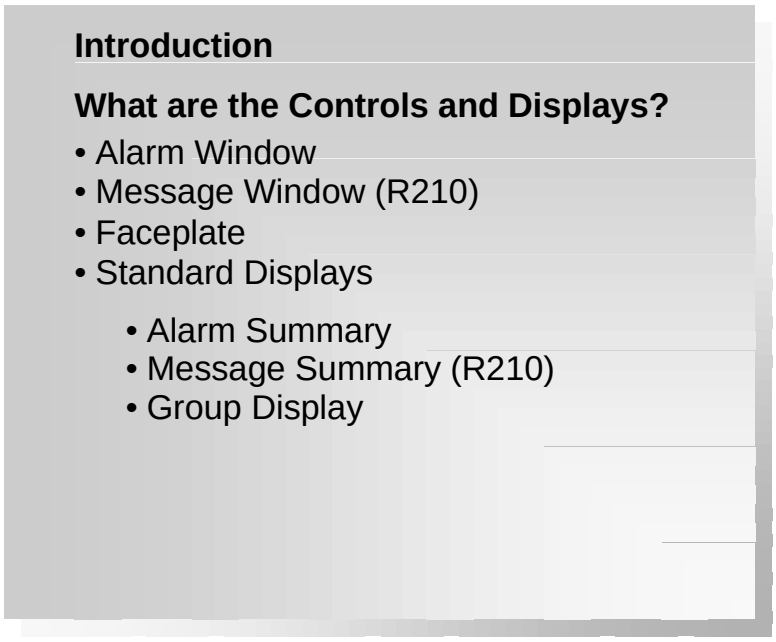

Controls

**R200 Alarm, Message, Faceplate and Standard Display Controls**




## Introduction

Several controls introduced in R200 provide a classic Universal Station (U.S.) functionality to your GUS displays.

## What are the Controls and Displays?

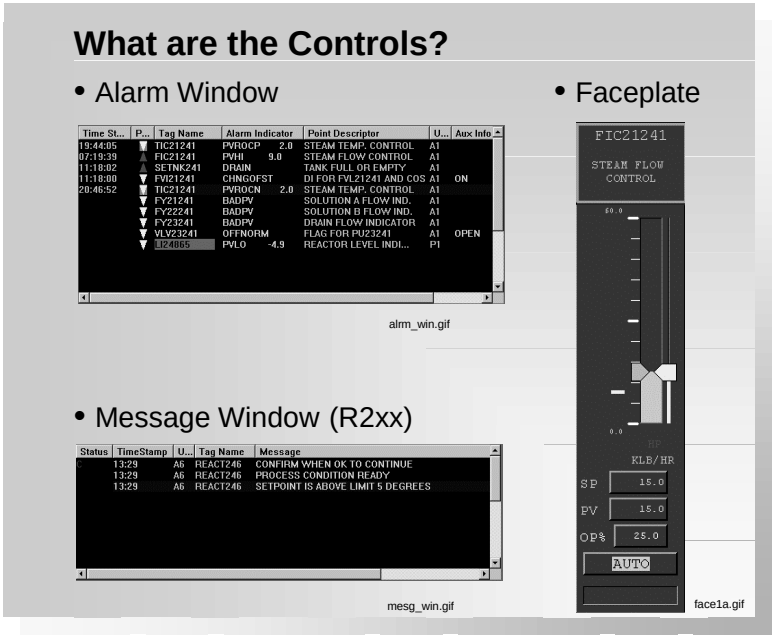
The controls are the following:

- Alarm Window
- Message Window (R2xx)
- Faceplate

The displays are following:

- Standard displays : Alarm Summary and Message Summary (R2xx).
- The Group display

## R200 Alarm, Message, and Faceplate Controls



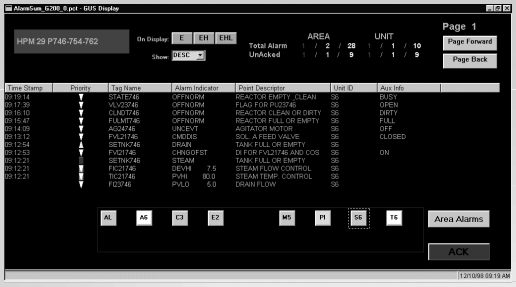

### What are the Controls?

The controls are following:

- Alarm Window - displays similar alarm data as a US or Native Window.
- Message Window - displays similar message data as US or Native Window
- Faceplate - displays process point data, such as regulatory control or digital composite points

## What are the Standard Displays?

- Alarm Summary
- Message Summary (R2xx)
  (not shown)
- Group Display




alm_sum_runr.gif

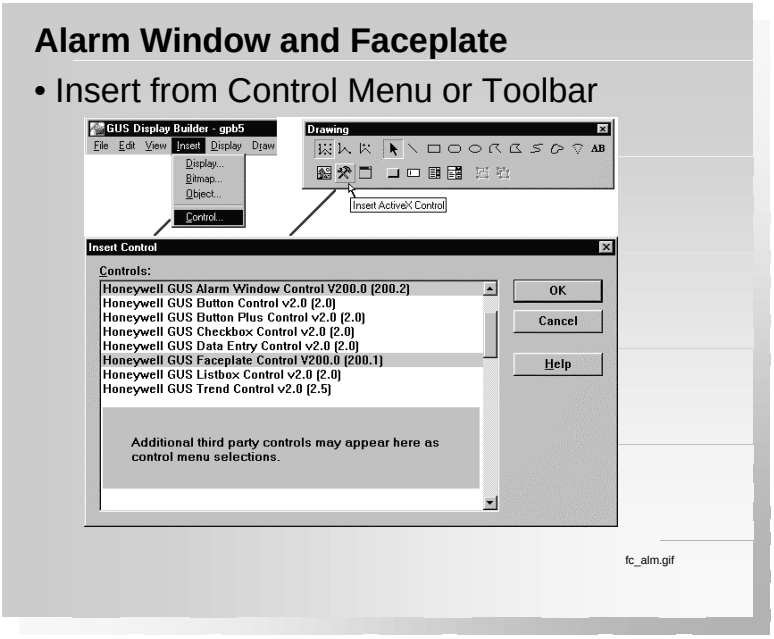

utly_group_ex2.gif

## What are the Standard Displays?

- Standard display - the Alarm Summary and Message Summary (R2xx) displays contain alarm and message window controls. They are called up from the Integrated Keyboard instead of the standard Native Window summary displays.

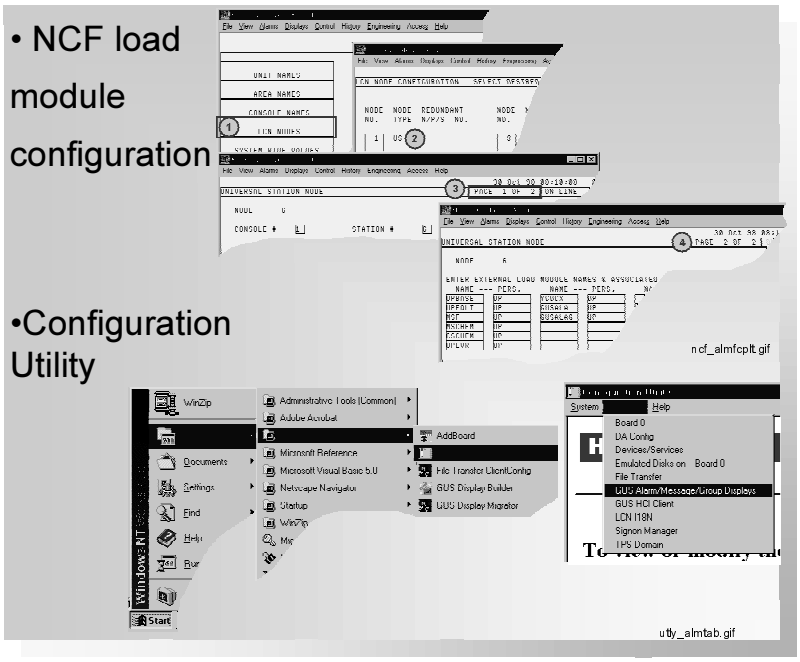
- The Group display provides status and parameters of up to 8 points.

# Configuration Overview




## Overview

Insert the R200 control from either the Control Menu or choose the ActiveX button  from the Drawing Toolbar.




