**Introduction**

ArchestrA Graphics empower the developer with advanced graphic creation and animation tools that allow for intricate, stunning visualization. The graphics, being vector-based, also allow for “resolution-neutral” symbols that do not distort when changing sizes. Custom Properties and scripts provide a way to make graphics modular and further enhance functionality.

Because the graphics are so flexible and powerful, a developer must consider that ArchestrA graphics oftentimes can utilize more processor time and take up more memory than traditional InTouch graphics. This document provides a set of guidelines to creating incredibly powerful, yet efficient graphics.

**Document Conventions**

This document uses a key to explain the ways the guidelines affect the runtime of the graphic. The bar shown shows five portions of a graphic’s life cycle. If the guideline affects one of these portions, it will be colored; otherwise, the guideline will be white.

**Call-Up Time**

The five portions are defined as follows:

- Retrieve the Graphic Definition
- Render the Graphic on the Window
- Bind Data to the Graphic
- Continuous Updates
- Closing the graphic

Additionally, graphics will be show to represent how the graphic affects memory usage, CPU load, and memory utilization. The graphics will not be shown if the guideline does not affect the property:
Gradient Usage

Visually, one of the main things that set ArchestrA graphics apart from traditional InTouch graphics is the ability to utilize color gradients on lines, text, and shape fills. While gradients can certainly make graphics “pop” in comparison to solid colors, they do take longer to render, as each individual color in the gradient is rendered at runtime.

When using gradients, it is important to make sure that the visual effect is actually perceived by the user. For example, consider the following graphic:

In this warning symbol, we have several transparencies and gradients; the main yellow fill gradient, a border gradient on the yellow triangle, a greyscale line gradient, a gloss transparency, and a shadow element under the skull drawing. Now, consider that this graphic will be used multiple times on a graphic, but shrunk to a much smaller size, as shown below:
Looking at this graphic, we can see that the visual effect of several elements has vanished; The greyscale line gradient, yellow border gradient, gloss transparency, and skull shadow all cannot be detected by the eye. However, this graphic holds the same “cost” from a memory and CPU usage perspective as the larger graphic. If, instead, a smaller graphic was created without the visual elements that cannot be seen, the “cost” of the graphic will be considerably less. As a result, the graphic will open faster, take up less static memory, and animate quicker.

Therefore, the recommendation is to use gradient color effects and transparencies only when visual effect will be able to be noticed by the end user.

**Custom Property Density**

Custom Properties allow the developer to add properties to an ArchestrA graphic that may be used in scripting and animations. The custom properties can also be tied to other references (i.e. InTouch tags or Application Server attributes) when embedded in another symbol or on an InTouch window. This can empower the developer to create advanced, detailed graphics.

When dealing with custom properties, there are two things to keep in mind:

1) Each custom property takes up memory and requires CPU time to bind and update, and
2) When embedding a graphic within another graphic, all the custom properties of that embedded graphic come with it. Hence, if you embed 5 graphics with 10 custom properties each, the resultant graphic will have 50 custom properties before even adding properties of its own.

Therefore, when creating custom properties, it is good to limit them to only necessary values.

A key feature of using ArchestrA graphics in a System Platform environment is that the graphics can be purely used for visualization; any calculations can be done in the objects and presented to the visualization layer. Thus, functionality can be done server side (in the objects) and the result can be presented as a single property to the graphic. When working with ArchestrA graphics in a system Platform environment, consider if a custom property may be better served in the object that the graphic is representing.

### Scripting and Multi-Variable Expressions

Scripts can be used in ArchestrA graphics to provide extra functionality to the graphic. There are a wide array of scripts that can be executed on a condition basis (on true/on false/data change of an expression), when a graphic first opens, closes, or periodically while the graphic is shown. This provides the developer with a plethora of options to enhance their graphics.

