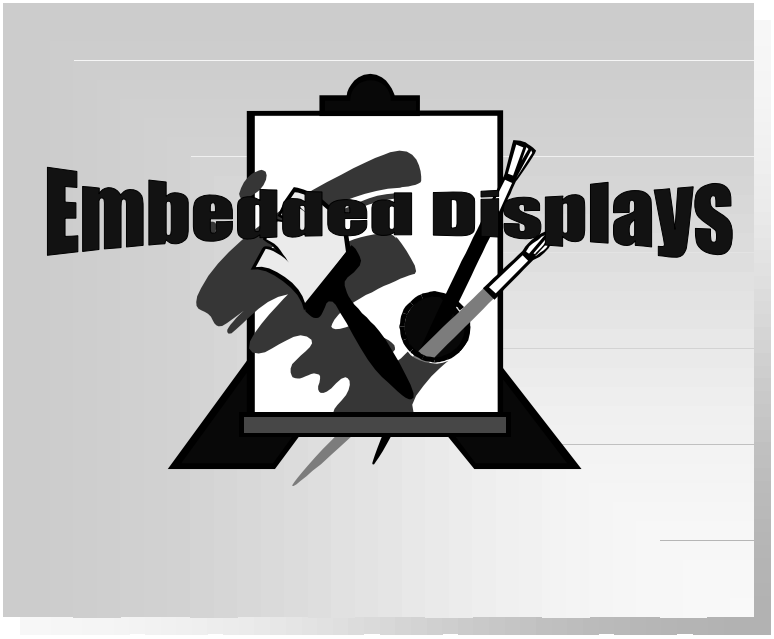
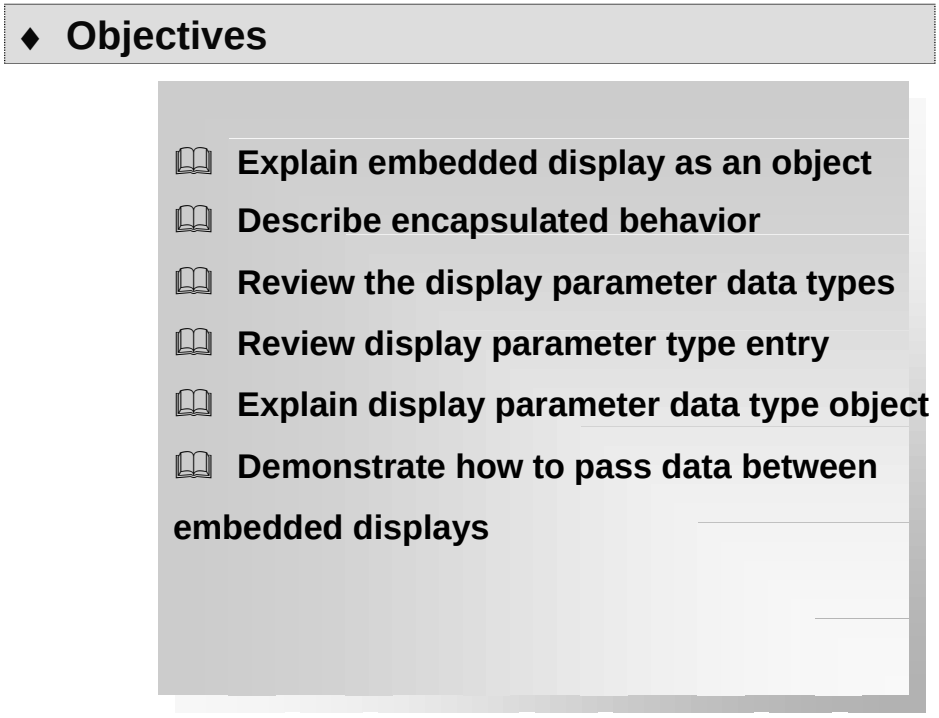
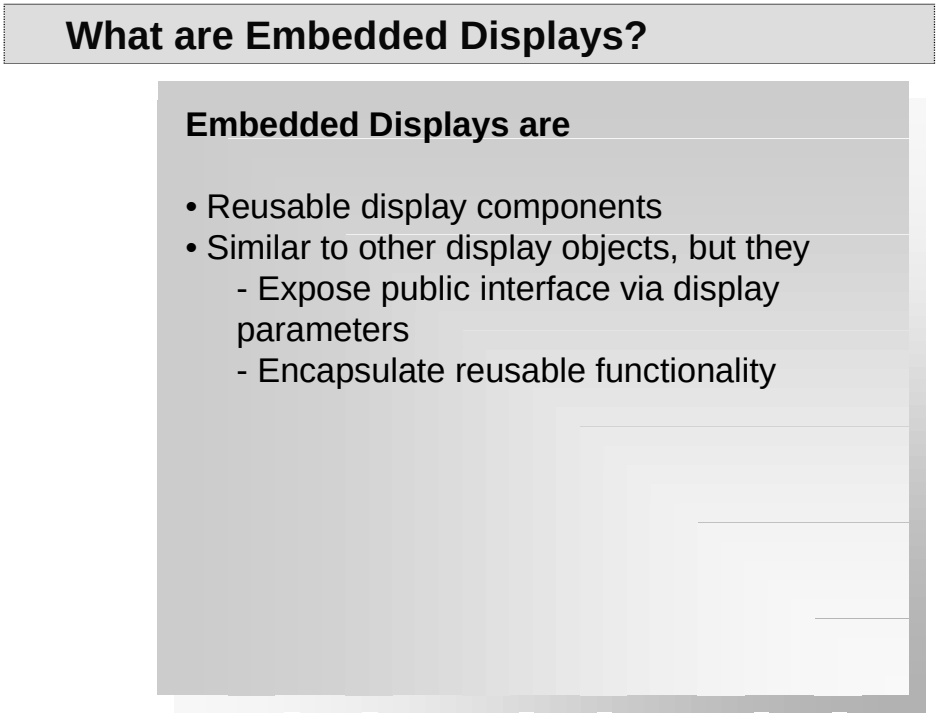

