor1 with color of Trace 1
- Trace1Color.fillcolor = OleObject1.TraceColor(nTraceID(1))
- ‘make Trace1Color rectangle visible
- Trace1Color.visible = TRUE

However, some of the data that was displayed, is internal to the trend control and changes very often. This is where the OnPropertyChange( ) event proves to be so useful. The OnPropertyChange( ) event is automatically executed at runtime when any selected property of the trend or any selected property of a trace changes. These properties are selected on the Change Notification property page at buildtime. Refer to “OnPropertyChange event” on page 173 for more information.

In the embedded display, SubPic_EmbedTrendExample.pct, the following properties were selected to notify the user when they were changed:

- HairlineCursorActive
- TimeLimitLeft
- TimeLimitRight
- XCursorTime
- YCursorReadout

In the OnPropertyChange( ) event, there is a case statement that handles each of these properties when its value changes. Of course, display authors can modify this case statement to handle more or less properties. For example, when the hairline cursor is made visible, the HairlineCursorActive property changes to TRUE causing the values at the hairline cursor to be displayed along with the time at the hairline cursor.

NOTE: if any additional properties are added to the case statement, be sure to add the new properties to the list on the Change Notification property page.

The YCursorReadout property in the case statement is of special interest. This property is the value at the hairline cursor of a specified trace. Because this value can be bad or no data could be available for that moment in time, it is recommended that the value of the YCursorReadoutStatus be checked before displaying the data. Refer to “Trend Status Constants” on page 112 for a list of the valid states and what each state means.

The following is a list of the objects, which display the values of the properties of the trend or the traces in the trend. These properties are made visible and invisible when one of the above properties changes or certain events is executed:

5.11.3 Objects to Display Color of each Trace

made visible when a trace is successfully added

made invisible when a trace is successfully deleted

- Trace1Color - rectangle object
- Trace2Color - rectangle object
- Trace3Color - rectangle object
- Trace4Color - rectangle object
- Trace5Color - rectangle object
- Trace6Color - rectangle object
- Trace7Color - rectangle object
- Trace8Color - rectangle object

5.11.4 Objects to Display Name of Point.Parameter being Plotted for each Trace

made visible when a trace is successfully added

made invisible when a trace is successfully deleted

- VariableID1 - text object
- VariableID2 - text object
- VariableID3 - text object
- VariableID4 - text object
- VariableID5 - text object
- VariableID6 - text object
- VariableID7 - text object
- VariableID8 - text object

5.11.5 Objects to Display YRangeHigh Values for each Trace

made visible when a trace is successfully added

made invisible when a trace is successfully deleted

- High1 - text object
- High2 - text object
- High3 - text object
- High4 - text object
- High5 - text object
- High6 - text object
- High7 - text object
- High8 - text object

5.11.6 Objects to Display YRangeLow Values for each Trace

made visible when a trace is successfully added

made invisible when a trace is successfully deleted

- Low1 - text object
- Low2 - text object
- Low3 - text object
- Low4 - text object
- Low5 - text object
- Low6 - text object
- Low7 - text object
- Low8 - text object

5.11.7 Objects to Display Values at Hairline Cursor for each Trace

made visible when the HairlineCursorActive property is set to true and the trace is active

made invisible when a trace is successfully deleted or when the HairlineCursorActive property is set to false (i.e. the hairline cursor is no longer displayed) or when there are no traces in the trend

- Trace1HCVal - text object
- Trace2HCVal - text object
- Trace3HCVal - text object
- Trace4HCVal - text object

- Trace5HCVal - text object
- Trace6HCVal - text object
- Trace7HCVal - text object
- Trace8HCVal - text object

5.11.8 Object to Display Time Value at the Hairline Cursor

made visible when the HairlineCursorActive property is set to true

made invisible when the HairlineCursorActive property is set to false or there are no traces in the trend

- XCursorTime - text object

5.11.9 Object to Display Time Value at the Right Side of the Trend

made visible and updates each time the TimeLimitRight property changes

- TimeLimitRight - text object

5.11.10 Object to Display Time Value at the Left Side of the Trend

made visible and updates each time the TimeLimitLeft property changes

- TimeLimitLeft - text object

5.11.11 Components of the Embedded Display, SubPic_EmbedTrendExample.pct

'Add A Trace' Target—This target is used to add a trace to the trend in the display when the user left-button clicks the target.

This target is used to add a trace to the trend in the display when the user left-button clicks the target.

The important points of this subroutine are:

1. the parameter for the name of the trace is set to the name of the point.parameter entered by the user. For example:

   display.params.TraceName1 = strTraceName

   where

   strTraceName ='tph_gd00.pv'
2. then, the AddTrace( ) event is called using the trace name parameter as the first argument. For example:

nTraceID(1) = AddTrace(**display.params.TraceName1**, nTraceColor(1))

1. when the AddTrace( ) event is called, the values for the YRangeHigh and YRangeLow properties are set, by default, to 100 and 0, respectively. If the user wishes to change these values, he/she should select the'Change Trace Range' target after adding the new trace
2. If the new trace was added successfully, the parameters for the YRangeHigh and YRangeLow properties must now be updated to the default values, which were set when the AddTrace( ) method was invoked.

display.params.TraceHigh1 = OleObject1.YRangeHigh(nTraceID(1))

display.params.TraceLow1 = OleObject1.YRangeLow(nTraceID(1))

'Delete A Trace' Target—This target is used to delete a trace from the trend in the display when the user left-button clicks the target.

The important points of this subroutine are:

1. The user can delete all traces from the trend at the same time by entering the number 0 when asked which trace to delete.

2. After testing to see if the specified trace is active (i.e. plotting data for a point.parameter), the DeleteTrace( ) method is invoked with the trace number as the argument. This deletes the specified trace and returns the number of the trace that was deleted. If all traces were deleted, the DeleteTrace( ) method returns 0.
3. If the user-specified trace was successfully deleted, it must be made inactive. For example, if the first trace was successfully deleted, the following statement would make trace 1 inactive.

nTraceID(1) = 0

'Change Trace Range' Target—This target is used to change the range of values to be plotted for a specified trace. This is accomplished by changing the values of the YRangeHigh and YRangeLow properties of the user-specified trace.

The important points of this subroutine are:

1. This example assumes that the user would like to enter the name of a point.parameter whose value is used as the value of the new YRangeHigh or YRangeLow property. Users of the embedded display, SubPic_EmbedTrendDisplay.pct can easily modify this subroutine to request absolute values from the user.
2. The user enters a string when asked for the name of the point.parameter, which corresponds, to the new YRangeHigh or YRangeLow value. This string must be converted to an object, so that the value of the point.parameter can be used. This is accomplished with the following statements:

dim objTraceHigh as object

…

set objTraceHigh = GetVar(strTraceHigh)

where strTraceHigh is the point.parameter string expression entered by the user

1. The YRangeHigh and YRangeLow properties must be updated to the value of the point.parameter specified by the user. For example:

   OleObject1.YRangeHigh(nTraceID(4)) = objTraceHigh

   OleObject1.YRangeLow(nTraceID(4)) = objTraceLow
2. The embedded display parameters which represent the YRangeHigh or YRangeLow properties must be updated to the value of the point.parameter specified by the user. For example:

display.params.TraceHigh4 = objTraceHigh

display.params.TraceLow4 = objTraceLow

OnDisplayStartup( ) Subroutine for the Trend Object—If, in the main display, values were entered for the trace name parameters, the OnDisplayStartup( ) routine populates the trend control with as many as 8 traces when the display starts running.

This routine is very similar to the'Add A Trace' routine with the exception that the YRangeHigh and YRangeLow values for each trace are specified as parameters. Therefore, the YRangeHigh and YRangeLow properties of each trace are set to the expressions entered when the embedded display was inserted into the main display.

5.11.12 Notes, Warnings, Cautions

This example assumes that all traces are added at runtime. To ensure that this example works correctly, please do not configure, at build time, any traces in the OleObject1 trend control which is located in SubPic_EmbedTrendExample.pct.

Before Starting—All users of the embedded display SubPic_EmbedTrendExample.pct must make the following change to the OnLButtonClick( ) script for the ‘Change Trace Range' target.

Change the point.parameter in the line:

set dispdb.var01 = GetVar('tph_gd00.pv')

to a valid point.parameter on your system.

Save the changes and re-embed the display in your own main display or the main display that Honeywell has provided - MainPic_EmbedTrendExample.pct.

Before Starting—If you intend to use the main display provided by Honeywell, all expressions entered for the embedded display parameters must be changed to expressions that are valid on your system. This can be accomplished by doing the following

- Select the embedded display in the main display MainPic_EmbedTrendExample.pct
- Select the menu item'Enter Parameters…' in the Edit menu
- Change each expression for each parameter to a valid expression on your system

If you intend to make any modifications to the scripts in the SubPic_EmbedTrendExample.pct display, the following information provides an explanation of the nTraceID variable that is used throughout the display.

In order to provide the user with information about the status of a trace in a trend control, the variable nTraceID is used throughout the scripts in the embedded display *SubPic_EmbedTrendExample.pct.* nTraceID is an array of size 8 and of type integer. So, nTraceID(1) is the first trace, nTraceID(2) is the second trace, and so on. The value of each element of nTraceID indicates whether or not the trace is active. If the value of any element is greater than 0, then the trace is active (i.e. the trace is successfully plotting data). If the value of any element of nTraceID is <= 0, the trace is inactive (i.e. the trace is not plotting any data)

Each element of nTraceID is set when a call is made to the AddTrace( ) method or a call is made to the DeleteTrace( ) method. When the AddTrace( ) method is called, the corresponding element of nTraceID is set with the return value of the AddTrace( ) call. If the AddTrace( ) call was successful, the return value is the number of the trace that was added. If the AddTrace( ) call was unsuccessful, the return value is -1. When the DeleteTrace( ) method is called, the corresponding element is set to 0.