## Configuration Prerequisites

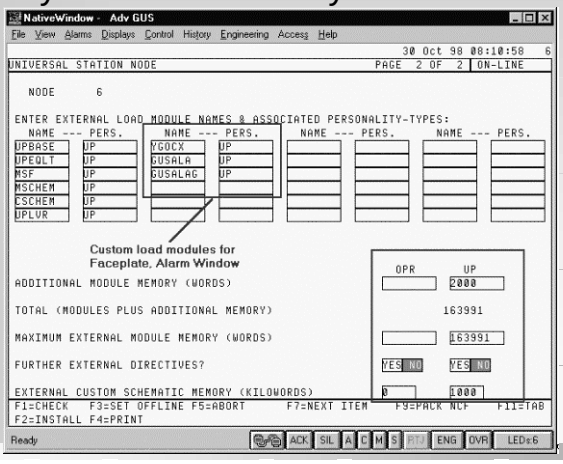
You have to perform the following configuration steps:

•If you want to use the Faceplate and/or Alarm Window control, add the necessary load modules as part of NCF configuration.
•If you want to use the Alarm Summary and/or Group Display, enable the displays from the Configuration Utility (Choose GUS Alarm/Message/Group Displays from the Configuration Utility menu).

# Configuration from NCF

## NCF Load Modules

- Required for Alarm Window and Faceplate
- YGOCX, GUSALA, GUSALAG
- Specify additional memory



ncf_ocx2.gif

## Add the NCF Load Modules

External Load Modules represent optional software system files that provide functionality to your GUS. For example, MSCHEM and CSCHEM support the GUS Display Builder, where you can build and run GUS displays that are displayed from a Windows NT environment. Other modules such as YGOCX, GUSALA, and GUSALAG support Honeywell controls such as a Faceplate control and Alarm Window. The External Load Modules reside on the History Module and are loaded into your GUS when the GUS personality is loaded. If you decide to use these functions and have the GUS software operate properly, the optional External Load Modules must be referenced in your NCF.

## Increase the memory on NCF Node Page 2

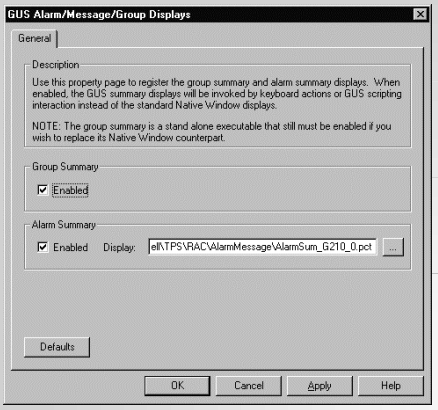
The total amount of memory is system calculated for you and displayed as the TOTAL (MODULES PLUS ADDITIONAL MEMORY). You cannot directly change this value; however, additional memory is required to support the optional GUS software. The steps to do this are:

- Enter 2000 into the text entry port for ADDITIONAL MODULE MEMORY (WORDS) UP port.
- Add the sum of 2000 and your system calculated total from the previous step.
- Put the sum into the MAXIMUM EXTERNAL MODULE MEMORY (WORDS) UP port.

## Configuration Utility Dialog

### IKB Invocation for Standard Display

- Enable from Configuration Utility
- Group Display : GroupDisp.exe




r210.alm.confutly.gif

### Enable from Configuration Utility

If you want the Group Display (shown earlier) to be invoked from the Integrated Keyboard (IKB) automatically instead of the Native Window's version of this display, then you have to enable that behavior from the Configuration Utility.

If the Native Window has focus (that is, the Native Window is the currently active window), a Native Window Group display appears when you press the Group key on the Integrated keyboard (IKB). If a GUS Display (that is, a .pct file) currently has focus, the GUS Group Display appears when you press the Group key on the Integrated keyboard.

# Alarm Window Appearance and Operation

**Shows same alarm data as US, only in your GUS Display**

**Example Alarm Window Appearance and Operation**


**Note: Additional Operations require scripted objects**

## Background

The Alarm Window shows similar alarm data as a classic US or Native Window.

## Alarm Window Operation

When one of Alarms on the Alarm Window is selected, the background color of the selected line turns blue.

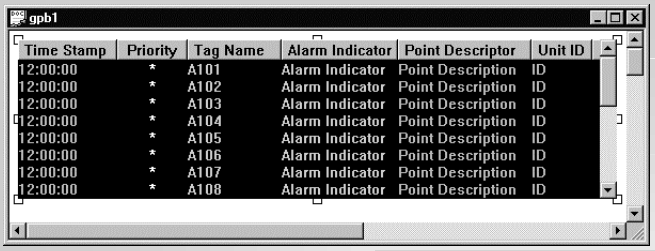
After selection of an Alarm, you can invoke related displays by pressing keys on the Integrated Keyboard such as DETAIL/ GROUP/ ASSOC DISP.

The selection of an Alarm will be canceled when the selected alarm is acknowledged or another alarm is selected.

For other Alarm operations such as acknowledging an alarm, you need to provide the Alarm Window with scripting to provide the necessary alarm responses.

# Alarm Window Buildtime Appearance

## Example Buildtime Appearance

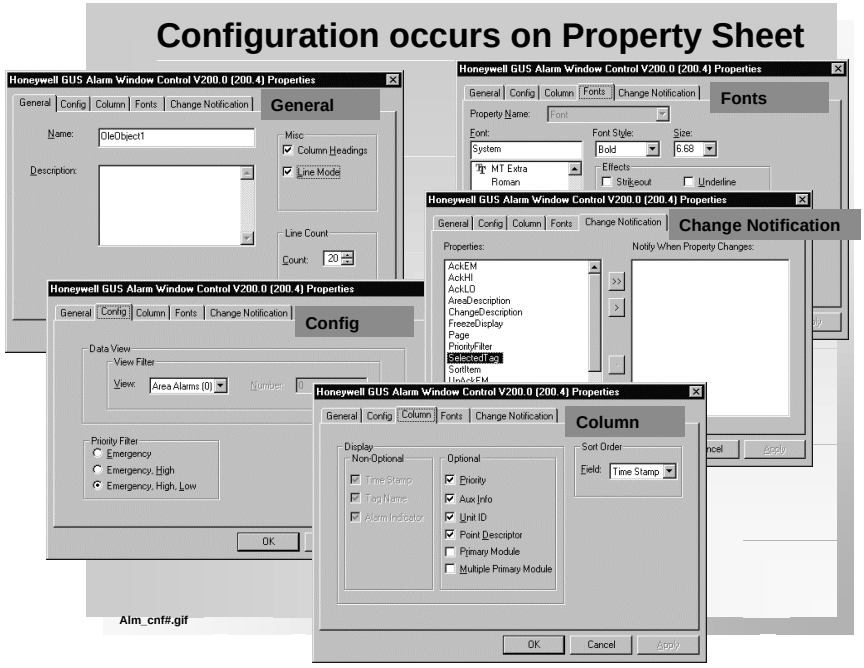



r210.alm.wind.gif

## Background

At buildtime, the Alarm Window appears with column headings and can be sized as needed.

# Alarm Window Configuration




## Alarm Window configuration occurs on property sheets

**General Page**
From this page you determine whether or not you want to enable column headings and/or line mode.When line mode is enabled, a selected tag that appears on a line turns blue. You can then acknowledge an alarm by line. If line mode is disabled, you acknowledge alarms by page.
**If you want to allow an operator to select a point in alarm and provide a button to call up a detail display, you MUST enable the line mode property** (default is non-selected). This is where you configure the maximum number of lines per page (up to 20 lines per page).

**Configuration Page**
View Filter - Display types can represent area or unit alarms.
Priority Filter - Select which alarm priorities should be displayed.

**Column Page**
Display Fields - Time Stamp*, Tag name*, Alarm Indicator*, Priority, Aux Info, Unit ID, Point Descriptor, Primary Module, Multiple Primary Module
(where * = always will be enabled to appear on the display).
Sort Order - Sort by timestamp or priority

**Font Page**
Controls font type, style, size.

**Change Notification Page**
Allows you to select properties which activate events. This page is used with the scripting function, OnPropertyChange(). Script using this event is executed each time one of the selected properties on this page changes. The property change can happen automatically as a result of an alarm control change or be a user initiated change. An example of an automatic change is if the unacknowledged emergency alarm count (UnAckEm) changes. An example of a user initiated change is selecting a tag (SelectedTag). Properties such as these or others from the Notification Page can be used to trigger OnPropertyChange() script execution.