Embedded displays



Objectives

At the end of this module you will be able to script performant embedded displays that satisfy defined functional requirements, reusability, and minimal rework. To accomplish that goal, the following objectives are covered in this module

- Explain the embedded display as a display object
- Describe what is meant by encapsulated behavior for an embedded display
- Review the display parameter data types
- Review how to enter display parameter types
- Explain in more detail the display parameter “object” data type
    - Give an example of how the object data type could be used
- Demonstrate how to pass data between embedded displays.
    - Describe how display parameters can be used to communicate between a display object and other display objects
- Examine the tradeoffs of using embedded displays within embedded displays



Embedded displays are reusable display components

Embedded displays enable you to build reusable display components. Some common examples of embedded displays are:

- process elements that you want to behave in a similar manner
- change zones (standard and custom)
- object libraries.

Embedded displays can also be used to provide an operator with a user-friendly and consistent interface. For example, the technique of target management (which will be explored later) is demonstrated with embedded displays.

Embedded displays are like user-defined objects

The embedded display feature of GUS Displays enables you 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 (i.e., display.params).

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 functionality can be considered as "encapsulation". Encapsulation makes the reusability of an embedded display possible.

Encapsulation means that certain scope rules apply - what can and cannot be accessed.

Embedded Display Scope Rules

Scope rules are the following:
• Embedded display components are not visible to the containing display.
• Containing display components are not visible to the embedded display.
• Can pass data back and forth using display parameters and DDB values

Main display script execution
• Main display scripts of an embedded display are executed in the context of the embedded display.

Script thread usage:
• Scripts share main threads

Scope rules

The scope of an embedded display's object names, public variables, or public functions is confined to the immediately containing display. This means that:
•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.
• In order to pass data between a display and embedded display (e.g, a tagname to a change zone) or embedded display to another embedded display, you have to use display parameters (display.params) or the display database (DDB) , for example "dispdb.ent01."

What happens to the main display script when display is embedded

Although a display can have a main display script, the main display script is associated with the display itself, rather than with a particular display object. When that 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.

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.

Embedded Display Replacement

Can do a global replace of an embedded display with a new version of the embedded display

Once a display is inserted/embedded into a container display, the embedded display becomes an object of the container display. If a change is made to the "original" version of the display after it has been embedded into another display, the embedded display does not automatically update. You must do a "Replace Embedded Display" if you want the newest version of the display.

Note: If you have 5 instances of an embedded display inserted in a container display and you do a replace command, ALL 5 instances of the embedded display are replaced.