By checking the value of the appropriate element in the nTraceID array, information can be presented to the user about a particular action, which might not otherwise be available. For example, without the use of the nTraceID array, if the user attempts to delete a trace that is not active, there is no feedback that this activity cannot occur.

5.12 Using the xPM Logic Display Control

Related topics

5.12.1 Support for xPM Logic Display control

With GUS R360, GUS Display Builder supports xPM Logic display control for xPM logic points. The control can be embedded in GUS displays for viewing the graphical representations of xPM logic points. The control is also supported in Experion Station-TPS (ES-T) nodes.

5.12.2 xPM Logic display control availability

The xPM Logic display control is automatically installed when the GUS Display Builder or the GUS Display Runtime is installed. The control is available in the GUS Display Builder - Insert > Control dialog box as Honeywell xPMLogic Display Control.

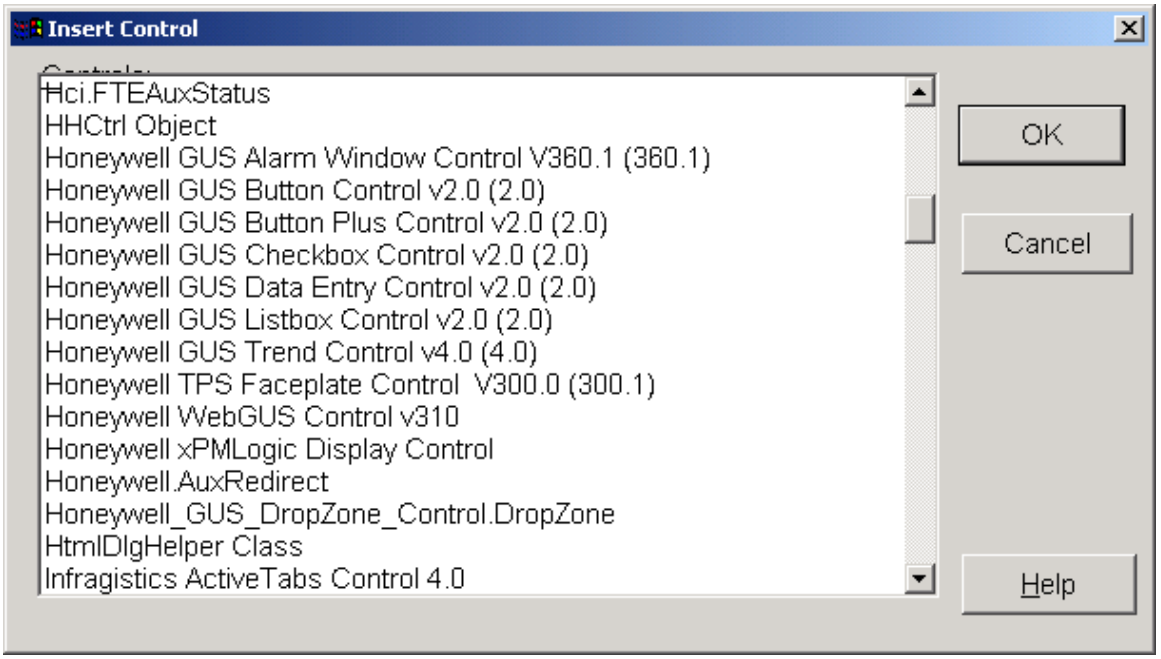


Figure 5: xPM Logic Display control in GUS Insert Control

5.12.3 xPM Logic display control properties

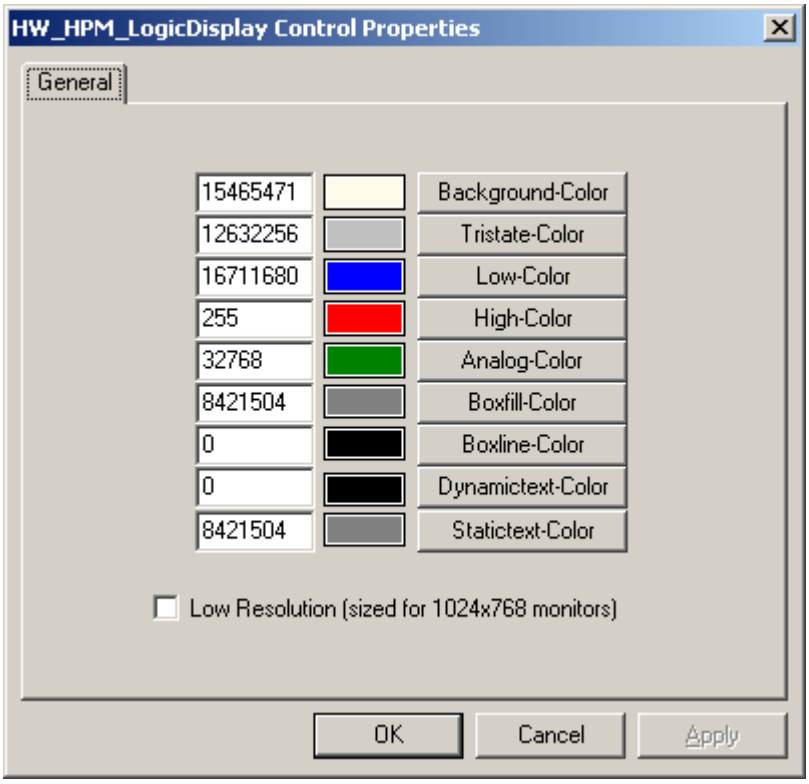
The following properties of the xPM Logic display control can be accessed through scripts:

Property	Type	Description
BgcolorX (Background-Color)	OLE_COLOR	Indicates the background color of the control. The default value is cream (15465471).
TscolorX (Tristate-Color)	OLE_COLOR	Indicates the color of the third state of a digital value. The default value is light gray (12632256).
LocolorX (Low-Color)	OLE_COLOR	Indicates the color of low state digital values. The default value is blue (16722680).
HicolorX (High-Color)	OLE_COLOR	Indicates the color of high state digital values. The default value is red (255).
AnalogcolorX (Analog-Color)	OLE_COLOR	Indicates the color of analog values. The default value is green (32768).
BoxfillcolorX (Boxfill-Color)	OLE_COLOR	Indicates the background color of the box. The default value is medium gray (8421504).
BoxlinecolorX (Boxline-Color)	OLE_COLOR	Indicates the color of the box outline. The default value is black (0).
TextdynacolorX (Dynamictext-Color)	OLE_COLOR	Indicates the color of the point-specific text labels. The default value is black (0).
TextstatcolorX (Statictext-Color)	OLE_COLOR	Indicates the color of the static text labels. The default value is gray (08421504).
LoResolution (Low Resolution)	Boolean	Used for resizing the graphic for 1024x768 resolution monitor.
KlickOnBlock (read-only)	String	Used for displaying the detail display of the logic point.

Attention
The color-related properties and the LoResolution property can be modified through scripts and the properties page of the xPM Logic display control.

To view and modify the properties of the control, perform the following:

1. Drag and drop the xPM Logic display control onto a point.
2. Right-click the point and select **Properties**. The properties page of the xPM Logic display control appears.



3. Modify the colors as required. For example, click **Background-Color** and choose the required color from the color palette.
4. Select **Low Resolution** if the screen resolution of the GUS display is 1024x768.
5. Click **Apply**.
6. Click **OK** to close the properties page.

The following is an example of modifying the control properties through scripts:

```
Sub OnDisplayStartup
‘change background color
Oleobject1.BgColorX='15465472'
‘resize for 1024x768 resolution
Oleobject1.LoResolution='TRUE'
Exit
```

5.12.4 xPM Logic display control methods

The xPM Logic Display control supports the following methods:

- Cmd(Parameter, Index, Value)
- Cmdline(command)

The Cmd method passes data values into the control. The syntax includes parameter to be passed, index, and value. If the method executes successfully, a true value is returned.

5.12.5 Cmd method example

The following is an example for resetting parameters using the Cmd method:

```
Dim iTemp As Integer
‘Reset SO
For iTemp = 1 to 24
Oleobject1.Cmd'SO', iTemp,'@@@@'
Next
‘Reset inputs and outputs
```

```
For iTemp = 1 to 12
Oleobject1.Cmd'LO', iTemp,'@@@@'
Oleobject1.Cmd'LODSTN', iTemp,' '
Oleobject1.Cmd'LISRC', iTemp,' '
Next
‘Reset flag
For iTemp = 1 To 12
OleObject1.Cmd'FL', iTemp,'@@@@'
‘Reset Numeric
For iTemp = 1 to 8
OleObject1.Cmd'NN', iTemp,' '
Next
```

The following configuration parameters must be updated only when the point is assigned or changed:

Parameter	Index	Value source
&N	"	.NAME
C1DESC	"	.C1DESC
C1SRC	"	.C1SRC
C2DESC	"	.C2DESC
C2SRC	"	.C2SRC
C3DESC	"	.C3DESC
C3SRC	"	.C3SRC
C4DESC	"	.C4DESC
C4SRC	"	.C4SRC
DEADBAND	'1' -'24'	.DEADBAND
DLYTIME	'1' -'24'	.DLYTIME
LISRC	'1' -'12'	.LISRC(n) (read as type VAR and then extract the name)
LODSTN	'1' -'12'	.LODSTN(n) (read as type VAR and then extract the name)
LOENBL	'1' -'12'	.LOENBL
LOGALGID	'1' -'24'	.LOGALGID
LOGMIX	"	.LOGMIX
LOSRC	'1' -'12'	.LOSRC
NODENUM	"	.NODENUM
NODETYP	"	.NODETYP
NTWKNUM	"	.NTWKNUM
PERIOD	"	.PERIOD
PNTFORM	"	.PNTFORM
PTDESC	"	.PTDESC
R1	'1' -'24'	.R1
R2	'1' -'24'	.R2
S1	'1' -'24'	.S1
S1REV	'1' -'24'	.S1REV - as'ON' or'OFF'
S2	'1' -'24'	.S2
S2REV	'1' -'24'	.S2REV - as'ON' or'OFF'
S3	'1' -'24'	.S3
S3REV	'1' -'24'	.S3REV - as'ON' or'OFF'
S4	'1' -'24'	.S4

Parameter	Index	Value source
SLOTNUM	"	.SLOTNUM
UNIT	"	.UNIT

The following data parameters must be updated dynamically:

Parameter	Index	Value source
FL	'1' -'12'	.FL(n) - as'0','1' or'----'
LO	'1' -'12'	.L(n) - as'0','1' or'----'
NN	'1' -'12'	.NN(n) - as'0','1' or'----'
PTEXECST	"	.PTEXECST
SO	'1' -'24'	.SO(n) - as'0','1' or'----'

Attention

- Index numbers 1-24 refer to the logic block number within the logic point.
- Index numbers 1-12 refer to the input, output, or logic output enable.