## Alarm Window Scripting

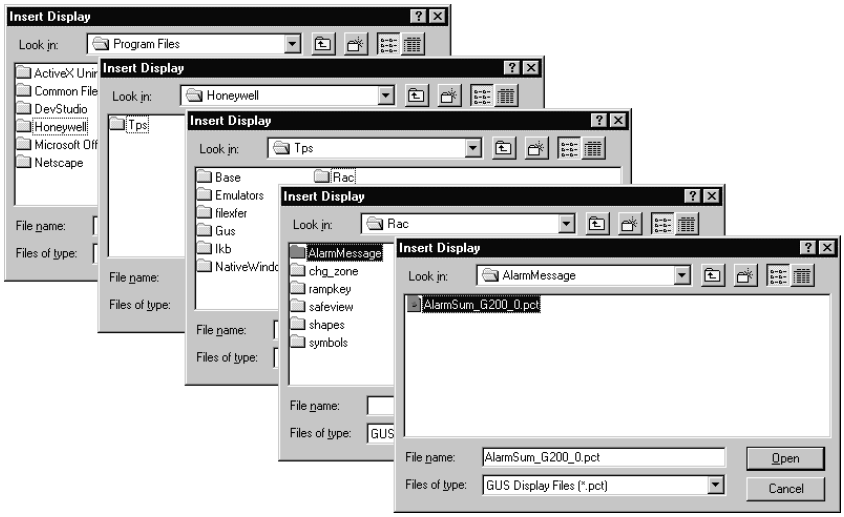
**To script the Alarm Window Control, use**

- **Properties**
- **Methods**
- **Events**

**Your best scripting resource is the Alarm Summary pct!**

### Location of the Alarm Summary Display

The AlarmSum.pct file resides in the RAC folder.




alm_rac.gif

## Alarm Window Scripting Example 1

**Example button script uses the selected tag property to call up a Detail display.**

```
Sub OnDataChange()       'DetailTarget
        'DispDB.STRING01 = oleobject1.selectedtag
        if DispDB.STRING01        = "" then
                me.visible = false
                me.selectable = false
        else
                me.visible = true
                me.selectable = true
        end if
End Sub

Sub OnLButtonUp()
        on Error Goto catch
        if me.visible = true then
                'DispDB.STRING01 = oleobject1.selectedtag
                DETAIL DispDB.STRING01
                oleobject1.selectedtag = " "
                dispdb.string01 = " "
        end if
        exit sub
catch:
End Sub
```

### Example of Scripting property: Button to call up Detail display

The following script provides a way for a button to call up a selected tag. The OnDataChange script uses the selected tag property of the alarm window control, OleObject1. When a tag selection occurs, the button, self referenced as "me," is made visible. If there is no tag selection, the detail button remains invisible. (Note: An OnPropertyChange script on the alarm window stores the selected tag as vValue into dispb.string01.)

```
Sub OnDataChange()         'DetailTarget
       'DispDB.STRING01    = oleobject1.selectedtag = vValue
       if DispDB.STRING01  = "" then
              me.visible = false
              me.selectable = false
       else
              me.visible = true
              me.selectable = true
       end if
End Sub
```

The trick in using a button only when it is needed in a display is that the button is selectable even if it invisible, so you have to find a way to prevent an inadvertent selection from making something happen. The way to do that is use the button's visible property. The OnLButtonUp script checks to see if the button is visible. If the button is visible, the DETAIL actor is used to call up the selected tagname. After calling up the tagname, the selected tag property and dispdb.string01 is set to a null string.

```
Sub OnLButtonUp()
       on Error Goto catch
       if me.visible = true then
              'DispDB.STRING01           = oleobject1.selectedtag
              DETAIL DispDB.STRING01
              oleobject1.selectedtag = " "
              dispdb.string01 = " "
       end if
exit sub
catch:
End Sub
```

## Alarm Window Scripting Example 2

**Example button script uses ACK method**

```
Sub OnLButtonUp()'AckTarget

        on Error Goto catch
        oleobject1.ack
        exit sub
catch:
End Sub
```

**Example uses PageForward PageBack methods**

```
Sub OnPageBack()
        on Error Goto catch
        if (dispdb.int201 <= 1)   then
                oleobject1.pageforward
                oleobject1.pageforward
                oleobject1.pageforward
                oleobject1.pageforward
        else
                oleobject1.pageback
        end if
        exit sub
catch:
End Sub
```

### Example of Scripting method: ACK and Page Back Buttons

Recalling that methods provide actions helps interpret the syntax in the following examples, which are in the form"objectname.method". The use of the alarm acknowledge method is the easiest to examine, its script form here is OleObject1.ack, where OleObject1 is the alarm window control name.

```
Sub OnLButtonUp()       'AckTarget
            on Error Goto catch
            oleobject1.ack
exit sub
catch:
End Sub
```

When using a method you have to anticipate possible runtime scenarios. In the next example, the script could have used the pageback method by itself. But, there may be a runtime scenario where there is only one page of alarm data. In that case, if there is only one page of alarm data, the page forward method is used to go past 4 of the 5 pages back to the original (and only displayed) page of alarm data.

```
Sub OnPageBack()
      on Error Goto catch
      ' dispdb.int201 set to alarm page in another script
      if (dispdb.int201 <= 1)  then
            oleobject1.pageforward
            oleobject1.pageforward
            oleobject1.pageforward
            oleobject1.pageforward
      else
            oleobject1.pageback
      end if
exit sub
catch:
End Sub
```

## Alarm Window Scripting Example 3

### OnPropertyChange Event Example

```
Sub OnPropertyChange(nProperty As Integer,nUnitNumber As Integer,vValue As Variant)

        On Error GoTo errorhandler
        Select Case nProperty
        Case ALARM_PAGE  'page
                         Dim i As Integer
                         i =     vValue
                         If i > 5 Then
                                 i = 1
                         End If
                         pagenum.text = i
                         dispdb.int201 = vValue

        Case ALARM_SELECTEDTAG    'selected tag
                         selectedtagname.text = vValue
                         dispdb.string01 = vValue
        End Select
Exit Sub
errorhandler:
        msgbox "OnPropertyChange  " & err.description & erl
End Sub
```

## Example of Scripting OnPropertyChange Event

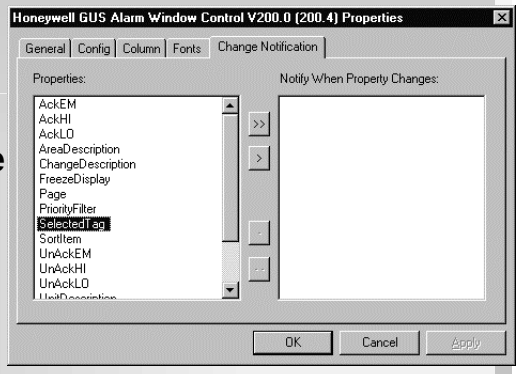
The following example uses the OnPropertyChange event to display the Alarm Page name and selected tag in the alarm window. The Property Change event can be triggered by either a system event, such as an alarm page change, or user action, such as selecting a tag. After this example there is some background discussion on how to use the OnPropertyChange event. (Note: The script fragment for case ALARM_SELECTED TAG is an abbreviated version of the actual script. Refer to the Alarm Summary pct for a full version of the code.)

```
Sub OnPropertyChange(nProperty As Integer,nUnitNumber As Integer,vValue As Variant)

            On Error GoTo errorhandler
            Select Case nProperty
            Case ALARM_PAGE            'page
                                       Dim i As Integer
                                       i =          vValue
                                       If i > 5 Then
                                                    i = 1
                                       End If
                                       pagenum.text = i
                                       dispdb.int201 = vValue

            Case ALARM_SELECTEDTAG     'selected tag
                                       selectedtagname.text = vValue
                                       dispdb.string01 = vValue
            End Select
Exit Sub
errorhandler:
            msgbox "OnPropertyChange  " & err.description & erl
End Sub
```

# Configure Alarm Window for Property Change

**To use the OnPropertyChange event**

**• Select properties at buildtime**




**• Employ OnPropertyChange syntax**

**• Use built in scripting constants**

**Best resource is Alarm Summary pct!**

## Select properties at buildtime

The Change Notification property page provides a list of properties available for change event notification. The default for all events is OFF for optimal performance. You may not change the selection of notification events at runtime. You should 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.

## Employ OnPropertyChange event syntax

The Basic Script syntax for this event is:

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

•nProperty is an integer value which is an index indicating the specific property whose value has changed.

•nUnitNumber is an integer which indicates the unit number, if applicable. If the property is not associated with a unit, 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: You can alternatively reference the Alarm Window Control explicitly to get the current value of the property).