Display Parameters “Rules of the Road”

- **Rules of the Road:**
  - All that the main display and all the other objects in the main display know about an embedded display is what display parameters (i.e., display.params) it has.
  - All you can change on an embedded display are its display parameters
- **Overview of Parameter Usage**

Rules of the Road:

•All the main display and all the other objects in the main display know about an embedded display is what display parameters (i.e., display.params) it has.
•And that is all you can change on an embedded display -- its display.parameters.

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).
•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) support building of embedded displays that manipulate LCN points 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 that lets you manipulate LCN points.
• Some kinds of display parameters (Value kind:Integer, string, etc) can also be used to store data that is accessible throughout the display. They can be used as an alternative to public variables and some DDB values.

Embedded Display Parameter Types

Three Types of Parameters:

- Reference-type Parameters (Entity and Variable)
  - Used to Access LCN variables and entities
- Value-type Parameters (e.g., Single, Double, Strings, Long and Short Integers)
  - Used to set static values
  - Used to pass values between the containing display and the embedded display
- Inline Parameters
  - Used to parameterize collector references (i.e, ACKSTAT)
  - Substitution/conversion at validation time

Three Types of Parameters:

•Reference-type Parameters (Entity and Variable)- Used to Access LCN entities and variables (e.g., LCN.FIC100, LCN.FIC100.pv)
•Value-type Parameters (e.g., Single, Double, Strings, Long and Short Integers)- Used to set static values and to pass values between the containing display and the embedded display
•Inline Parameter- Used to parameterize collector references (i.e, ACKSTAT). Text substitution/ conversion at validation time (I.e., through the use of \p and \pe)

Overview of reference-type parameters

Parameters whose data type is either variable or entity are specialized for access to LCN variables and entities. The reference is set up in the Enter Parameters dialog. 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 a entity-type display parameter and you can reference a TDC (TPS) variable (such as a point.parameter) indirectly through a variable-type display parameter.

Overview of value-type parameters

Parameters whose data 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, these kinds of parameters can be used to pass data between the containing and embedded display.

Overview of Inline parameters

Parameters whose data type 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, at insertion time, to enter the BasicScript name form for LCN references on the Enter Parameters dialog (e.g. LCN.Tagname). When you script the embedded display, use the \pe operator to parameterize collector references; this eliminates the LCN prefix of the tagname and does a direct text substitution. Collectors do not recognize the LCN.tagname syntax -- they need to see only the tagname.

Accessing Inline Types

Inline parameters are accessed by using
- the parameter name in a script (typical approach),
- expression field of a dynamic page,
- or in the expression of an enter parameters dialog.

Rules of the Road
- Do not prefix with 'display.params'.
- Only accessed in the display that defines them.
- Two operators '\p(paramname)' or '\pe(paramname)'

Inline parameters are accessed by using

- the parameter name in a script (typical approach),
- expression field of a dynamic page on a properties dialog,
- or in the expression of an enter parameters dialog.

Rules of the Road

•Do not prefix the name of an Inline display parameter with 'display.params'.
•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.
•Two special operators are provided. 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.