5.12.6 Using Cmdline method to draw graphical representation

The Cmdline method is used for drawing a customized graphical representation of the logic associated with the point. The following drawing commands can be used with the Cmdline method:

Command	Action
Reset	Clears the display area and initializes all the parameter values.
Wire	Draws the logic associated with the point.
ClsOFF	Redraws the logic associated with the point, without clearing the display area.
ClsON	Clears the display area and then redraws the logic associated with the point.

Attention

The commands are case-sensitive.

5.12.7 Cmdline method example

The following is an example for drawing the graphical representation using the Cmdline method:

```
Public NewTag as String
Public PriorTag as String
Public LogicName as String
Public ChangingTag as Boolean
‘First tag name
LogicName=''
‘Read tag name
NewTag=Trim$(UCase$(GetDisplayParameter))
If (NewTag >' ') Then
TextTitle.text=NewTag
Call TagChanged
‘Validate the tag name in TagChanged
Else
TextTitle.SetFocus
Endif
‘If tag name is valid, redraw
OleObject1.CmdLine'ClsOFF'
ChangingTag = TRUE
‘Initialize inputs and outputs
ResetIOVars
If (NewTag <> LogicName) And (LogicName >' ') Then
```

```
PriorTag = LogicName
ButtonPrivTag.ButtonText(0) ='Prior: ' & PriorTag
ButtonPrivTag.Selectable = True
ButtonPrivTag.TextColor = TDC_BLACK
End If
LogicName = NewTag
‘Clear the display area and redraw
OleObject1.CmdLine'ClsON'
‘Initialize parameters
OleObject1.CmdLine'Reset'
Set DispDB.Ent01 = DispDB.Ent02
ButtonRefresh.Selectable = True
ButtonRefresh.ButtonText(0) ='Refresh'
ButtonDetail.Selectable = True
ButtonRefresh.TextColor = TDC_BLACK
ButtonDetail.TextColor = TDC_BLACK
TextTitle.FillColor = TDC_HALF_WHITE
‘set parameter values
SetIOVars
UpdateDisplay
Display.Params.RefreshConfig = TRUE
```

5.12.8 Using the xPM Logic display control in scripts

The process of using the xPM Logic Display control in scripts, involves the following tasks:

- Read the tag name associated with the point
- Reset the control
- Pass (set) all configuration parameters into the control using the Cmd method
- Draw the graphical representation of the point using the Cmdline method

5.12.9 KlickOnBlock property

This is a read-only property whose value is set based on the location within the control selected with the left-mouse button. If that location is within one of the boxes displaying the name of a source or destination tag then the KlickOnBlock property reports that point.parameter as a string; otherwise it is an empty string.

The following is an example of a script on the control that invokes the detail display for the selected point:

```
Sub OnLButtonDown
Dim DetTag As String
Dim i As Long
On Error GoTo NODET
i = Instr(me.KlickOnBlock,'.')
If i > 1 Then
‘selection is on a block with a tag name
DetTag = Left$(me.KlickOnBlock, i-1)
‘truncate to point name only
DETAIL DetTag
End If
Exit Sub
NODET:
Msgbox DetTag &': Detail could not be invoked'
End Sub
```

5.12.10 Scripting targets to call up the xPM Logic display

A target on other GUS displays can be scripted to call up the xPM Logic display. This is accomplished by passing the tag name as the DisplayParameter argument of the InvokeDisplay method. The following is an example of the script:

```
Sub OnLButtonClick
InvokeDisplay'd:\displays\mydisplay.pct','','hpm29lg01'
Exit
```

5.13 Tips for Developing GUS-Compatible ActiveX Controls Using Visual Basic 5.0

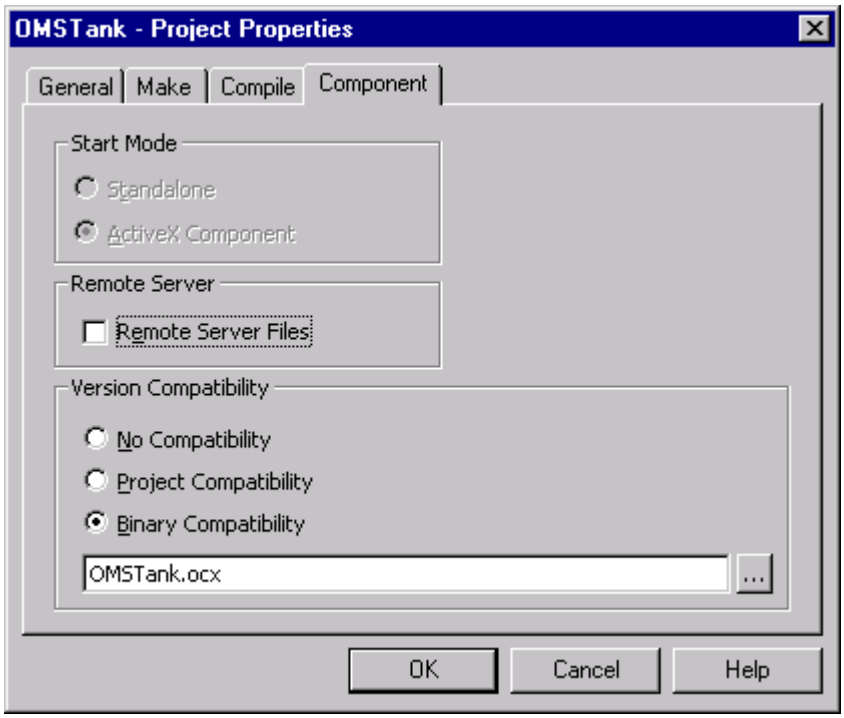
Related topics

“Use Binary Compatibility” on page 190
“Don't Access Ambient Properties on UserControl_ReadProperties” on page 190

5.13.1 Use Binary Compatibility

Be sure to select'Binary Compatibility' option for the project. Failure to do so will destroy (replace) your ActiveX's existing binary ID, by which other applications (e.g., your GUS displays) reference the ActiveX control.

1. Under the project menu, select'<your-project> Properties…'
2. Select the Component tab
3. Select the Binary Compatibility option
4. Enter the control's .ocx file name



5.13.2 Don't Access Ambient Properties on UserControl_ReadProperties

A standard subroutine called UserControl_ReadProperties is called to read the control from its storage. Calls to back to the GUS container made at this point will cause the control to fail to load. An example of this code is the underlined line below:

‘Load property values from storage

Private Sub UserControl_ReadProperties(PropBag As PropertyBag)

m_Caption = PropBag.ReadProperty('Caption', m_def_Caption)

```
m_ShowStpGauge = PropBag.ReadProperty('ShowStpGauge', m_def_ShowStpGauge)
m_ShowLimits = PropBag.ReadProperty('ShowLimits', m_def_ShowLimits)
m_ShowGraphic = PropBag.ReadProperty('ShowGraphic', m_def_ShowGraphic)
Set Font = PropBag.ReadProperty('Font', Ambient.Font)
```

Don't access Ambient object on load!

5.14 IKB Annunciation

Related topics

“IKB Annunciation Purpose” on page 192
“Establish a connection to the IKB” on page 192
“Example for Scripting on the Display Object” on page 192
“Scripting on an object to annunciate” on page 193

5.14.1 IKB Annunciation Purpose

The Honeywell.ikb.client object provides an interface that allows GUS scripts to use the IKB sound contacts. To use this interface, the GUS display needs to establish a connection to the IKB.

5.14.2 Establish a connection to the IKB

To establish a connection to the IKB, use this sequence:

Public IKBobj As Object ‘IKB interface object

Public IKBannunciate as Boolean ‘Is IKB interface active?

Set IKBobj = CreateObject('honeywell.ikb.client')

IKBannunciate = IKBobj.LConnect()

This script should be executed once per display, normally during display startup. If the connection is established, the IKBobj.LConnect() command returns true, and any script within the GUS display can annunciate the IKB sound contacts with the LCmd() function:

bReturn = IKBobj.LCmd(IKB_Code)

where:

IKB_BEEP (= 51003) Sounds on internal annunciator

IKB_SHORT_BEEP (= 51002) Short pulse on internal annunciator

(to be confirmed:

IKB_HORNS_RESET (= 51000) silences active annunciations

IKB_EXT1_ON (= 51004) sets External Annunciator # 1 on

IKB_EXT1_OFF (= 50236) sets External Annunciator # 1 off

IKB_EXT2_ON (= 51005) sets External Annunciator # 1 on

IKB_EXT2_OFF (= 50237) sets External Annunciator # 1 off

IKB_EXT3_ON (= 51006) sets External Annunciator # 1 on

IKB_EXT3_OFF (= 50238) sets External Annunciator # 1 off

)

5.14.3 Example for Scripting on the Display Object

Public IKBobj As Object ‘IKB interface object

Public IKBannunciate as Boolean ‘Is IKB interface active?

