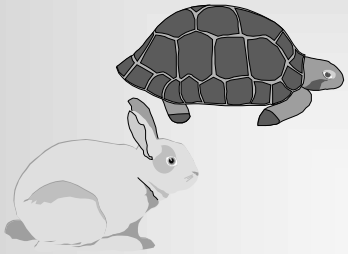
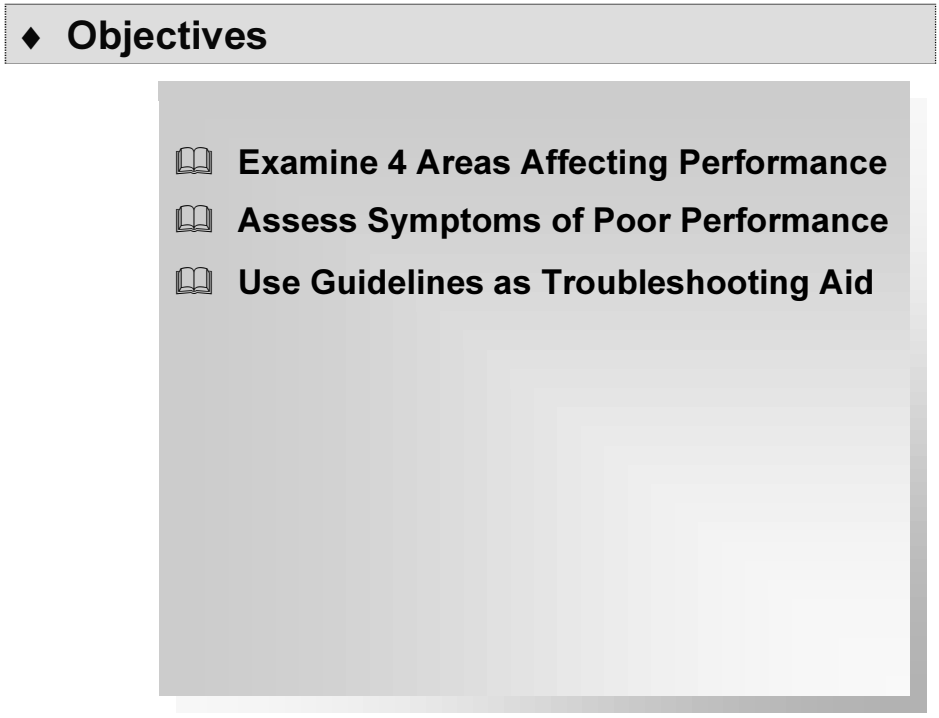
Display Performance



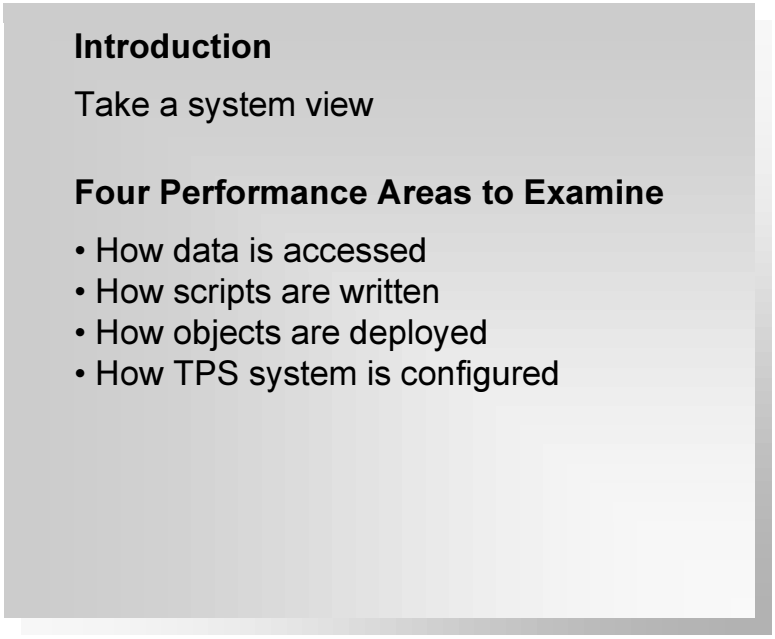


Objectives

In this module, you will review the guidelines in the Display Builder Authoring Tutorial from the perspective of troubleshooting a poorly performing display. At the end of this module you will be able to do the following:

- Examine 4 areas that affect display performance.
- Assess symptoms of poor performance.
- Use Display Authoring Tutorial Guidelines as a troubleshooting aid.

4 Areas that Affect Display Performance



Introduction

An overview of how performance of the Display Builder occurs within the TPS system can be reduced, at the risk of oversimplification, to four areas. While recognizing that there will be some overlap in these four areas, to get an approach defined, four performance areas are reviewed. Later in this material, these areas are correlated to the performance guidelines in the Display Authoring Tutorial as a troubleshooting job aid.