The \pe string substitution operator is useful when you parameterize a collector reference. In general, you should design your embedded display such that you have to enter the LCN reference (i.e, LCN.FIC100) on the Enter Parameters dialog. When you script the embedded display, use the \pe operator to parameterize collector references. Collectors do not recognize the LCN.tagname format; the \pe operator eliminates the LCN prefix from the tagname. The following script example has an inline parameter named TempValue. This shows a collector reference and a non collector reference to the same display parameter. In this example, you would enter a name such as 'LCN.FIC100' in the enter parameters dialog when embedding this display.

```
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 can also 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.

Enter Parameters: Initial vs OnDataChange

Enter Parameters Dialog: Initial Value Expression vs OnDataChange Expression

- Always use OnDataChange for an LCN data reference even if it is a value-type reference
  - -For static LCN values, set the collection rate of the parameter to 0.
- Use Initial Value Expression for static values (e.g., “Tank 1”)

Enter Parameters Dialog: Initial Value Expression vs OnDataChange Expression

- **Always use OnDataChange for an LCN data reference even if it is a value-type reference**
  - **-For static LCN values, set the collection rate of the parameter to 0.**

Background: 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 script. This means that if you enter 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 at display startup results in an execution of the OnDataChange expression.

- **Use Initial Value Expression for static values (e.g., “Tank 1”)**

Background: Use the initial value expression when the data assigned to the parameter is static. This may be useful for display parameters used to show a label in the embedded display. (Note that if you put any LCN object references in this expression, they are immediate-type accesses (reads/writes) that are evaluated once at display startup.)

About that Enter Parameters dialog

The Enter Parameters dialog allows you to specify a value for a parameter. 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.

Inheritance

What is embedded display "inheritance" ?
Answer: Simply stated, the value of a property is "sent" from the containing object.

Advantage:
Increases the re-usability of an embedded display.

Inheritance binding:
Properties are bound by the *user* of the embedded display, not the *author* of the embedded display.

Purpose:
The embedded display *user* specifies the property values, which means an embedded display can be used in different ways.

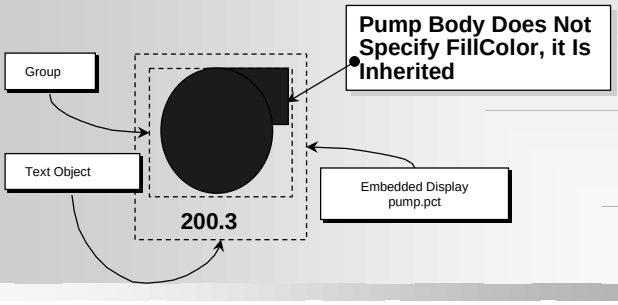
When an embedded display is created, the "author" of the display must indicate whether the objects in the display will be ***allowed*** to inherit different property characteristics. However, in order for this to actually happen, when an "end-user" inserts the embedded display into a parent or container display, that person must also enable the inheritance functionality. In this way, the user can specify unique property values for each instance of the embedded display that is inserted.

Example : Pump.pct

Example shows embedded display, Pump.pct

•To maximize re-use, the *author* of Pump.pct *permitted* the FillColor of the pump body to be inherited.

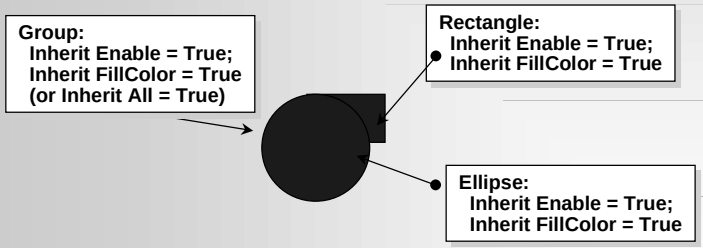
•At build time, the pump initially assumes the current default color of the display.



Example: Building Pump.pct

When the Pump body was built, the *<u>author</u>*

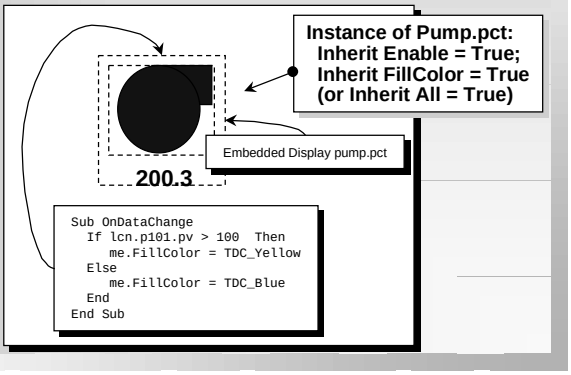
• set the Inherit Enable & Inherit Fill Color for both the ellipse and the rectangle to true. This *allows* the FillColor of the objects to be "inherited" from their container (the group), and
• grouped the objects and set the Enable & Inherit Fill Color for the group to true.



Example: Using Pump.pct at Site A

A *user* of Pump.pct at Site A inserts the pump and
• sets the Inherit Enabled for the embedded display to true, and
• scripts a change to the FillColor property of this embedded display instance.

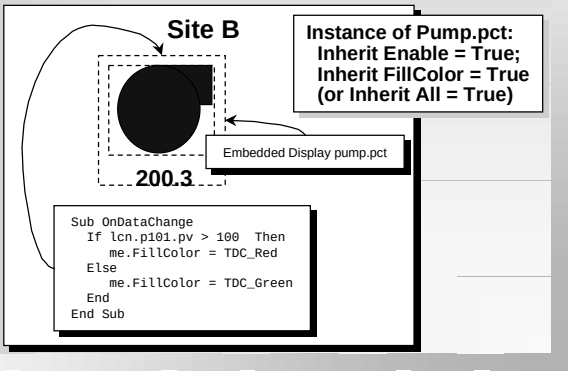
When the script associated with the embedded display changes the FillColor property for the Embedded Display, the FillColor is inherited by the group, and by the rectangle and ellipse