Sub OnDisplayStartup()

```
on error resume next
Set IKBobj = CreateObject('honeywell.ikb.client')
IKBannunciate = IKBobj.LConnect()
End Sub
Sub OnDisplayShutdown()
Set IKBobj = Nothing
End Sub
```

5.14.4 Scripting on an object to annunciate

```
Public IKBobj As Object ‘IKB interface object
Public IKBannunciate as Boolean ‘Is IKB interface active?
Sub OnLButtonUp()
dim ret as boolean
If (IKBannunciate) then
ret = IKBobj.LCmd(IKB_HORN_3)
else
‘no IKB connection - so beep
beep
end if
End Sub
```

6 Actors

Related topics

6.1 Actors Introduction

This section contains a set of tables mapping the TPS Network picture editor actors (built-in functions) to the capabilities of BasicScript. The table lists the name of the actor, indicates whether or not it exists in BasicScript, and provides comments that describe the functionality of the actors, with associated scripting examples.

Note: These actors are supported by GUS scripting: QPRINT, QDISPLAY, QFILE.

6.2 Relationship Between TDC Actors and Scripts

This section describes the built-in functions of BasicScript that provide capabilities equivalent to the built-in functions of the TPS Network picture editor. Most of the functions of the TPS Network picture editor have been implemented in BasicScript, except where they were more naturally expressed by existing BasicScript language elements. This section consists of a set of translation notes intended for those who are translating TPS Network pictures to GUS Displays and a set of tables that map TPS Network built-in functions to the capabilities of BasicScript.

NOTICE: Use TDC Actors ONLY in scripts handling USER INTERFACE events

TDC Actors will only run in scripts handling user-interface events. That is, the script must expect a user action like OnLButtonClick, OnDblClick, etc.

If a TDC Actor is in a script like OnPeriodicUpdate or OnDataChange, you will get a runtime error.

6.2.1 Notes for usage of the built-in functions

This section describes how to use the built-in functions of BasicScript that are intended to provide capabilities equivalent to the built-in functions of the TPS Network picture editor. Refer to the Picture Editor Reference Manual and the Actors Manual for details. The notes provided here enable you to map from TPS Network to the capabilities of BasicScript.

6.2.2 Tag names and DDB Syntax

Tagnames in BasicScript are properties of the built-in LCN object. For example, A100.PV becomes LCN.A100.PV.

INT01 becomes DISPDB.INT01

Refer to “Access to Process Data from Scripts” on page 54 for additional information.

6.2.3 X, Y coordinates

Display coordinates in the TPS Network picture editor are character-based, with the bottom left corner as the origin. Coordinates in BasicScript are expressed in pixels, with the left top as the origin. Note that this affects the following functions: Operator Input Actors, MOVE and REM_MOVE.

6.2.4 ON, OFF

BasicScript does not have built-in constants for ON and OFF. Use TRUE and FALSE.

6.2.5 Enumerations

Enumeration constants are provided via the 'enum, built-in function. For example, QUE_KEY(ENTER) becomes QUE_KEY(enum('$buttons:enter')).

6.2.6 Date, Time, Duration

Use the BasicScript date literal format when passing date literals to built-in functions. That is, literal dates are specified by appending a # characters to the beginning and end of a time or date. For example, a time of 07:45 becomes #07:45#.

If the year is greater than 80, it should be expressed as 19xx.

For example, a date of 12-29-85 becomes #12-29-1985#.

Durations are expressed using a different time base in BasicScript. TPS Network is based on 1979-Jan-1, while BasicScript uses 1899-Dec-31.

To construct a duration from a time value, add an offset value of 28856. For example, duration of 133-05:00:59 become 133+28856+#05:00:59#.

The standard BasicScript functionality of Date (data type) date literals applies.

6.2.7 BasicScript extensions are either commands or functions

BasicScript extensions can be implemented either as a function or a command. A function returns a value, and has a parenthesized argument list. A command does not return a value, and its argument list is not parenthesized. The actor functions that do not return a value have been implemented as commands. For example, DELAY (0,1,1) in the TPS Network becomes DELAY 0, 1, 1.

6.2.8 BasicScript Line Restrictions

BasicScripts allows only one command per line. For example,

DELAY(0,1,1);GROUP(100,1)

in the TPS Network becomes

Delay 0, 1, 1

GROUP 100,1

6.3 Display Actors

This section contains a set of tables mapping the TPS Network picture editor display actors (built-in functions) to the capabilities of BasicScript. Each table lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

Attention

Beep after OnDisplayActor event occurs only if the event is not handled by the container, and then cancelled by the container's CancelEvent call. (see WebGUS CancelEvent method).

The normal response is no beep.

Tip

See the Actor's Manual SW09-655 to review Actor syntax and examples.

Attention

If the customer has a script that responds to the raised event and calls the CancelFunction method, then the default behavior of the Actor will not occur.

Related topics

“ALARMANN” on page 200
“ALARMSUM” on page 200
“ATTRIBPS” on page 201
“BOX_OV” on page 201
“BOX_PS” on page 201
“BOX_STAT” on page 201
“CANCLPRT” on page 201
“CHGZONE” on page 201
“CLEAR_CZ” on page 202
“CONSSTAT” on page 202
“CROSSCRN” on page 202
“CUSTSAVE” on page 202
“DETAIL” on page 202
“DET_PAGE” on page 203
“DMD_UPD” on page 203
“DSP_FILE” on page 203
“EHR” on page 203
“FRM_SCRN” on page 203
“GROUP” on page 204
“GRP_EDIT” on page 204
“GRPTREND” on page 204
“HWPTSUM” on page 204
“HWY_STAT” on page 205
“INITFLOP” on page 205
“IOM_DIAG” on page 205
“LM_DIAG” on page 205
“MODGROUP” on page 205
“MSGSUM” on page 206
“MULTI_OV” on page 206
“NIM_DIAG” on page 206

6.3.1 ALARMANN

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Alarm Annunciator Display in Native Window

6.3.2 ALARMSUM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
UnitNumber *UnitNumber {integer} is the index of the desired unit (0 for the Area Summary).*	Yes	Invokes Native Window or GUS Alarm Summary Display, as configured.

6.3.3 ATTRIBPS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Attribute Point Summary Display in Native Window.

6.3.4 BOX_OV

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.5 BOX_PS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Box Point Summary Display in Native Window.

6.3.6 BOX_STAT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network, Module Network {integer} is the HIWay number. Module {integer} is the Box number.	Yes	Invokes Box Status Display in Native Window.

6.3.7 CANCLPRT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
PrinterNbr PrinterNbr {integer} is the number of a configured console printer.	Yes	Cancel Printout on Console printer.

6.3.8 CHGZONE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.9 CLEAR_CZ

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.10 CONSSTAT

Attention
GUS scripting extension, not available in the LCN Picture Editor.

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Console Status Display on Native Window.

6.3.11 CROSSCRN

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Station Station {integer} is the number of the station within the console to receive the next command.	Yes	Routes the next actor to the Native Window on a different station within the console. **Exception**: The SCHEM actor in GUS is treated as an InvokeDisplay request and will NOT be rerouted by the CROSSCRN actor.

6.3.12 CUSTSAVE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Custom Save Display in Native Window.

6.3.13 DETAIL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
PointName PointName {string} is the entity to be displayed.	Yes	Invokes Detail Display in Native Window.

6.3.14 DET_PAGE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
PointName, PageNbr PointName {string} is the entity to be displayed. PageNbr {integer} is the page within the Detail display to be shown.	Yes	Invokes specified page of Detail Display in Native Window

6.3.15 DMD_UPD

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.16 DSP_FILE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
PathName Pathname {string} is the TPN path for the requested file.	Yes	Invokes Native Window to display a TPN text file

6.3.17 EHR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Event History Display in Native Window.

6.3.18 FRM_SCRN

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.19 GROUP

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
GroupNbr, SelectedSlot GroupNbr {integer} is the group to be displayed. SelectedSlot {optional integer} is the slot to be initially selected. If 0 or omitted, then no slot will be selected. Display {optional integer} sets the display format: 0 = select based on configuration rules 1 = invoke the NativeWindow Group Display 2 = invoke the GUS GroupDisplay If GROUP actor is invoked without setting display format, it must default to 0.	Yes	Invokes the requested GROUP Display.

6.3.20 GRP_EDIT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Group Edit Display in Native Window.

6.3.21 GRPTREND

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
GroupNbr, SelectedSlot GroupNbr {integer} is the group to be displayed. SelectedSlot {optional integer} is the slot to be initially selected. If 0 or omitted, then no slot will be selected.	Yes	Invokes Group Trend Display in Native Window.

6.3.22 HWPTSUM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network, Module, Slot Network {integer} is the HIWay number. Module {integer} is the Box number. Slot {integer} is the slot within the Module.	Yes	Invokes UCN Slot Summary Display in Native Window.

6.3.23 HWY_STAT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network Network {integer} is the HIWay number.	Yes	Invokes Hiway Status Display in Native Window.

6.3.24 INITFLOP

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Floppy Initialization Display in Native Window.

6.3.25 IOM_DIAG

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network, Module, Slot Network {integer} is the UCN number Module {integer} is the device number Slot {integer} is the slot within the device	Yes	Invokes UCN IO Module Diagnostic Display in Native Window.

6.3.26 LM_DIAG

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network, Module	Yes	Invokes Logic Manager Diagnostic Display in Native Window.

6.3.27 MODGROUP

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
GroupNbr GroupNumber {integer} is the module group to be displayed. Valid numbers are 401 - 450.	Yes	Invokes Module Group Display in Native Window.

6.3.28 MSGSUM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Message Summary Display in Native Window.

6.3.29 MULTI_OV

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.30 NIM_DIAG

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network, Module Network {integer} is the UCN number Module {integer} is the device number.	Yes	Invokes NIM Diagnostic Display in Native Window.

6.3.31 NODE_PS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Node Point Summary Display in Native Window.

6.3.32 OSUMMENU

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Organizational Summary Display in Native Window.

6.3.33 OVERLAY

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.34 OVERVIEW

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Overview Display in Native Window.