## Use built in scripting constants

The lists shown below identify built-in scripting constants. Each constant corresponds to the integer value of a property. The OnPropertyChange() function compares the integer value of the nProperty parameter to the specified constants to locate the property and react to the change.

| | |
|---|---|
| AckEm | ALARM_AckEm |
| AckHi | ALARM_AckHi |
| AckLo | ALARM_AckLo |
| AreaDescription | ALARM_AreaDescription |
| ChangeDescription | ALARM_CHANGEDESCRIPTION |
| FreezeDisplay | ALARM_FreezeDispLAY |
| Page | ALARM_Page |
| PriorityFilter | ALARM_PRIORITYFilter |
| SelectedTag | ALARM_SelectedTag |
| SortItem | ALARM_SortItem |
| UnAckEm | ALARM_UnAckEm |
| UnAckHi | ALARM_UnAckHi |
| UnAckLo | ALARM_UnAckLO |
| UnitDescription | ALARM_UnitDescription |
| UnitID | ALARM_UnitID |
| UnitStatus | ALARM_UnitStatus |
| ViewFilter | ALARM_ViewFilter |

# Alarm Window Scripting Example 4

## Showing Unit Status Example

```
Sub OnPropertyChange(nProperty As Integer,nUnitNumber As Integer,vValue As Variant)

        On Error GoTo errorhandler
        Select Case nProperty
        Case ALARM_UNITSTATUS 'Unit Status
                Select Case nUnitNumber
                        Case 1
                                ShowUnitStatus UnitStatusRect1, vValue
(other case selections appear)
                End Select
        End Select
        End Sub

Sub ShowUnitStatus(UnitStatusRect As Object, status As Variant)
Select Case status
Case 1
        UnitStatusRect.Blink = false
        UnitStatusRect.Fill = false
(other case selections appear)
End Select
End Sub
```

## Example of Showing Unit Status

The following example uses the OnPropertyChange event to update the unit alarm status changes, it triggers the OnPropertyChange event. The script statements within the select..case logic call another subroutine called ShowUnitStatus. The object name of the rectangle representing the unit, UnitStatusRect# is transferred to ShowUnitStatus, along with the Unit Status stored in vValue.

```
Sub OnPropertyChange(nProperty As Integer,nUnitNumber As Integer,vValue As Variant)
On Error GoTo errorhandler
Select Case nProperty
       Case ALARM_UNITSTATUS 'Unit Status
              Select Case nUnitNumber
              Case 1
              ShowUnitStatus UnitStatusRect1, vValue
              Case 2
              ShowUnitStatus UnitStatusRect2, vValue
              Case 3
              ShowUnitStatus UnitStatusRect3, vValue
              (other case selections appear)
       End Select
End Select
End Sub
```

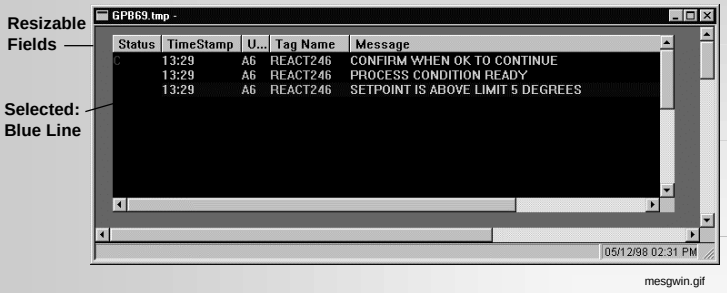
In the following subroutine, ShowUnitStatus, the rectangle is defined as a data type object. The Unit Status, passed as vValue is stored in status as a data type variant. That allows the subroutine to evaluate current unit alarm status.

```
Sub ShowUnitStatus(UnitStatusRect As Object, status As Variant)
Select Case status
Case 1
              UnitStatusRect.Blink = false
              UnitStatusRect.Fill = false
Case 2
              UnitStatusRect.Blink = false
              UnitStatusRect.Fill = True
              UnitStatusRect.fillcolor = lalmclr
(other case selections appear)
End Select
End Sub
```

# Message Window Appearance and Operation

**Shows same message data as US, but appears in GUS Display**

**Example Message Window appearance and operation**




mesgwin.gif

**Note: Additional operations require scripted objects**

## Background

The Message Window shows the same data as that displayed in a classic Universal Station or Native Window.

## Message Window Operation

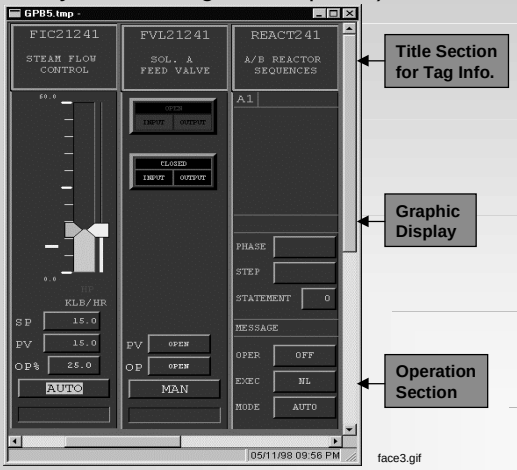
When one of the Messages on the Message Window control is selected, the background color of the selected line turns blue. The selection of a message is canceled when the selected message is acknowledged or another message is selected.

For operations such as acknowledging, you need to provide the Message Window with scripted objects.

# Faceplate Object: Introduction

- **Purpose: Operate from GUS Custom or Group Display**
- **Supports several data point types**

(e.g., Regulatory Control, Digital Composite)




## Purpose of Faceplate Object

The Faceplate control provides operators with a way to operate Data Points in the TPS System. Its functions are comparable to that of the US's slot shown in the Group Display. The Faceplate control displays several Parameters of each Data Point (for example, for Regulatory Control Data Point type, SP, PV, OP, Mode and etc.). Operators can change a parameter's value. The Faceplate control is mainly used in GUS Custom Displays and Group Displays.

In additional to the GUS displays, you can use the Faceplate control in any OCX container applications.

## Point Types Supported

NIM:PM/APM/HPM:

- Regulatory Control
- Regulatory PV
- Analog Input
- Analog Output
- Digital Input
- Digital Output
- Digital Composite
- Flag
- Numeric
- Timer
- Device Control
- Process Module Data Point

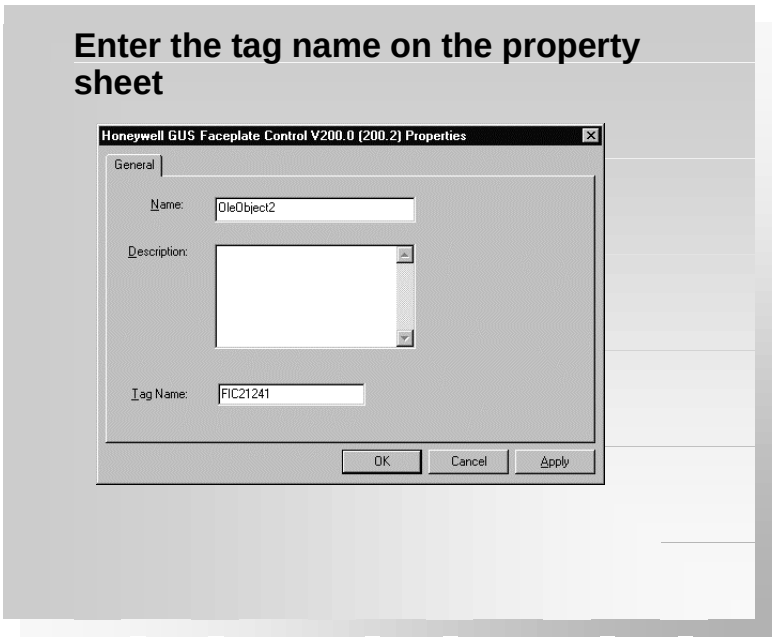
Application Module:

- Regulatory Control
- Flag
- Numeric
- Timer
- Switch

Data Hiway (HG):

- Regulatory Control
- Analog Composite
- Analog Input
- Analog Output
- Digital Input
- Digital Output
- Digital Composite
- Flag
- Counter
- Timer