Example: Using Pump.pct at Site B

A *user* of Pump.pct at Site B inserts the pump and
• sets the Inherit Enabled for the embedded display to true, and
• scripts a change to the FillColor property of the embedded display instance.

When the script associated with the embedded display changes the FillColor property for the Embedded Display, the FillColor is inherited by the group, and by the rectangle and ellipse



Performance Tips for Embedded Displays

1. If embedded displays are used frequently, then attempt to “conserve” objects, by:
   - Using the fill and line properties of a text object; it is sometimes possible to remove extra objects from an embedded display.
   - Within frequently used embedded displays attempt to “conserve” OnDataChange scripts. (NOTE: small performance improvement)

2. Replace embedded displays that have no parameters with groups when possible (instead of inserting, do a copy of the objects into the display)

Display Authoring Tutorial Guidelines

“Conserve” objects
- Fill and line of text object instead of rectangles
- Reduces number of objects if many instances

Use a display parameter of object data type
- Used to write to another embedded display
- Direct write is fast
- Immediately raises OnDataChange event

Display Database (DDB) pass data:
- Used to write to another embedded display
- Not as performant as object data type

(the following excerpt is from the Display Builder Authoring Tutorial, topic: Embedded Displays)

“Conserve” objects within frequently used embedded displays

Using the fill and line properties of a text object, it is sometimes possible to remove extra rectangles from an embedded display. This change is minor in the embedded display; however, if a display uses many instances of that embedded display, the total number of objects is reduced significantly, which can improve performance.

Use a display parameter of data type object to read/write data between embedded displays if possible

Using the object parameter to write directly to another embedded display’s parameters is fast. This also causes the immediate firing of the OnDataChange scripts when a display parameter changes.

Data can also be read and/or written between embedded displays using the Display Data Base. This is a less performant method because writing data to a DDB item means an out-of-process write to the DDB located in the HOPC Server.

Data Type Object: Concept

Background

- The object data type is like a variable.
- Usually designates another object.
- For now, think of the object data type as “owning or controlling” the other object; in this case, another embedded display object.

Data Type Object: Concept Detail

Steps to using object data type

1. **In Picture1: Define a display parameter as object data type. Remember the object name for scripting (e.g., "controlee")**

2. **In Picture1: Code script that writes to the object (e.g., display.params.controlee.params.visible = true)**

3. **In Picture1: When embedding Picture1, name another embedded display (Picture2) as the object to be read/written to when prompted for object name. For example, Picture2 will be named "Box1"**

4. Picture2: After embedding Picture2, rename the embedded display to agree with the entry name you made in step 3 above. For example, Picture2 will be renamed "Box1" from its Properties General Tab.

(the following example is from the Display Builder Authoring Tutorial, topic: Embedded Displays)

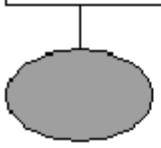
Background:

The goal of this example is to illustrate the concept of passing data between embedded displays using the object data type. After completing steps 1 through 3, you will have identified which object you want to control, but no "connection" or "link" to the other object is made until step 4.

Steps 1, 2, 3 Detail:

In this example, the display author defined an object data type and gave it a name of "controlee." The display author then codes script (see example below) that references the object's display parameter name, which completes step 1 and 2. When embedding this display, the display builder enters the name of the object to be controlled in the display from the Enter Parameters dialog. In this example, the controlee's name is entered as "Box1", which completes step 3.

Display Parameter	Type	Value	
controlee	object	Box1	Note: This value is written at buildtime.