6.3.35 OVW_EDIT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Overview Edit Display in Native Window.

6.3.36 PALLETE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.37 PM_CMD

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	

6.3.38 PMM_DIAG

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network, Module Network {integer} is the UCN number Module {integer} is the device number.	Yes	Invokes PMM (or NIM) Diagnostic Display in Native Window.

6.3.39 PM_STAT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network, Module Network {integer} is the UCN number Module {integer} is the device number.	Yes	Invokes Process Manager Status Display in Native Window.

6.3.40 PROCMOD

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
PointName PointName {string} is the entity to be displayed.	Yes	Invokes Process Module Display for sequence point in Native Window.

6.3.41 PROMPT_C

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Clears Prompt in the status bar of the (owner) display window.

6.3.42 PROMPT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Message Message {string} is the message to be displayed as a prompt.	Yes	Displays the prompt string in the status bar of the (owner) display window.

6.3.43 PRT_FILE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
PathName, PrinterNbr Pathname {string} is the TPN path for the requested file PrinterNbr {integer} is the number of a configured console printer.	Yes	Print contents of an TPN text file; on Native Window printer.

6.3.44 PVR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Process Variable Retrieval Display in Native Window.

6.3.45 REM_MOVE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.46 REM_STR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.47 REP_MENU

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Report/ Menu in Native Window.

6.3.48 RTJ

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Real Time Journal Display in Native Window.

6.3.49 SCHEM

Attention
When used in conjunction with the CROSSCRN Actor, the display must be limited to a maximum of 8 characters.

Parameter(s)	Available as a scripting extension	GUSDisplay Script Behavior
DisplayName DisplayName {string} is the name of the requested display.	Yes	Invokes GUS Display or Native Window schematic.
Example: SCHEM('!Excel.exe')		

6.3.50 SYS_MENU

Attention
Not available as a TPN Actor for classic LCN pictures.

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
		Invokes System Menu Display in the Native Window.

6.3.51 SYS_STAT

Attention
Not available as a TPN Actor for classic LCN pictures.

Parameter(s)	Available as a scripting extension	GUSDisplay Script Behavior
		Invokes System Status Display in the Native Window.

6.3.52 TITLESUM

Parameter(s)	Available as a scripting extension	GUSDisplay Script Behavior
	Yes	Invokes Title Summary Display in Native Window.

6.3.53 TRENDOVW

Parameter(s)	Available as a scripting extension	GUSDisplay Script Behavior
	Yes	Invokes Trend Overview Display in Native Window.

6.3.54 UCN_STAT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Network Network {integer} is the UCN number.	Yes	Invokes UCN Status Display in Native Window.

6.3.55 UNIT_PS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Unit Point Summary Display in Native Window.

6.3.56 UNITSUM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Invokes Unit(Module) Summary Display in Native Window.

6.3.57 UNITTRND

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
UnitNumber UnitNumber {integer} is the index of the desired unit.	Yes	Invokes Unit Trend Display in Native Window.

6.3.58 UPDATE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.3.59 USAGE_PS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
		Invokes Usage Point Summary Display in Native Window.

6.3.60 USER_CZ

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.4 Keyboard Event Actors

This section contains a set of tables mapping the TPS Network picture editor keyboard event actors (built-in functions) to the capabilities of BasicScript. Each table lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor, both in a ES-T node script .

Tip
See the Actor's Manual, SW09-655, to review Actor syntax and examples.

6.4.1 $KEYCHG

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
KeyLevel KeyLevel (long) is the enumeration value for the access level. Acceptable states are: VIEW, OPR, SUP, and ENGR Format is $KEYCHG(enum('Keylevl:state')).	Yes	KEY_CHG sets internal keylevel status at for the GUS display ($keylevl).

6.4.2 $KEYRST

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	KEY_RST restores $keylevl to match workstation key position.

6.4.3 ENTER

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Error. The functionality is provided by: QUE_KEY(enum('$button:enter')

6.4.4 ENT_EXEC

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Message Message (String) is displayed in the message box.	Yes	A dialog box is displayed to handle what would have appeared in the message area of the UIS display. Message box: CANCEL will exit the current script, OK will continue script execution.

6.4.5 KEY

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Station, KeyCode Station {integer} is the number of the station within the console. Keycode {long} is the enumeration value of the keystroke - see the documentation on the $BUTTONS enumeration set.	Yes	Sends a simulated keystroke to the NativeWindow running on the specified station.

6.4.6 MOVE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
X, Y, Absolute, Region X {integer} is the X-coordinate. Y {integer} is the Y-coordinate. Absolute {Boolean} - if TRUE, X&Y identify the coordinates within the window where the cursor will be placed. If FALSE, the cursor will move from its current location by the value of X & Y. Region {integer} is ignored within GUS, but still required, so use 0.	Yes	Moves cursor within the display

6.4.7 QUE_KEY

Tip
QUE_KEY works on a TPS Domain desktop, but not on a non-TPS Domain desktop. On a non-TPS Domain desktop the error message'No permission to change access level' when an attempt to change the access level is made.

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
KeyName KeyName {string} is the enumeration value of the keystroke - see the documentation on the $BUTTONS enumeration set. Yes		Sends a simulated keystroke to the object that has focus. Exception: KEY_SUP, KEY_ENG and KEY_OPR set the keylevel for the entire workstation. **Caution**: Any script that issues a QUE_KEY of a ramp key (RAISE, LOWER, FSTRAISE or FAST_LOW) must also issue a QUE_KEY of the related ramp release key (RAISE_RL, LOWER_RL, FSTRASRL, or FSTLWRL)
		Example: QUE_KEY enum('$buttons:enter')

6.4.8 Keylevel collector

Tip
The current KeyLevel can be obtained using a DispDB variable or the Collector shown below.

Collector Name	GUS Display Script Behavior
$KEYLEVEL	Returns the state of the GUS Display Keylock Level. Collector('$KEYLEVL') will return one of the following values: VIEW, OPR, SUP, and ENGR

6.5 Operator Input Actors

This section contains a set of tables mapping the TPS Network picture editor operator event actors (built-in functions) to the capabilities of BasicScript. Each table lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor, both in a GUS display script.

These commands open up a Text Input Port (TIP) for data entry and convert the input value to the specified data type. A message box identifies the conversion failed. All of the Actors have the same set of parameters, except R_SENM and R_SENM_N which have an additional (first) parameter , EnumSet.

Tip
See the Actor's Manual, SW09-655, to review Actor syntax and examples.

6.5.1 Parameter list

The Operator Input Actors contain the following parameters. All the parameters must be enclosed in a set of parenthesis where:

- **X {integer}** is the X-coordinate for the left-side of the TIP.
  - If Absolute is true, this is calculated from the edge of the Display.
  - If Absolute is false, this is calculated from the left-side of the scripted object.
- **Y {integer}** is the Y-coordinate within the display for the top of the TIP.
  - If Absolute is true, this is calculated from the edge of the Display.
  - If Absolute is false, this is calculated from the top of the scripted object.
- **TipLength {integer}** is the length of the TIP in pixels.
  - Allow 9 pixels for each text digit.
  - Allow up to 14 pixels per'W' characters.
- **Prompt {string}** message to be displayed on the status bar while the TIP is open.
- **Absolute {Boolean}**
  - If TRUE, coordinates are calculated from the top-left corner of the display.
  - If FALSE, coordinates are calculated based on the position of the **scripted** object. In either case, the location is automatically adjusted when the display is zoomed.
- **Region {integer}**
  - 1 places the TIP in the upper-left corner of the display (ignoring the X & Y values).
  - O places the TIP according to the X and Y values.

6.5.2 R_BOOL( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above.	Yes	Returns string as a Boolean value

6.5.3 R_DATE( )

Tip
R_Date() is valid for dates since 10/1/1981. If a date between 1/1/79 and 9/30/81 is entered it will be converted to a duration and returned as the number of days since 1900.

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	Yes	Returns formatted date string as a {double} date/time value

6.5.4 R_DUR( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	Yes	Returns formatted duration (days hours minutes seconds) date as a {double} date/time value

6.5.5 R_ENM( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	No	

6.5.6 R_ENT( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	Yes	Returns the entity value for the named point

6.5.7 R_ENT_N( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	Yes	Returns the string if it is the name of a valid LCN point

6.5.8 R_INT( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	Yes	Returns string as an integer value

6.5.9 R_PAR( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	Yes	Returns the string if it is the name of a valid non-subscripted LCN parameter

6.5.10 R_REAL( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above	Yes	Returns string as a {single} real value

6.5.11 R_SENM( )

Parameter(s)	Available as a scripting extension	GUSDisplay Script Behavior
EnumSet( ) See “Parameter list” on page 215 above EnumSet {string} is the name of an LCN enumeration set, such as'MODE'.	Yes	Returns the {long} enumeration value if the entry is the name of state within the EnumSet

6.5.12 R_ SENM_N( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
EnumSet( ) See “Parameter list” on page 215 above EnumSet {string} is the name of an LCN enumeration set, such as'MODE'.	Yes	Returns the string if it is the name of state within the EnumSet

6.5.13 R_STR( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above.	Yes	Returns a string value

6.5.14 R_TIME( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above.	Yes	Returns formatted date string as a {double} date/time value

6.5.15 R_VAR( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
See “Parameter list” on page 215 above.	Yes	Returns the Var value for the named point.parameter

6.6 Store to DDB Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor, both in a GUSdisplay script.

6.6.1 S_BOOL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement. For example: lcn.a100.pv = TRUE.

6.6.2 S_DATE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement. For example: S_DATE(DATIME1, C_DATTIM(12-31-96, 11:30:32)) from a TPS Network picture becomes DISPDB.DATIME1 = (#12/31/1996# + #11:30:32#)..

6.6.3 S_DUR

Parameter(s)	Available as a scripting extension	GUSDisplay Script Behavior
	No	Use the BasicScript assignment statement.

6.6.4 S_TIME

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement.