# Faceplate Configuration



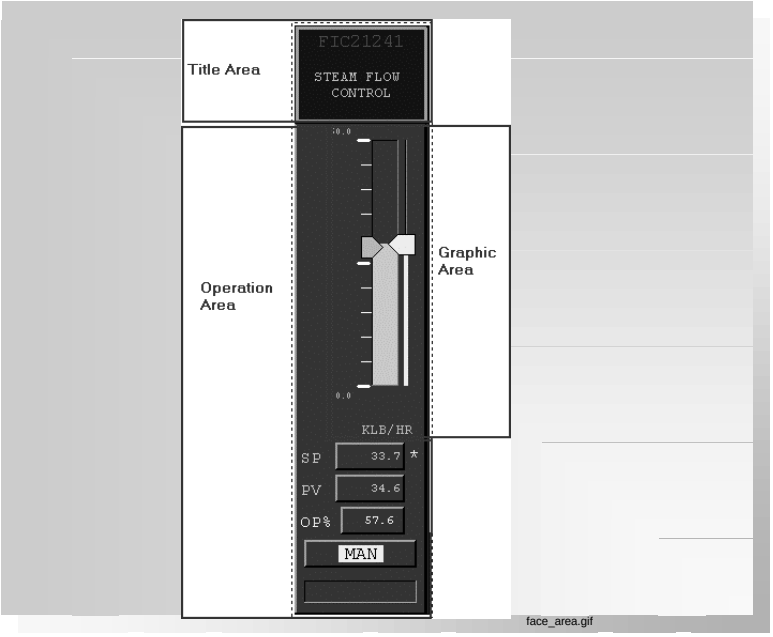

## Configuration of the Faceplate control

The Faceplate property sheet appears by double-clicking the Faceplate control or by choosing the Edit-Property command. The property sheet consists of only one page: General.

There is only one type of the Faceplate control and it supports various point types. Once the tag name, or the Data Point name, is provided either by property configuration or by scripting, the Faceplate determines the point type from the tag name.

NOTE: There is no need for an “LCN.” prefix, as shown in the above figure.

# Faceplate Appearance




## Appearance of the Faceplate

The Faceplate control consists of three areas: Title area, Graphic area and Operation area.

• Title:

Shows the identity of the Data Point. Also, it indicates that the Faceplate is being selected with its background color turning blue.

It also shows the following tagname information: Point Descriptor, Point Status.

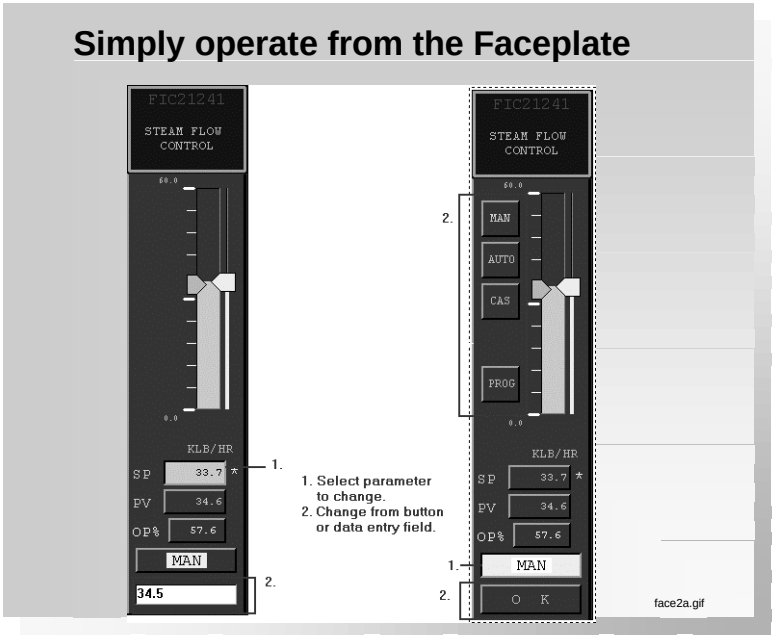
• Graphic:

The graphic part is where the values of the parameters are shown graphically. Usually, the analog values are shown as bar graphs and the digital values are shown as boxes.

• Operation:

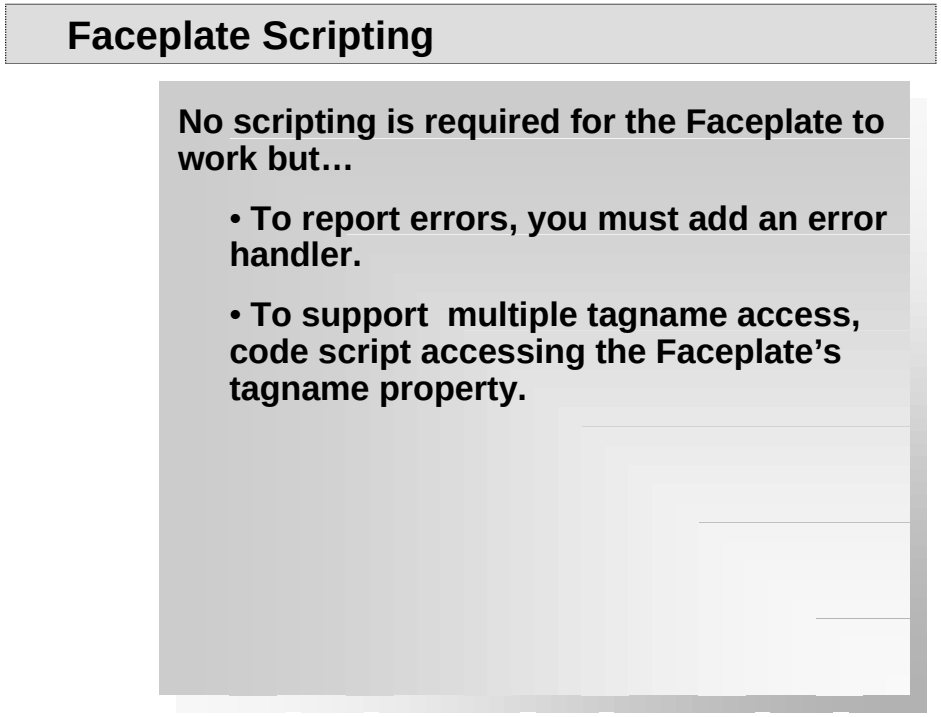
The operation part shows the various parameter values, indicators, and targets to change the parameter values. For example, parameters values are shown as buttons.

# Faceplate Operation




## Operation of the Faceplate

Select the parameter to change and make the entry from a data entry field or pushbutton.




## Scripting the Faceplate

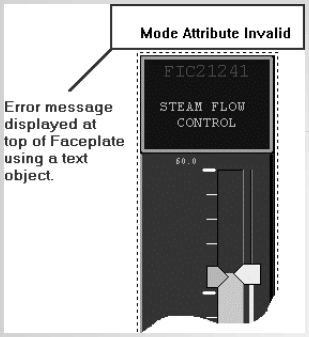
While a faceplate control requires no scripting at all for it to operate in a GUS display, there are two scenarios where scripting is required:

- The Faceplate control does not have default error handling; for it to report errors, you must add an error handler.

- You can make a single Faceplate control display whatever point type or tagname you need simply by coding script that accesses the Faceplate's tagname property.

## Faceplate Error Handling

- **Error Handling supports message prompts**
- **Requires text object typically positioned at top of Faceplate**

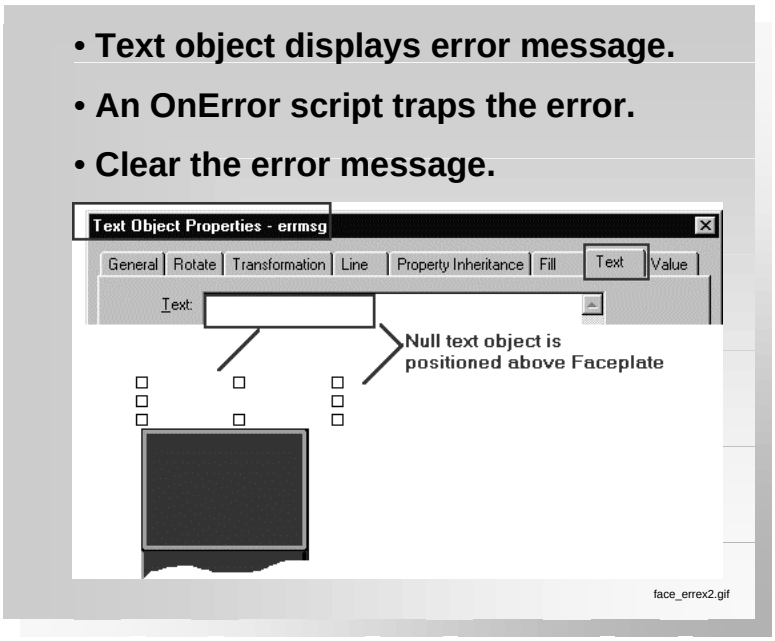



face_errex.gif

### Provide error handling

You should provide error handling because the displayed errors are not only runtime errors, but message prompts that an operator encounters when trying to make an invalid entry. For example, if an operator tries to change a setpoint and the tagname is in the wrong mode, then you would want to display that as an error message. For users familiar with the classic Universal Station, these errors are the messages that would appear in the prompt field when an operator attempts an invalid parameter change.