Four Performance Areas to Examine

Quick summary of common approaches to better performance

•How data is accessed:
- • access data from cache whenever possible
- • build-time bind (i.e., use SET operator with DISPDB)
- • adjust data collection rates (0 for static values)

• Whether scripts are performant
- • do not duplicate values in OnDisplayStartup scripts with those in OnDataChange scripts
- • do not solicit user input via modal dialog boxes (MsgBox, AskBox, etc) in OnDataChange scripts
- • access LCN data from public variables or display parameters whenever possible

•How objects are deployed
- • conserve objects in a display (use text with fill property versus rectangle)
- • replace non-parameterized embedded displays with groups
- • if large number of OLE objects are needed, use alternatives (primitives not OLE button)
- • practical animation (blinking, frame approach, not over bitmap)

• How TPS system is configured
- • memory
- • GUS focused machine
- • display loading on UCN nodes

Tutorial Guidelines for Optimization

Troubleshoot and optimize with guidelines

- Use the following chart as a decision aid

Display Optimization Decision Aid Use the following troubleshooting and optimization priorities when analyzing a non-performant display (where the priorities 1= highest, 3=lowest priority). Devote your attention to the highest priorities. Refer to the section in the Display Authoring Tutorial for guideline details, implement the recommended change(s), and determine if performance improves.	Data Access	Scripts	Object deployment	TPS configured
1.2 Building a Display				
Keep display density (graphics & data) to a "minimum"			3	
Use as few LCN point.parameters as necessary in any display	2			
Design displays utilizing a multiple-window workspace			2	
Minimize the number of OLE Controls in a display			2	
Use bitmaps judiciously in your displays			2	
Avoid putting animated objects over a bitmap			2	
When displays start getting complex, name objects			3	
1.3 Running Displays				
Hardware configuration contributes to performance				2
Keep PC focused on GUS				1
For fastest callup, place display files on a local drive				2
For faster callup, use "InvokeDisplay( )"to invoke a display from another display				3
1.4 Accessing Data in Scripts				
Use public variables or display parameters instead of DDB items or LCN points	1	1		
Use the DDB instead of the LCN for temporary storage or passing of data	1	1		
Use an error handler instead of doing LCN reads for errors	1	1		
Information on how to script indirect references	1	1		
In OnDataChange scripts, use "dispdb.<entity>.[name]" to reference the name of a DDB entity	1	1		
Use OnDataChange script with some added for logic for values that need to be read only once	2	2		
1.5 Scripting Events Properly				
OnDataChange	1	1		
OnPeriodicUpdate	1	1		
User events, such as OnLButtonClick	1	1		
OnDisplayStartUp	1	1		
OnDisplayShutDown	2	2		
1.6 Scripting Animation in a GUS Display				
Minimize the use of animation in a GUS display		2		
When testing a display having animation, monitor the CPU usage of the display as part of the test		3		
Consider the "frame" approach to animation instead of the "translation-and-rotation of objects"		3		
1.7 Using Recent Change Zone Enhancements				
Consider using the Display Data Base (DDB) item, $cz_enty in your custom Change Zone	1	1		
1.8 Referencing Data in Subroutines and Functions				
Be judicious when referencing data in subroutines and functions invoked from OnDataChange scripts		1		
Consider using a public script variable when reading the same LCN variable repeatedly		1		
1.9 Fine Tuning Performance				
Consider the following methodology as an approach to fine tuning performance	1	1		
1.10 Embedded Displays				
"Conserve" objects within frequently used embedded displays			2	
Use a display parameter of data type object to read/write data between embedded displays if possible		2		
1.11 Reducing Display Loading On UCN Nodes				
The basic rules for grouping	3			3

Estimating Display Performance

Introduction

Refer to section in Tutorial Guidelines

GUS User Interface Style	Display Callup Performance	
US Replacement	**Native Window Displays:**	equivalent to TDC
Basic Monitoring and Control	**GUS Display:**	2 - 5 seconds
	Native Window Displays:	equivalent to TDC
Multi-Level Monitoring and Control	**Main Display** :	2 - 5 seconds
	Mid level:	1 - 2 seconds
	Point Viewers:	1 - 1.5 seconds
	Native Window:	equivalent to TDC
Wide Area Monitoring and Control	**Overview**:	5 - 7 seconds
	"Close up":	2 - 5 seconds
	Mid level:	1 - 2 seconds
	Native Window:	equivalent to TDC

Introduction

Refer to the section in the Display Builder Authoring Tutorial, "GUS Display Performance," for guidelines on estimating performance.

Performance Fine Tuning

How to fine tune a display

1. Open the display in the GUS Display Builder.
2. View the Collection List.
   Note the number of variables being collected and
   Note the collection rate of each variable.
   Collect a variable at the slowest rate possible.
3. Note the number of embedded displays in the main display and the number of instances for each embedded display.
4. Repeat steps 1 – 3 for each display.
5. From the data gathered, determine which embedded displays are used the most and start by reviewing them for possible performance enhancements.