6.6.5 S_ENT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement. For example: S_ENT(ENT01, G_ENT(ENT02)) from a TPS Network picture becomes SET DISPDB.ENT01 = DISPDB.ENT02.

6.6.6 S_INT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement. For example: S_INT(INT01, G_INT(INT02)) from a TPS Network picture becomes DISPDB.INT01 = DISPDB.INT02

6.6.7 S_REAL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement.

6.6.8 S_SENM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement.

6.6.9 S_STR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement.

6.6.10 S_VAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement.

6.7 Read Data and Store into DDB Actor

The table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor, both in a GUS display script

6.7.1 RS_LOC

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript assignment statement. For example: RS _LOC(INT01, 1, 2 , 5,'enter', TRUE, 0)) from a TPS Network picture becomes DISPDB.INT01 = R_INT(1, 2, 5,'enter', TRUE, 0).

6.8 Read from DDB Actors (DispDB)

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actorin a GUS display script .

6.8.1 G_BOOL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.8.2 G_DATIME

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.8.3 G_ENT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.8.4 G_INT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.8.5 G_REAL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.8.6 G_SENM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.8.7 G_STR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.8.8 G_VAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9 Store to DDB Actors (DispDB)

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actorin a GUS display script

6.9.1 SS_BOOL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.2 SS_DATE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.3 SS_DUR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.4 SS_ENT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.5 SS_INT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.6 SS_REAL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.7 SS_STR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.8 SS_TIME

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.9 SS_VAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.9.10 EQ_LIST

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.10 Read Data and Store into System Actor

The table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor, in a GUS display script

6.10.1 RS_SYS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.11 Read from System DDB Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

6.11.1 GS_BOOL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.11.2 GS_DATIME

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.11.3 GS_ENT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.11.4 GS_INT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.11.5 GS_REAL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.11.6 GS_SENM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.11.7 GS_STR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.11.8 GS_VAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object. For example: S_VAR(VAR01, GS_VAR(AMPT.CODSTN(1)) from a TPS Network picture becomes Set DISPDB.VAR01 = lcn.AMPT.CODSTN(1).value.

6.11.9 GS_VAR_S

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Refer to “Display Database (DispDB)” on page 62 for a discussion of the DispDB built-in object.

6.12 Range Check Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

6.12.1 RNG_INT( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Value, LowLimit, NighLimit) Value {integer} is the value to be tested. LowLimit {integer} is the minimum acceptable value. HighLimit {integer} is the maximum acceptable value.	Yes	Reports an error 1011 is the Value is less than the LowLimit or greater than the HighLimit; Otherwise, returns the Value.

6.12.2 RNG_REAL( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Value, LowLimit, NighLimit) Value {integer} is the value to be tested. LowLimit {integer} is the minimum acceptable value. HighLimit {integer} is the maximum acceptable value.	Yes	Reports an error 1011 is the Value is less than the LowLimit or greater than the HighLimit; Otherwise, returns the Value.

6.12.3 RNG_STR( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Value, MinLength, MaxLength) Value {string} is the value to be tested. MinLength {integer} is the minimum acceptable length. MaxLength {integer} is the maximum acceptable length.	Yes	Reports an error 1011 is the Value is shorter than the MinLength or longer than the MaxLEngth; Otherwise, returns the Value.

6.12.4 RNG_VAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.12.5 RNG_LOC

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.13 General Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

6.13.1 C_DATTIM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript '+' operator.

6.13.2 C_V_ENUM

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.13.3 C_VAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.13.4 CONCAT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript '+' operator

6.13.5 DELAY

Attention
A Delay in any User Interface script will block all other User Interface script for the duration. (OnDataChange scripts are not blocked as long as a Delay is NOT included in any OnDataChange script.).

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Minutes, Seconds, Milliseconds	Yes	Pauses the script for the specified period of time. Minutes, Seconds, Milliseconds are all required integer counts - so a delay of 1.5 minutes would be scripted as; Delay 1, 30, 0.

6.13.6 EXTR_ENT( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Tag) Tag {string} is the tag name to be tested.	Yes	Returns the entity name (as whatever follows a'\') extracted from a string representing a TPN tag name (no lcn. prefix allowed). Reports error 1012 if more than 16 characters follow the'\'.

6.13.7 EXTR_ID( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Tag) Tag {string} is the tag name to be tested.	Yes	Returns the pnid extracted from a string representing a TPN tag name (no lcn. prefix allowed). Reports error 1012 if the tag does not include a'\' or if more than 2 characters precede the'\'.

6.13.8 EXTR_PAR( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Tag) Tag {string} is the tag name to be tested.	Yes	Returns the parameter name (as whatever follows a'.' extracted from a string representing a TPN tag name (no lcn. prefix allowed). Reports error 1012 if the tag has too many characters on either side of the'.'

6.13.9 NOP

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	Allowed, but has no effect on script behavior

6.13.10 TDC_OPTION

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

6.14 Trend Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script

6.14.1 TR_ADD

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the AddTrace method of a Trend control.

6.14.2 TR_CLINE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the HairlineCursor and HairlineReadout properties of a Trend control. Note that the Hairline functionality of the Trend control replaces the centerline of the TPS Network trend phantom.

6.14.3 TR_DATA

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the DataSource property of a Trend control.

6.14.4 TR_DEL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the DeleteTrace property of a Trend control.

6.14.5 TR_RANGE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the YRangeHigh and YRangeLow properties of a Trend control.

6.14.6 TR_SCALE

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the YScaleHigh and YScaleLow properties of a Trend control.

6.14.7 TR_SCRLL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the RuntimeScrolling, ScrollBackTime, ScrollForwardTime and YScaleLow properties of a Trend control.

6.14.8 TR_TIME

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the TimeBase and YScaleLow properties of a Trend control.

6.15 Arithmetic Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

6.15.1 ADD_I

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘+' operator

6.15.2 SUB_I

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘-' operator.

6.15.3 MUL_I

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘*' operator.

6.15.4 DIV_I

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘/' operator.

6.15.5 MOD_I

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘mod' operator.

6.15.6 NEG_I

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript unary ‘-‘ operator.

6.15.7 ADD_R

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘+' operator.

6.15.8 SUB_R

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘-' operator.

6.15.9 MUL_R

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘*' operator.

6.15.10 DIV_R

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript ‘/' operator.

6.15.11 NEG_R

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript unary ‘-‘ operator.

6.16 Logical Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

6.16.1 AND

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript unary ‘AND‘ operator.

6.16.2 OR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript unary ‘OR‘ operator.

6.16.3 XOR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript unary ‘XOR‘ operator.

6.16.4 NOT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript unary ‘NOT‘ operator.

6.17 Comparative Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

6.17.1 CMP_I

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript comparison operator.

6.17.2 CMP_R

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript comparison operator.

6.17.3 CMP_S

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript comparison operator.

6.17.4 CMP_E

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript comparison operator.

6.18 Conditional Actors

The table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor, in a GUS display script

6.18.1 IF

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript'if' statement. For example: `IF (CMP_R(GS_REAL(A100.PV), GT, 10)); SS_BOOL(A100.PV, TRUE); ELSE; SS_BOOL(A100.PV, FALSE); ENDIF in a TPS Network picture becomes if (LCN.A100.PV >= 10)) then LCN.A100.PV = TRUE else LCN.A100.PV = FALSE end if.`

6.18.2 EXISTS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	`if lcn.a100.pv.status = HOPC_CONFIGURATION_ERROR then EXISTS = false else EXISTS = true end if`

6.19 Conversion Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script

6.19.1 FLOAT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the BasicScript'sng' function.

6.19.2 CNV_I( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Value, Format) Value {integer} is the integer to be formatted. Format {string} specifies the format to be applied. The string should begin with'I' followed by an appropriate combination of: + - Z 9.	Yes	Converts an integer to a formatted string.

6.19.3 CNV_UID( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(UnitID) UnitID {string} is a 2-character Unit Identifier.	Yes	Returns the integer index to the Unit in the current Area configuration.

6.19.4 IE_ENT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the ‘external' property of an entity DDB item. For example S_STR(STRING01, IE_ENT(ENT01)) in a TPS Network picture becomes DISPDB.STRING01= DISPDB.ENT01.[NAME].

6.19.5 IE_VAR_E

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the ‘external' property of an entity DDB item. For example: S_STR(STRING01, IE_VAR_E(VAR01)) in a TPS Network picture becomes DISPDB.STRING01 = EXTR_ENT(DISPDB.VAR01.NAME).

6.19.6 IE_VAR_P

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the EXTR_PAR function on the value of the external property of an VAR DDB item. For example: S_STR(STRING01, IE_VAR_P(VAR01)) in a TPS Network picture becomes DISPDB.STRING01 = EXTR_PAR(DISPDB.VAR01.NAME) (EXTR_PAR is a new function in BasicScript. It does not exist on the TPS Network).

6.19.7 IE_VAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the ‘external' property of a VAR DDB item. For example: S_STR(STRING01, IE_VAR(ENT01)) in a TPS Network picture becomes DISPDB.STRING01 = DISPDB.VAR01.NAME.

6.19.8 ROUND( )

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
(Value) Value {single} is the real number to be rounded.	Yes	Returns the integer closest to the Value. (generates Overflow error 6 if value is outside of 16-bit integer range).

6.19.9 TRUNC

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Translated into Basic'FIX'.

6.19.10 EI_ENT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the external property of an ENT DDB item. For example: S_ENT(ENT01, EI_ENT(CONCAT(G_STR(STRING01), G_STR(STRING02)))) in a TPS Network picture becomes SET DISPDB.ENT01 = GETENT(DISPDB.STRING01 + DISPDB.STRING02).

6.20 Keylock Actors

Each table below lists the name of the actor, its parameters if any, indicates whether or not it exists in BasicScript, and provides comments that describe the behavior of the actor in a GUS display script.

6.20.1 $KEYRST

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	Yes	KEY_RST restores $keylevl to match workstation key position.