## Scripting Faceplate Error Handling




### Scripting Faceplate Error Handling

Although there is more than one approach to error handling, a typical approach to support error handling is the following:

- A text object displays the error message. Typically you would place the text object above the Faceplate control for operator convenience.
- An OnError script traps the error.
- A script clears the error message. The script can use the ButtonClicked event to clear the error message when the operator selects another Faceplate button.

The Faceplate control has an OnError event that you can script to trap the error. If a runtime error occurs, the message is stored as a parameter of that event. The message string is then assigned to the text object. In our example, the text object is named errmsg. The message itself is displayed above the Faceplate control.

```
Sub OnError(Message As String)
'assign error message to text object errmsg
      errmsg.text = message
End Sub
```

Clicking a different Faceplate button (for example, SP,OP) clears the error message text.

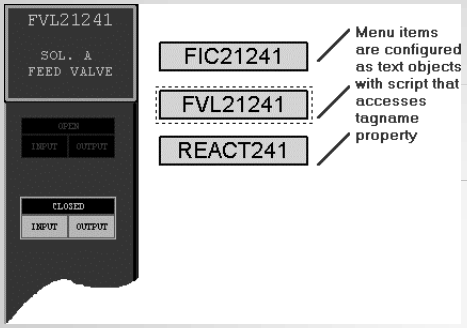
```
Sub ButtonClicked(Button As Integer)
      errmsg.text = ""  'clear out the errmsg object when
                  'another Faceplate button clicked
End Sub
```

## Scripting Faceplate for Tagnames

- **Access tagname property.**

```
'Button text object1
Sub OnLButtonClick()
     oleobject1.tagname = "FIC21241"
End Sub
```

- **Text objects can be used for menu.**

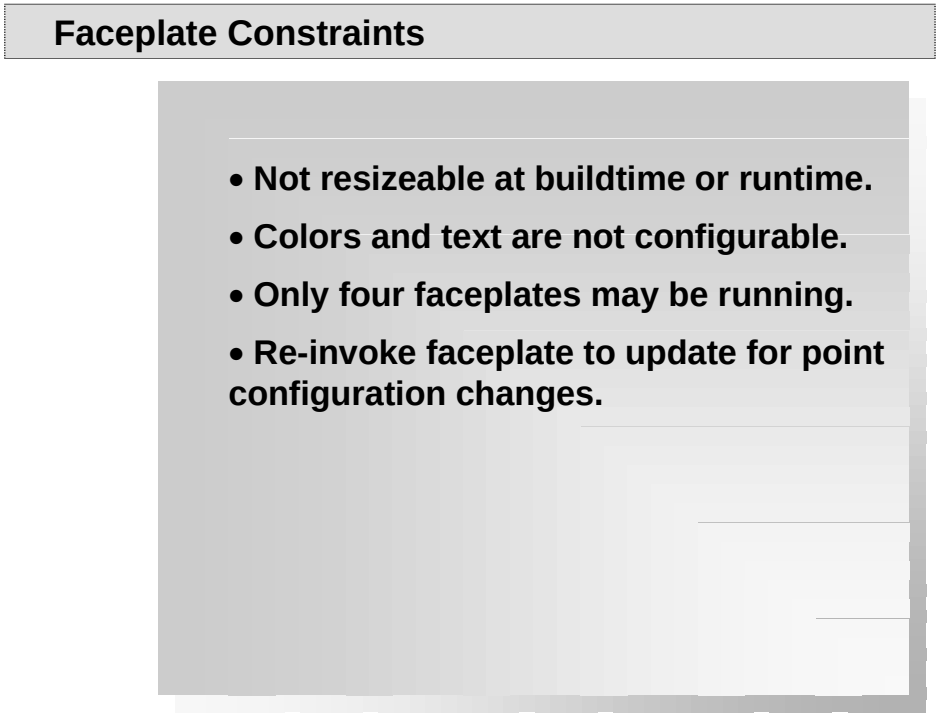



3tag.gif

### Scripting the Faceplate for More than One Tagname

A single faceplate control can display more than one tagname. The following script examples show that by accessing the tagname property of the Faceplate, you can change the Faceplate's tagname at runtime. The buttons can be text objects arranged to appear as a menu of tagname selections.

```
'Button text object1
Sub OnLButtonClick()
      oleobject1.tagname = "FIC21241"
End Sub

'Button text object 2
Sub OnLButtonClick()
      oleobject1.tagname = " FVL21241"
End Sub

'Button text object 3
Sub OnLButtonClick()
      oleobject1.tagname = " REACT241"
End Sub
```




## Constraints

The GUS Faceplate control has the following constraints.

- The faceplate control is not resizeable at buildtime or runtime.

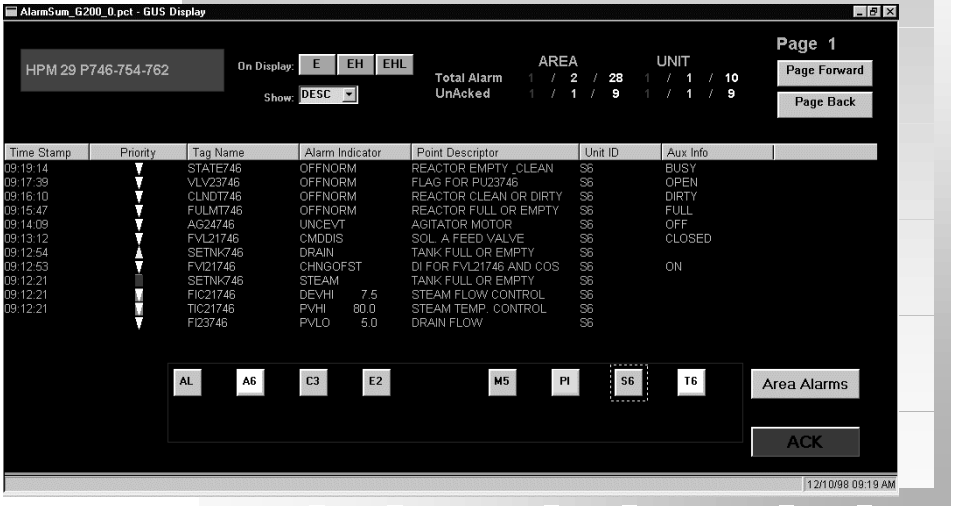
- The colors and text of the faceplate control are not user configurable from the property sheet or scripting.

- Only four faceplate controls may be running on a single GUS node. If the GUS Group Display is running, then only three faceplate controls may run. (The GUS Group display is an application that contains a series of Faceplate controls. The Group display provides functions similar to a Universal Station's Group display.)

- You must re-invoke the faceplate control for a point to update point changes caused when an engineer changes the point's configuration from another engineering type activity. Changes to parameters shown in the faceplate control that were caused by the engineering change will not be displayed until the faceplate is re-invoked for that point.

## Standard Displays: Alarm Summary

- **Alarm Summary is a .PCT File**
- **Contains Alarm Window Control**



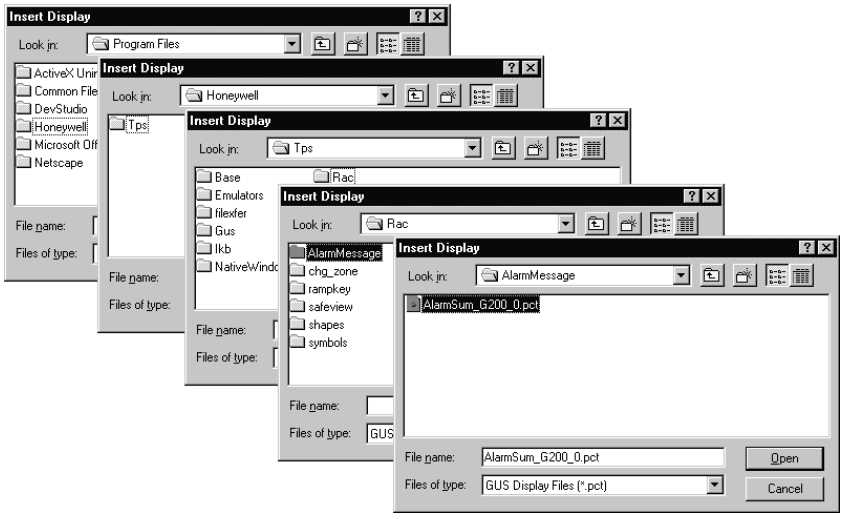

alm_sum_runr.gif

### Alarm Summary Display

The Alarm Summary Display is a .pct file. It contains the Alarm Window Control and objects which use the available properties and methods to show the alarm information or interact with the Alarm control .