```
OnLButtonClick()
   display.params.controlee.params.visible = TRUE
End Sub

OnRButtonClick()
   display.params.controlee.params.visible = FALSE
End Sub
```

Data Type Object: Concept Detail, continued

Steps to using data type object

1. In Picture1: You defined a data type object as a display parameter. Remember object name for scripting (e.g., “controlee”)

2. In Picture1: You coded script that writes to the object (e.g., display.params.controlee.params.visible = true)

3. In Picture1: When embedding Picture1, you named another embedded display (Picture2) as the object to be read/written to when prompted for object name. For example, Picture2 will be named “Box1”

4. **Now for Picture2: When embedding Picture2, rename the embedded display to agree with the entry you made in step 3 above. For example, Picture2 will be renamed “Box1” from its Properties General Tab.**

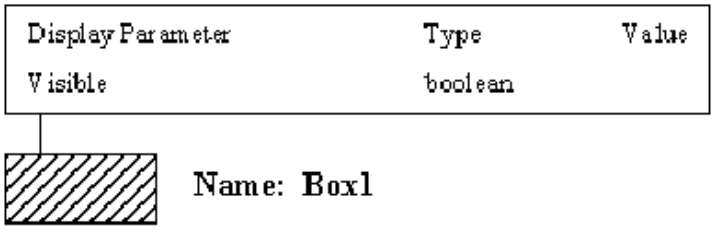
(the following example is from the Display Builder Authoring Tutorial, topic: Embedded Displays)

Step 4 Detail:

When embedding the display to be read from or written to, the display author renames the embedded picture to agree with object name entered in step 3. In this example, the embedded display was renamed to “Box1” from the Properties General tab, which completes step 4.

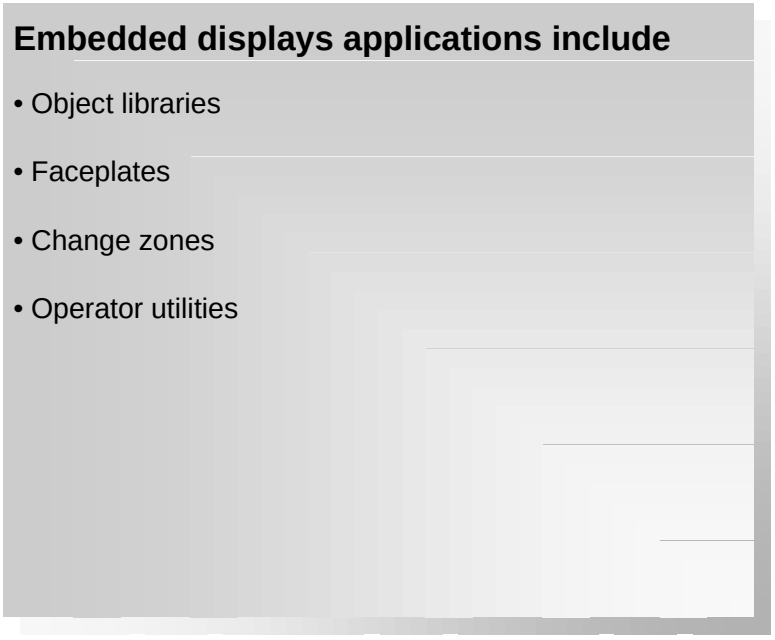
Display results:

The first embedded picture, the controller, can now write directly to the controlee’s display parameter “visible” on mouse clicks during runtime. When the first embedded picture writes to the controlee’s display parameter “visible”, this immediately fires the OnDataChange event on the “controlee” (the second embedded picture.




```
Sub OnDataChange()
    if display.params.visible = TRUE then
        me.fillcolor = TDC_CYAN
    else
        me.fillcolor = makecolor (128,128,128)
    end if
End Sub
```

Application Examples



Embedded displays applications include

The applications for embedded displays can be categorized in different ways. Here are just a few examples:

•Object libraries - these are displays that represent commonly used process elements.

• Faceplates - similar to change zones -- these often have graphical representations of values.

• Change zones - used to interact with the process and make changes.

• Operator utilities - these provide a friendlier interface to the operator. An example of an operator utility is a target manager or operator control panel.

Application Example: Target Manager Utility

Background:

- Problem: How to know which target is selected
- Goal: Show an operator the current target selection
- Solution: Manage target highlighting

One approach to target management

- Show highlighting when target is selected; show other targets in a non-selected highlight.
- Use object data type to read/write to all targets
  - Resets previous selection to non highlighted
  - Highlights new selection
  - Can highlight based on change zone entity

The following pages provide greater detail on this target management technique. There is also a hands-on Target Manager lab in the Change Zone section of your course material.

Application Example: Target Manager, continued

Summary:

- Embedded displays use object data type to R/W
- Embed one target_manager pct and as many selected_target pcts as needed.

Design of selected_target pct:

- Selected_target pct writes to target_manager pct that it has been selected (OnLButtonClick)
- Selected_target pct writes to target_manager pct its display name (so that target_manager "knows" which object to control) (OnLButtonClick)
- Selected_target pct writes to $cz_enty an entity name (OnLButtonClick)
- Selected_target pct monitors changes to selection flag (OnDataChange)

Example Selected_Target Script and Display Parameters

Display Parameter: Selected
Initial Value: false
Type: Boolean

Display Parameter: TargetMgr
Data Change Value: TM [note: this is the re-name of the Target Manager embedded picture]
Type: Object

[Script for the selected target]