6.20.2 $KEYCHG

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
KeyLevel KeyLevel (long) is the enumeration value for the access level. Acceptable states are: VIEW, OPR, SUP, and ENGR Format is $KEYCHG(enum('Keylevl:state')).	Yes	KEY_CHG sets internal keylevel status at for the GUS display ($keylevl).

6.20.3 KEY_ENG

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

7 Collectors

Related topics

“Runtime Data Collection” on page 244
“Alarm Collectors” on page 245
“History Collectors” on page 249

7.1 Runtime Data Collection

Related topics

“CollectHistory” on page 244

7.1.1 CollectHistory

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
InitFlag, RefTime InitFlag {boolean}: If True, history data will appear as'***' while collection request is outstanding. If False, the prior value will continue to be displayed until the new values are returned from the HM. RefTime {optional date} specifies the Date/Time to be used as the'current time' for the collection request. If omitted, the current TPN system time is used.	Yes	Invokes data retrieval for all history collectors in the display.

7.2 Alarm Collectors

Attention

AlarmStatus is an enumeration. Within collectors, it should be typed as one of the following values (not enclosed in quotes):

$ALRMSTS:ACKLO, $ALRMSTS:ACKHI, $ALRMSTS:ACKEM, $ALRMSTS:UNACKLO, $ALRMSTS:UNACKHI, or $ALRMSTS:UNACKEM.

Attention

The syntax for these collectors generally requires nesting of quoted strings. In scripts, the nested strings must be delimited by pairs of double-qoutes. For example:

- collector('$UNITSTS("01")')

If an inline parameter is referenced as within the string, then an additional level of nested quotes is required. For example:

- Collector('$PRIMSTS(""\pe(tag)"")'

Related topics

“ACKSTAT( )” on page 245
“$ADDBCNT( )” on page 246
“$ADDBSTS( )” on page 246
“$ANNCNT( )” on page 246
“$ANNSTS( )” on page 246
“$AREACNT( )” on page 246
“$AREASTS” on page 247
“$PDDBCNT( )” on page 247
“$PDDBSTS( )” on page 247
“$PNTCNT( )” on page 247
“$PNTSTS( )” on page 247
“$PRIMCNT( )” on page 248
“$PRIMSTS( )” on page 248
“$UNITCNT( )” on page 248
“$UNITSTS( )” on page 248

7.2.1 ACKSTAT( )

Parameter(s)	GUS Display Script Behavior
(Point) Point {string} is the name of a TPN point (no lcn.prefix or quotes).	Returns a string reporting the alarm state within the current area & console of the requested TPN point. Values are: 'NOALARM' 'UNAKALRM' 'AKDALRM'

Actor	GUS Display Script Behavior
ACK	Acknowledges all points that are scripted with the ACKSTAT collector within the display.

7.2.2 $ADDBCNT( )

Parameter(s)	GUS Display Script Behavior
(StringNN, AlarmStatus) StringNN is the identifier of a displDB string (no prefix). AlarmStatus see, “Alarm Collectors” on page 245 note above.	Returns the {integer} number of alarms in the specified alarm status category for the annunciator group title specified in the DispDB string.

7.2.3 $ADDBSTS( )

Parameter(s)	GUS Display Script Behavior
(StringNN) StringNN is the identifier of a displDB string (no prefix).	Returns the {enumeration} composite alarm status of a the annunciator group title specified in the DispDB string.

7.2.4 $ANNCNT( )

Parameter(s)	GUS Display Script Behavior
(AnnumGrp, AlarmStatus) AnnunGrp is a configured Annunciator group title. AlarmStatus see, “Alarm Collectors” on page 245 note above.	Returns the {integer} number of alarms in the specified alarm status category for the specified annunciator group title.

7.2.5 $ANNSTS( )

Parameter(s)	GUS Display Script Behavior
(AnnumGrp) AnnunGrp is a configured Annunciator group title.	Returns the {enumeration} composite alarm status of a specified annunciator group.

7.2.6 $AREACNT( )

Parameter(s)	GUS Display Script Behavior
(AlarmStatus AlarmStatus) see, “Alarm Collectors” on page 245 note above.	Returns the {integer} number of alarms in a specified alarm status category for the local area.

7.2.7 $AREASTS

Parameter(s)	GUS Display Script Behavior
	Returns the {enumeration} composite alarm status of the local area.

7.2.8 $PDDBCNT( )

Parameter(s)	GUS Display Script Behavior
(StringNN, AlarmStatus) StringNN is the identifier of a displDB string (no prefix). AlarmStatus see, “Alarm Collectors” on page 245 note above.	Returns the {integer} number of alarms in the specified alarm status category for the PRIMMOD name specified in the DispDB string.

7.2.9 $PDDBSTS( )

Parameter(s)	GUS Display Script Behavior
(StringNN) StringNN is the identifier of a displDB string (no prefix).	Returns the {enumeration} composite alarm status of a the PRIMMOD name specified in the DispDB string

7.2.10 $PNTCNT( )

Parameter(s)	GUS Display Script Behavior
(Point, AlarmStatus) Point is the Entity Id of a configured TPN point (no lcn. prefix). AlarmStatus see, “Alarm Collectors” on page 245 note above.	Returns the {integer} number of alarms in a specified alarm status category for the specified point.

7.2.11 $PNTSTS( )

Parameter(s)	GUS Display Script Behavior
(Point) Point is the Entity Id of a configured TPN point (no lcn. prefix).	Returns the {enumeration} composite alarm status of a specified point

7.2.12 $PRIMCNT( )

Parameter(s)	GUS Display Script Behavior
(Prommod, AlarmStatus) Primmod is the Entity Id of a configured TPN primmod point (no lcn. Prefix). AlarmStatus see, “Alarm Collectors” on page 245 note above.	Returns the {integer} number of alarms in a specified alarm status.

7.2.13 $PRIMSTS( )

Parameter(s)	GUS Display Script Behavior
(Prommod) Primmod is the Entity Id of a configured TPN primmod point (no lcn. Prefix).	Returns the {enumeration} composite alarm status of a specified Primmod name.

7.2.14 $UNITCNT( )

Parameter(s)	GUS Display Script Behavior
(UnitId, AlarmStatus) *UnitId is the 2-character identifier of a unit in the current area.* AlarmStatus see, “Alarm Collectors” on page 245 note above.	Returns the {integer} number of alarms in a specified alarm status category for the specified unit.

7.2.15 $UNITSTS( )

Parameter(s)	GUS Display Script Behavior
(UnitId) *UnitId is the 2-character identifier of a unit in the current area.*	Returns the {enumeration} composite alarm status of a specified unit.

7.3 History Collectors

Collectors are accessed by GUS scripting using the Collector() function. Collector parameters are validated by GUS Display Builder Validate command. Each Collector may be put into a collection group, with defined scan rate. At runtime, data is collected whenever the CollectHistory Actor is executed.

Attention

For the Collectors below, valid StartTime values are;

- 0 Current Time (in which case, Offset counts periods prior to the current time).
- 1 Beginning of the Hour, Offset counts periods after start.
- 2 Beginning of the Shift, Offset counts periods after start.
- 3 Beginning of the Day, Offset counts periods after start.
- If any other value is specified, no validation error is reported, but the runtime collection will return incorrect data.

Attention

The vintage HM** and TM** history collectors are NOT supported in GUS scripting. Attempts to use them will result in erroneous values being returned. These collectors are;

HMDAY, HMHOUR, HMMIN, HMMONTH, HMSHIFT, HMUSR_AV, and TM**

7.3.1 DAY_S( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns the {enumeration} composite alarm status of a specified unit.

7.3.2 DAY_T( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns the timestamp for a Daily historized value.

7.3.3 DAY_V( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset, ValueType) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time. ValueType {integer} is choice of: 1. Average 2. Maximum 3. Minimum 4. Summation	Retrieves the {single} Daily historized value of variable.

7.3.4 HOUR_S( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns'*' if there are not enough samples to provide a valid Hourly average. Otherwise, returns blank.

7.3.5 HOUR_T( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns the timestamp for a Hourly historized value.

7.3.6 HOUR_V( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset, ValueType) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time. ValueType {integer} is choice of: 1. Average 2. Maximum 3. Minimum 4. Summation	Retrieves the {single} Hourly historized value of variable.

7.3.7 MINUTE_T( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns the timestamp for a historized snapshot value.

7.3.8 MINUTE_V( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Retrieves the {single} historized snapshot value of variable.

7.3.9 MONTH_S( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns'*' if there are not enough samples to provide a valid Monthly average. Otherwise, returns blank.

7.3.10 MONTH_T( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns the timestamp for a Monthly historized value.

7.3.11 MONTH_V( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset, ValueType) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time. ValueType {integer} is choice of: 1. Average 2. Maximum 3. Minimum 4. Summation	Retrieves the {single} Monthly historized value of variable.

7.3.12 SHIFT_S( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns'*' if there are not enough samples to provide a valid Shift average. Otherwise, returns blank.

7.3.13 SHIFT_T( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns the timestamp for a Shift historized value.

7.3.14 SHIFT_V( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset, ValueType) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time. ValueType {integer} is choice of: 1. Average 2. Maximum 3. Minimum 4. Summation	Retrieves the {single} Shift historized value of variable.

7.3.15 USER_S( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns'*' if there are not enough samples to provide a valid User-Defined average. Otherwise, returns blank.

7.3.16 USER_T( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time.	Returns the timestamp for a User-Defined historized value.

7.3.17 USER_V( )

Parameter(s)	GUS Display Script Behavior
(Tag, StartTime, Offset, ValueType) Tag is the point.parameter {no lcn. prefix} of a TPN Tag. StartTime {integer} - see “History Collectors” on page 249 Attention above. Offset {integer} is the number of periods from the start time. ValueType {integer} is choice of: 1. Average 2. Maximum 3. Minimum 4. Summation	Retrieves the {single} User-Defined historized value of variable.