How to fine tune a display

(the following is excerpted from the Tutorial Guidelines, "Fine Tuning Performance")

1. Open the display in the GUS Display Builder.

2. View the Collection List.
   - Note the number of variables being collected. [the collection list contains 12 items per page x number of pages]
   - Note the collection rate of each variable. Determine if the collection rate is valid for each variable.
   - Collect a variable at the slowest rate possible. If a variable is known to be static, make its collection rate zero. [Example: EUDESC at 4 seconds is not practical]. By setting the collection rate to 0, the value of the variable will only be collected at startup. **This will not improve performance at invocation time, but it will reduce the load on the LCN and HOPC Server while the display is running.**

3. Note the number of embedded displays in the main display and the number of instances for each embedded display.

The easiest way to get this information is to invoke the "Replace Embedded Display" dialog. All the information is provided in this dialog. Then simply cancel the dialog.

4. Repeat steps 1 – 3 for each display.

5. From the data gathered, determine which embedded displays are used the most and start by reviewing them for possible performance enhancements. Verify that the collection rate is valid for each variable.

Example Performance Assessment

Results from customer site performance assessment on next page

- Your optimization effort may differ
- Use as a guide for your displays
- Note: the SET operator now recommended for GDB

Results of a performance assessment

The following results are from a customer site performance assessment. Although your optimization effort may differ from the following example, you can use this assessment as a guide to optimizing your displays. (Note: the SET operator, which build-time binds, was added as a GDB performance enhancement since this assessment was made.)

Step	Optimization focus	Rationale or results of action
1	The performance assessment began with a display optimization on the most frequently used embedded displays.	A frequently used non-optimized embedded display greatly impacts overall performance
1a	Within the frequently used embedded displays, attempt to conserve the number of OnDataChange() scripts.	Rationale: Remember that each entry in the value tab of a text object generates a separate OnDataChange() scripts. Action: You could consolidate these by reading the related values within a single OnDataChange() script and explicitly set the text objects to the values.
1b	Within the frequently used embedded displays, attempt to conserve objects.	Action: Use the fill and line properties of a text object to remove extra rectangles from an embedded display. Rationale: The change is minor in the embedded display, however, if a display uses many instances of that embedded display, the total number of objects are reduced which can improve performance.
1c	Within the frequently used embedded displays, use in line parameters.	Rationale: Inlines were though to have helped performance. Rationale: Since inlines are used for collectors (ACKSTAT), use inlines for other data access.
1d	Replace embedded displays that have no parameter with groups if possible.	Action: Consider keeping a file of process symbols called 'symbols,' which contains such items. Action: If an embedded display does not depend on an object to make it work, (i.e., only the me object is used), and it is unlikely that this embedded display will ever need to be replaced, then consider adding it to the symbols file. Rationale: A group is faster than an embedded display, since there is some overhead involved in keeping track of embedded displays.
2	Review process data access from OnDataChange scripts and subroutines	Rationale: Optimize data access for faster access.
2a	For values that are read only once, still use OnDataChange to read once.	Action: Put logic around those variables so that code is only walked once. (i.e., "If not once"). Rationale: At display startup, values for OnDataChange scripts are read.
2b	Identify any process data values that are accessed more than once. (i.e., Is FIC100.PV accessed more than once?)	Action: Store the value in a BasicScript variable.
2c	Identify whether any lcn access is made repeatedly among a collection of subroutines.	Action: Consider storing the value in a global BasicScript variable. Caution: Only lcn variables and display parameters will generate an OnDataChange event.
2d	Review error handler code	Action; use an error handler to check the status of an LCN reference, as opposed to reading the status of the point explicitly.
3	Review other scripts, objects	Rationale: Additional performance gains
3a	Adjust collection rates	Action: Use 0 for static values. Rationale: This allows values to from cache, which is faster than accessing lcn
3b	Review OnDisplayStartup scripts	Action; place same read in an OnDataChange. Rationale: OnDataChange takes advantage of cache. Each lcn read in OnDisplayStartup adds .5 seconds to display callup time. Placing same read in OnDataChange does not.
3c	Review OnPeriodicUpdate scripts	Action: Consider using blinking for animation to minimize performance impact.
3d	Review how OLE objects are used	Action: For displays that need to be fast, use alternatives like BasicScript tools, or draw your own button (Advantage: can change button color).
3e	Review file sizes	Action: Keep file sizes to a minimum. Tip: Display migration drastically reduces file size.
4	Building Tips	
4a	Save your work often, and periodically back up to another file.	Action: A Save As usually results in a smaller sized file than just a save.
4b	Name objects	Rationale: When displays get complex, it is good practice to name objects. It is then possible to view the script on the object by selecting the named object from a browser window pull down list.
4c	Put all code, including subroutines and functions, on one object if possible.	Rationale: Easy to locate code if it is on one object. Other objects make calls to desired sub or function.