```
Sub OnLButtonClick()
    Set display.params.targetmgr.params.newtarget = display
    dispdb.[$cz_enty] .external = "f2014" 'this tagname is passed to the change zone

    'PUT additional code here

End Sub

Sub OnDataChange ( )
    on error goto err_hdlr

    'PUT additional code here or after the code for the target behavior

    If display.params.selected = TRUE Then 'TargetMgr writes to Selected parameter
        me.fillcolor = tdc_blue              'this is the selected target behavior
    else
        me.fillcolor = tdc_half_white        'this is the non-selected target behavior
    end if
    exit Sub
err_hdlr:
End Sub
```

Application Example: Target Manager, continued

Summary:

• Embedded displays use object data type to R/W
• Embed one target_manager pct and as many selected_target pcts as needed.

Design of target_manager pct:

• Target_manager accesses selected_target through object called NewTarget.
•Target_manager creates another object in script, oldtarget, to handle reset to non-selected state.
• Target_manager pct writes back to a flag in selected_target pct that it is currently selected
• Target_manager pct resets any other selected_target pct

Example Target Management Script and Display Parameters

[Note: Once the Target Manager pct has been embedded, rename it from "EmbeddedPicture#" to **TM** in order to pass object data]
Parameter: NewTarget (the selected target will write its display name to the TM at runtime)
Type: Object

'Script for target manager

```
public oldtarget as object      'used to reset a target to non-selected color
Sub OnDataChange()
  on error goto err_hdlr
  if display.params.newtarget is nothing then          'no selected target to manage
    exit Sub
  end if
    if dispdb.[$cz_enty].[name].status = HOPC_CONFIGURATION_ERROR then
                    'handles "clear" action from changezone
      display.params.newtarget.params.selected = FALSE          'makes selected_target flag false
    else
        if oldtarget is nothing then          'no target to reset
        oldtarget.params.selected = FALSE          'resets "old" target to not selected
       end if
     set oldtarget = display.params.newtarget          'puts selected target into "old" target
     display.params.newtarget.params.selected = TRUE          'sets the "new" target to selected
     end  if
     exit Sub
err_hdlr:
End Sub

Sub OnDisplayShutdown()
                    set oldtarget = nothing          'code needed to support product
End Sub
```

Application Example: Object Libraries

Background:

Create your own process elements that provide operators a consistent user interface

Example application: Pump

- Pump changes color, blinks, visible based upon process conditions.
- Uses inline parameters for entity

Scripting examples provided in handouts

Embedded display pump: Code fragments
[note: refer to handouts for sample scripts, example below is fragment]

```
Global PC as Long                    'Store the Pump Color in PC
                    'This routine changes the color of the pump inside.
                    'Defining variable INSIDE simplifies multiple uses of this routine.
Sub Change_Color(INSIDE as Long)
                    Polygon1.fillcolor = INSIDE
                    Polygon2.fillcolor = INSIDE
                    Ellipse1.fillcolor = INSIDE
End Sub

Sub OnDataChange()
                    On error Goto ODC_error
                    PV = Tag_name.pv        'Defined earlier as inline parameter
{code  to set blinking, colors, etc in handout}
' Run routine to change colors
If  PV="STOP" Then
                    PC=tdc_red
  Else
                    PC=tdc_green
  End  If
                        call Change_Color(PC)
                    Exit Sub
                    ODC_error:
                        call Change_Color(tdc_yellow)
End Sub
```

Application Example: Faceplates

Background:

Create your own faceplates that provide operators only the information they need for operations

Example application: Digital controller

- Digital controller faceplate for digital composite type points.
- Uses object data type to pass data, including alarm status
- Only need to insert faceplate once, used by all digital composite points

Details in the following lab exercises