7.3.18 I_***_S

Parameter(s)	GUS Display Script Behavior
	Indirect version of Above DAY_*, HOUR_*, MINUTE_*, MONTH_*, SHIFT_*, and USER_*. The only difference is that the Tag parameter is the name of a DispDB [without the prefix] variable instead of a TPN point.parameter.

7.3.19 I_***_T

Parameter(s)	GUS Display Script Behavior
	Indirect version of Above DAY_*, HOUR_*, MINUTE_*, MONTH_*, SHIFT_*, and USER_*. The only difference is that the Tag parameter is the name of a DispDB [without the prefix] variable instead of a TPN point.parameter.

7.3.20 I_***_V

Parameter(s)	GUS Display Script Behavior
	Indirect version of Above DAY_*, HOUR_*, MINUTE_*, MONTH_*, SHIFT_*, and USER_*. The only difference is that the Tag parameter is the name of a DispDB [without the prefix] variable instead of a TPN point.parameter.

8 Database and Functions

Related topics

“Display Data Base” on page 258
“Intrinsic Functions” on page 262
“New Functions Exclusive to ES-T for TPS Network Pcture Functions” on page 264

8.1 Display Data Base

NT side DDB provides 256 each of the primary data types with local and global scopes. These are allocated on display basis. When one display invokes another display, its Global DDB values are copied to the new display.

8.1.1 $ALMCOLR

GUS Display Script Behavior
READONLY: An integer that specifies the alarm priority color option selected in the NCF: 0 = two color option (red, yellow), 1 = 3 color option (user selected colors).

8.1.2 $AL_ENTY

GUS Display Script Behavior
An Entity containing the identity of the last TPN point that is selected on the Alarm Summary, Unit Alarm Summary, Alarm Annunciation or Organizational Summary displays. Changes are propagated to all displays on the workstation.

Attention
The GUS Alarm Summary is not supported on Experion nodes.

8.1.3 $ARAID

GUS Display Script Behavior
An 8-character string name for the current area.

8.1.4 $ARAID01…..$ARAID10

GUS Display Script Behavior
READONLY: An 8-character string name for each indicated area as configured in the NCF.

8.1.5 $ARADS01….$ARADS10

GUS Display Script Behavior
READONLY: A 24-character string description for each indicated area as configured in the NCF.

8.1.6 $ARADSC

GUS Display Script Behavior
READONLY: A 24-character string description of the current area

8.1.7 $CONNUM

GUS Display Script Behavior
READONLY: An integer that represents the current console number

8.1.8 $CONDSC

GUS Display Script Behavior
READONLY: A 24-character string description of the current console

8.1.9 $CONDS01….$CONDS10

GUS Display Script Behavior
READONLY: A 24-character string description of the indicated console as configured in the NCF.

8.1.10 $CZ_ENTY

GUS Display Script Behavior
An Entity used to identify the point of interest for a ChangeZone.

8.1.11 $DT_ENTY

GUS Display Script Behavior
An Entity containing the identity of the last TPN point for which a Detail Display was invoked. It is initialized to NULL_ID. Changes are propagated to all displays on the workstation.

8.1.12 $EALMCLR

GUS Display Script Behavior
READONLY: An integer that specifies the Emergency Alarm priority color option selected in the NCF.

8.1.13 $FASTRL

GUS Display Script Behavior
READONLY: An integer that represent the Fast Raise/Lower percentage per increment.

8.1.14 $GRPBASE001….$GRPBASE400

GUS Display Script Behavior
READONLY: An object ID that can be used to access the group definition for the current area. The properties of interest are: Title {string} = Title of the group. ExtIdLst (ix) – where ix is integer 1-8 {string} = Name of the TPN entity assigned to slot number ix in the group.

8.1.15 $HALMCLR

GUS Display Script Behavior
READONLY: An integer that specifies the High Alarm priority color option selected in the NCF.

8.1.16 $KEYLEVL

GUS Display Script Behavior
Enumeration identifying the display's current keylevel. Whenever a display is invoked, this value is initialized to match the GUS Validated keylevel for the station. The acceptable values are'VIEW' (0x00580000),'OPR' (0x00580001),'SUP' (0x00580002), and'ENGR' (0x00580003).

8.1.17 $LALMCLR

GUS Display Script Behavior
READONLY: An integer that specifies the Low Alarm priority color option selected in the NCF.

8.1.18 $LNG_TAG

GUS Display Script Behavior
READONLY: This global DDB contains - true if the TPN is configured to support long tag names (1-16 chars), false if not configured for long tag names (1-8 chars).

8.1.19 $MY_AREA

GUS Display Script Behavior
READONLY: An integer that represents the area number on which the display is running.

8.1.20 $MY_PNA

GUS Display Script Behavior
READONLY: An integer that represents the node number on which the display is running.

8.1.21 $PERSON

GUS Display Script Behavior
READONLY: A string that contains the TPN node personality type ('USUNIV' for ES-T).

8.1.22 $REDYEL

GUS Display Script Behavior
READONLY: An enumeration that contains the minimum alarm priority indicated by the color red. It contains one of the following values:'LOW' (0x000A0002),'HIGH' (0x000A0003), or'EMERGNCY' (0x000A0004).

8.1.23 $STNNUM

GUS Display Script Behavior
READONLY: An integer that represents the station number on which the display is running.

8.2 Intrinsic Functions

Related topics

“BIT_TEST” on page 262
“EXTERNAL” on page 262
“INTERNAL” on page 262
“STATUS” on page 263

8.2.1 BIT_TEST

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Value, Bit Value {integer} is the word to be tested. Bit {integer} is the bit to evaluate.	Yes	Returns {Boolean} True if the specified bit within the Value is set (1); Otherwise, returns False.

8.2.2 EXTERNAL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the ‘external' property of a parameter object. For example: EXTERNAL(A100.PV) in a TPS Network picture becomes LCN.A100.PV.EXTERNAL, where PV is an enumeration.

8.2.3 INTERNAL

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the ‘internal' property of a parameter object. For example: INTERNAL(A100.PV) in a TPS Network picture becomes LCN.A100.PV.INTERNAL.

8.2.4 STATUS

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	Use the ‘status' property of a point object. For example. STATUS(A100.PV) in a TPS Network picture becomes LCN.A100.PV.STATUS.

8.3 New Functions Exclusive to ES-T for TPS Network Pcture Functions

Related topics

“TDCFORMAT” on page 264
“GETTREND” on page 264
“LCN.Update” on page 265
“EXTR_PAR” on page 265

8.3.1 TDCFORMAT

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Value, Format. Value {integer, double or single} is the integer to be formatted. Format {string} specifies the format to be applied. If Value is an integer, the string should begin with'I' followed by an appropriate combination of: + - Z 9. If Value is a double or single, the string should begin with'R' followed by an appropriate combination of + -. Z 9	Yes	Converts an integer to a formatted string. TDC VALUE like formatter that supports INTEGER, REAL, STRING, BOOLEAN, DATE/TIME, and UNKNOWN. TDCFORMAT(data, format string) This function has two arguments. The first is the data to be formatted. The type of the first argument depends on the contents of the format string. The second argument is a string containing a TPS Network picture editor format string. The return value is a string containing the formatted data. The type of the format string must match the type of data being formatted. The type of an LCN object is not known until runtime, so you must use the'unknown' or'general' type format strings. If you wish to use a more specific format string for LCN objects, use one of the Basic conversion functions such as csng, cstr, date, or cint on the input argument.

8.3.2 GETTREND

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
	No	

8.3.3 LCN.Update

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
CollectionGroup CollectionGroup {integer} specifies the Data Collection Group to update. 0 specifies All groups. Note: the syntax is a script line command without parentheses: LCN.Update n	Yes	Causes an immediate read to update the values of all lcn. data access items configured for the specified Data Collection Group

8.3.4 EXTR_PAR

Parameter(s)	Available as a scripting extension	GUS Display Script Behavior
Tag Tag {string} is the tag name to be tested.	Yes	Returns the parameter name (as whatever follows a'.' extracted from a string representing a TPN tag name (no lcn. prefix allowed). Reports error 1012 if the tag has too many characters on either side of the'.'

9 Notices

Trademarks

Experion®, PlantScape®, SafeBrowse®, TotalPlant®, and TDC 3000® are registered trademarks of Honeywell International, Inc.

OneWireless™ is a trademark of Honeywell International, Inc.

Other trademarks

Microsoft and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Trademarks that appear in this document are used only to the benefit of the trademark owner, with no intention of trademark infringement.

Third-party licenses

This product may contain or be derived from materials, including software, of third parties. The third party materials may be subject to licenses, notices, restrictions and obligations imposed by the licensor. The licenses, notices, restrictions and obligations, if any, may be found in the materials accompanying the product, in the documents or files accompanying such third party materials, in a file named third_party_licenses on the media containing the product, or at http://www.honeywell.com/ps/thirdpartylicenses.

9.1 Documentation feedback

You can find the most up-to-date documents on the Honeywell Process Solutions support website at:

http://www.honeywellprocess.com/support

If you have comments about Honeywell Process Solutions documentation, send your feedback to:

hpsdocs@honeywell.com

Use this email address to provide feedback, or to report errors and omissions in the documentation. For immediate help with a technical problem, contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

9.2 How to report a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, please follow the instructions at:

https://honeywell.com/pages/vulnerabilityreporting.aspx

Submit the requested information to Honeywell using one of the following methods:

- Send an email to security@honeywell.com.

  or

- Contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

9.3 Support

For support, contact your local Honeywell Process Solutions Customer Contact Center (CCC). To find your local CCC visit the website, https://www.honeywellprocess.com/en-US/contact-us/customer-support-contacts/Pages/default.aspx.

9.4 Training classes

Honeywell holds technical training classes on Experion PKS. These classes are taught by experts in the field of process control systems. For more information about these classes, contact your Honeywell representative, or see http://www.automationcollege.com.