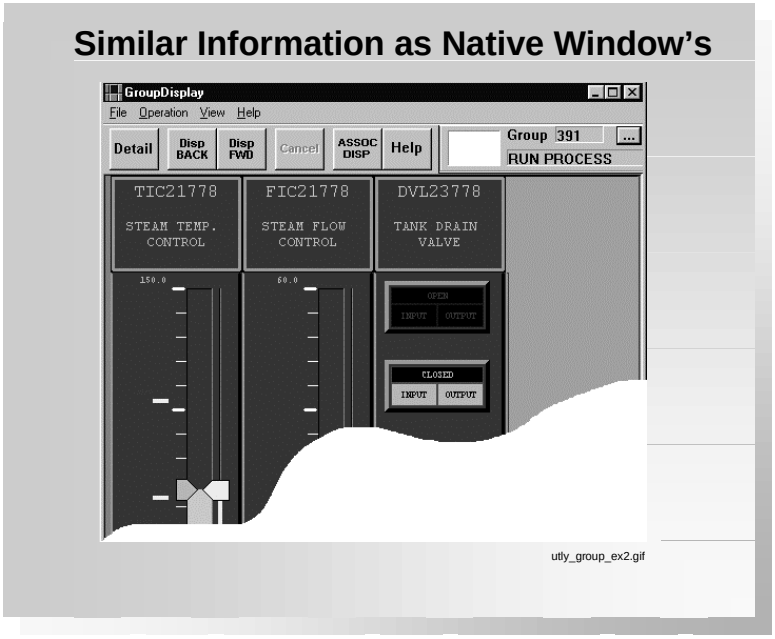
### Location of the Alarm Summary Display

The AlarmSum.pct file resides in the RAC folder.




alm_rac.gif

# Standard Display: Group Display

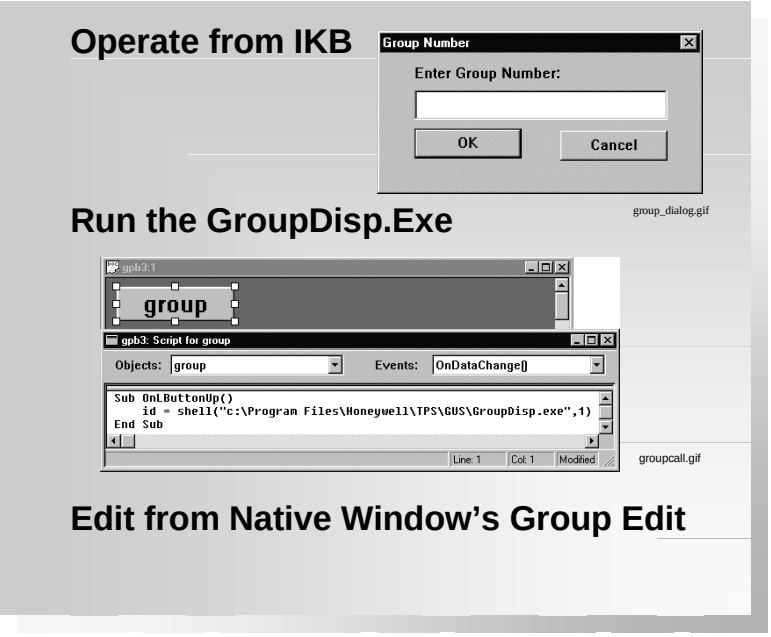



## Background

The Group Display is one of the standard displays provided in R200. Operators can monitor and manipulate operating parameters and modes through these displays. The TPS Network supports 400 groups and 50 process module groups; the same information is available in the Standard Group Display.

Each group is shown in a paged window, and parameters and status of up to 8 Data Points can be displayed in a group or page.

# Group Display Invocation




## Invoke Group Display

With a GUS display active, the Group Display can be executed by pressing the Group key on the standard or Integrated Keyboard. (Note: After enabling the Group Summary from the Configuration Utility, you may have to logoff and logon for the Group Number dialog to appear.)

(Or, you can double-click the icon of the Group Display or from the Explorer find the execution file (File pathname: Program Files\Honeywell\TPS\GUS**GroupDisp.exe**) and "run" it.)

## Select and Operate a Data Point

The Group Display consists of 8 Data Points represented by Faceplate control. No Data Point is selected as an initial state when a new display is called up. For operations on Data Points, you need to select a Data Point by clicking one of the Faceplates. The top portion, the title box, of the Faceplate then turns blue, indicating that changes are permitted.

Most of the operations are made for the selected Data Point using the Integrated Keyboard (IKB), or directly made for each Data Point using the Point Manipulation Keys (PMK).

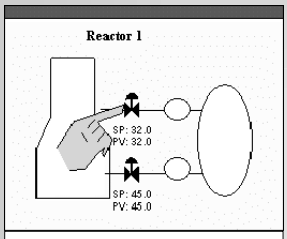
## Edit from Native Window Group Edit

The Group Display uses the current Area Database Configuration for that Group. It can be edited from the Native Window's Group Edit Display. (Choose Displays > System > System Menu > Group Edit Display from the Native Window.)

# Point Manipulation Key (PMK) Object

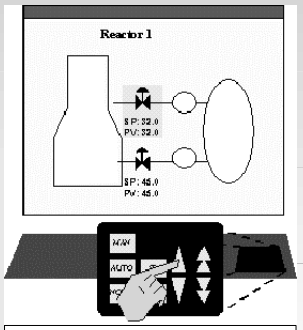
## Introduction - What is PMK?

- Built-in GUS scripting object
- Implements IKB point manipulation keys




**Operator selects point from GUS display**

pmkopr1.gif




**Operator's IKB accesses PMK object to change point's SP, OP, modes.**

pmkopr2.gif

..

## PMK: Overview

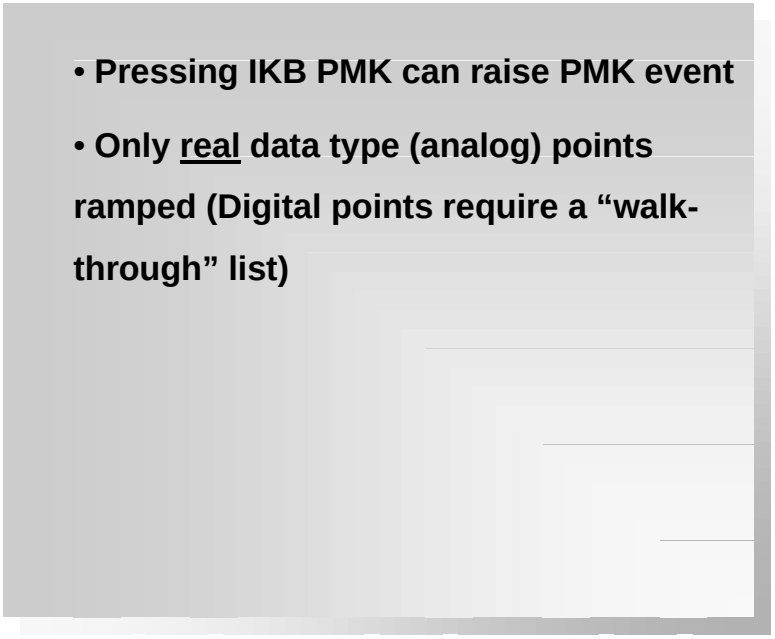
**What PMK provides your displays**

- Ability to ramp setpoint and outputs
- Can send IKB events to GUS display objects

**Example Applications**

- IKB access to embedded displays representing process elements (e.g., control valve)
- IKB access to custom change zone

## Raising PMK Events




Each time an operator presses a Point Manipulation Key (PMK) on the IKB, an equivalent PMK event can be raised. You can then script actions that occur whenever a key is pressed by the operator. These same PMK events can be emulated in scripting (i.e., use a KEY method) on the PMK object.

### What is raised/lowered

Not all data points can be raised/lowered by the PMK object. Only points that are of type real can be raised. In other words, digital points, flag points, switches and logic points cannot be raised/lowered by the PMK object. Regulatory, Analog, Process Module and Counter points can be raised/lowered by the PMK object if they have at least one of the following parameter types: OP, SP, PVTV and AVTV.

In other words, when the OUT button is pressed, all real data points raise/lower the OP value.

Refer to the *Display Authoring Tutorial* for an example of using PMK with a Digital Point.

## What the PMK Object Does

**GUS scripts interact with PMK through**

- methods
  (e.g., `PMK.Key PMK_SP`)
- properties
  (e.g., `Set PMK.Entity = LCN.A100`)

**Only <u>one</u> PMK object per GUS display**

- Global in scope
- Script must protect from unintended ramping

### What it does

The PMK is a built-in GUS **<u>scripting</u>** object that implements the point manipulation keys found on the IKB and other Honeywell engineering keyboards. GUS scripts interact directly with the PMK Object through the invocation of methods (e.g., `PMK.Clear`) or by setting properties.

Since the PMK Object is a built-in object in GUS displays it does not need to be declared (or DIMed in GUS Basic) by the user. There exists only <u>one</u> PMK object per GUS display, and it is global in scope. For example, an embedded display and main display both use the same PMK object without modifications to scripts.

### How it works

Once the PMK has a registered entity and a registered parameter, all point manipulation keys (SP, OP, MODE) from the IKB will result in changes to the value of the registered entity if the AutoWrite property is set to TRUE (default is True). If the AutoWrite property is set to FALSE, changes will not be made to the registered data point. In either case, scriptable events will be fired when PMK keys are pressed.

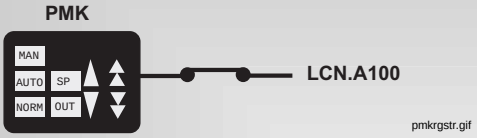
Point manipulation keys can be emulated for PC keyboard users through GUS scripts that invoke methods on the PMK object (e.g., `PMK.Raise`). Primitive objects in Display Builder can script responses to PMK events raised when point manipulation keys are pressed on the keyboard or when a primitive object is selected in the graphic. For example, a ramp key could be graphically displayed in your picture, and by selecting this ramp-key object you could accomplish the same results as selecting the ramp key on your IKB.

## How to Get the PMK Object to Work

**Simple script example illustrates concepts**

**All you need to do is**

1. Register the entity and the parameter
   e.g., `Set PMK.Entity = LCN.A100`




2. Use a method to access values
   e.g., `PMK.Key PMK_SP`
3. Deactivate the PMK when you are done
   e.g., `PMK.CLEAR`


### Registering the PMK Object

Once an entity has been registered with the PMK scripting object, the user can specify a parameter (SP or OP) to register and then can ramp that data point through the keyboard. The following script shows one method of registering the PMK object with the SP parameter.

This script also protects the user from ramping the wrong value.

```
Sub OnGotFocus()
Set PMK.Entity = LCN.A100     'registers the point
PMK.Key PMK_SP                'registers the parameter
End Sub
Sub OnLostFocus()
PMK.Clear                     'clears the point from the PMK object
End Sub
```

The PMK unregisters the entity A100 when the object (text) loses focus. So if the operator selects another object (text2) and hits the Raise key nothing will happen to A100 because the PMK.Clear function removes the connection between the PMK object and the point A100.

## Which Events can Be Used with PMK?

**Remember: one PMK per display and it is global**

- Use an OnGotFocus Event to register and access
- Use an OnLostFocus Event to deactivate PMK
- Use an OnDataChange to update display

- Use an OnGotFocus Event to register and access

e.g.,

```
Sub OnGotFocus()
    Set PMK.ENTITY = lcn.fic21241  'registration of data point
    PMK.KEY PMK_SP       'SP is the initial registered parameter
End Sub
```

- Use an OnLostFocus Event to deactivate PMK

e.g.,

```
Sub OnLostFocus()
    PMK.CLEAR            'Deactivate PMK connection to point
End Sub
```

- Use an OnDataChange to update display

e.g.,

```
Sub OnDataChange()
    me.text = LCN.FIC21241.SP
End Sub
```

It is recommended that engineers script OnEnterFocus events to register the PMK for their embedded picture and OnExitFocus to clear the PMK for their embedded pictures.

## What About PMK Error Handling?

**To enable Error Handling:**

1. Set the PMK.ErrorHandler to this object
   e.g.,
   ```
   Set PMK.ErrorHandler = me
   'enable handling
   ```
2. Trap PMK Error Events
   e.g.,
   ```
   Sub OnPMKError (ErrCode As
   Long, ErrString As String)
   ```

**Note: PMK.ErrorHandler Deactivated Upon Clear**

### To enable Error Handling:

1. Set the PMK.ErrorHandler to this object
   e.g.,
   ```
   Sub OnGotFocus()
       Set PMK.ErrorHandler = me        'enable handling
       Set PMK.Entity = LCN.FIC21241
       PMK.KEY PMK_SP
   End Sub
   ```

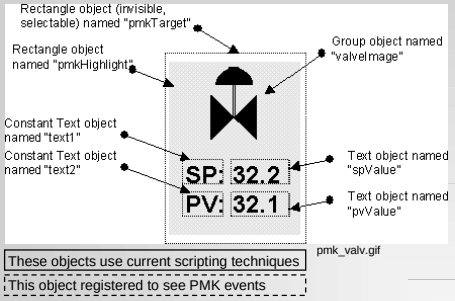
2. Trap PMK Error Events
   e.g.,
   ```
   Sub OnPMKError (ErrCode As Long, ErrString As String)
       msgbox errstring
   End Sub
   ```

Note: PMK.ErrorHandler is Deactivated Upon Clear!

# Example Application: Valve Embedded Display

**Embedded display built in traditional way**

- Define display.params of type Entity (e.g., “point”)
- Name your objects for ease of use
- Code PMK script, for convenience, on one object




**To use PMK, all you need to do is**

- Register the PMK to Display Parameter (display.params.point)
- Register object to handle PMK events
- Have display object show the value
- Deactivate PMK when done

## Embedded display built in traditional way

- Define display.params in the traditional way
  e.g., “point”
- Name your objects for ease of use
- Code PMK script, for convenience, on one object

## To Use PMK, all you need to do is

- Set the PMK to Display Parameter (display.params)
  e.g.,

```
Set PMK.Entity = Display.params.point
```

- Register object to handle PMK events
  e.g.,

```
Set PMK.EventHandler = SP value
```

- Have display object display value
  e.g.,

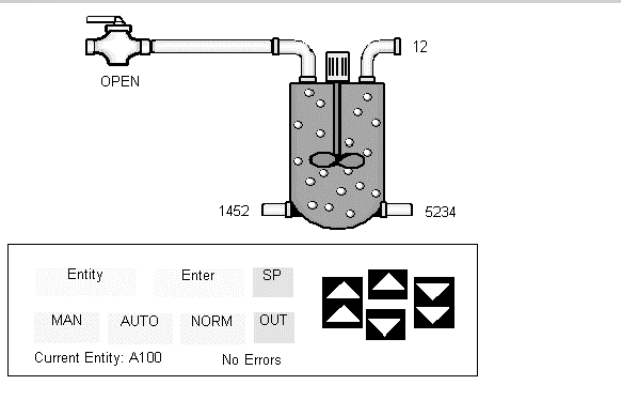
```
'text object shows value
me.text = display.params.point
```

- Deactivate PMK when done

```
PMK.clear
```

## Example Application: Change Zone Keypad

- **An embedded display or screen object can represent the IKB**




- **Acts like virtual keyboard**

The above picture contains the following objects:

1. An embedded display named emdKeyPad, which represents the keypad object at the bottom of the display.

2. An embedded display named emdValve, which represents the valve in the upper-left corner of the display.

3. A bitmap object representing a boiler.

4. Three text objects named text1, text2, and text3 respectively. These text objects are around the boiler and have values of 1452, 5234, and 12 respectively.

The embedded keyboard acts like a virtual keyboard for operators who might not have an IKB. Choosing the Raise button on the embedded display will be equivalent to choosing the raise key on the IKB keyboard.

After selecting one of the text objects around the boiler followed by a SP/OUT key press the operator will be able to raise/lower the SP or OP parameter for the text object's data point.

Additional display building and scripting details are in the reference document.