One must keep in mind that these scripts are executed synchronously within InTouch; until the script completes execution, nothing else happens on the displayed windows, including display changes. Hence, when creating a graphic with multiple scripts, one must consider the impact if it is set to run consistently while the graphic is opened.

Considerations for scripting in ArchestrA graphics include:

1) Setting a script to run at a high resolution - What is the desired result? Is every execution really needed? If not, then why set the script to run at a high time resolution.

2) Script density – How many scripts are on the symbol? If there are multiple scripts, are they always going to be used on every graphic?

3) Graphic density on a window – How many of this graphic will be on an InTouch window? Will there be other graphics with multiple scripts on the same window?

The same considerations should be taken for multi-variable expressions in animations. These expressions are evaluated as ad-hoc scripts. Hence, the more variables in the expression, the longer the expressions take to evaluate, and thus the more time it takes the animation to display/change.

Again, pairing ArchestrA graphics with System Platform provides the developer with a unique advantage. Application Server is built to run volumes of scripts. Hence, placing scripts that are execution heavy in
TechTip: ArchestrA Graphic Development Guidelines

Application Objects allows the ArchestrA graphics to be simply used for graphical representation. A developer should constantly be looking at scripting to determine if the script can be done server-side (in the objects). This also applies to Multi-Variable Expressions; a graphic will require less static CPU time if the expression is executed server-side, and then provided to the graphic as a single attribute.

Bitmap Usage

Bitmaps are commonly used to provide the user with real pictures of equipment, regions, customer logos, etc. While bitmaps are the most common selection when embedding an image, they also are uncompressed, and thus take up a significantly larger memory footprint than their compressed counterparts.

Below is a table of other common formats that are significantly more compressed than bitmaps, and their advantages/disadvantages:

<table>
<thead>
<tr>
<th>Format</th>
<th>Typical Compression Ratios</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG (High)</td>
<td>10:1 - 20:1</td>
<td>High quality - has little or no loss in image quality with continuous tone originals. Worse results for flat color and sharp-edge art.</td>
</tr>
<tr>
<td>JPEG (Medium)</td>
<td>30:1 - 50:1</td>
<td>Moderate quality - usually the best choice for the Web.</td>
</tr>
<tr>
<td>JPEG (Low)</td>
<td>60:1 - 100:1</td>
<td>Poor quality - suitable for thumbnails and previews. Visible blockiness (pixelation).</td>
</tr>
<tr>
<td>PNG</td>
<td>10-30% smaller than GIFs</td>
<td>PNG's behave similarly to GIFs only better; they work best with flat-color, sharp-edged art. PNGs compress both horizontally and vertically, so solid blocks of color generally compress best.</td>
</tr>
</tbody>
</table>

Another common issue occurs when a developer takes a large bitmap and shrinks it in InTouch / ArchestrA Graphics to a considerably smaller size for presentation on the screen. While the visual effect is smaller, the memory usage is as if the graphic were full size. In this case, a developer should open the bitmap in a photo editor, shrink it to the desired size, and save it in a compressed format. This will provide a significant memory savings and also significantly affect the graphic’s call-up time.
Embedding large numbers of graphics in a parent graphic can provide the developer with an easy way to configure a graphic to serve multiple purposes. However, doing this does come at a cost; every graphic that you bring in brings along its own custom properties, scripts, and individual graphic elements.

Consider the following graphic; this single warning graphic consists of the six embedded graphics around it, all layered on top of each other – a custom property determines what to show. In this case, there are several details on the graphic that aren’t shown at the size it is embedded on the screen, and additionally, only one graphic is shown at a time.

The developer would be better served to simply pick the individual graphic from the graphic toolbox to embed on the screen, thus eliminating both the container graphic AND the other five embedded graphics from the definition. If there are a large number of these graphics on the screen, this translates to significant CPU and memory savings, and will result in a significantly faster call up time.
Element Grouping

ArchestrA Graphics allow the developer to group graphic elements together. While this is a fairly common practice, there is an oft overlooked benefit of grouping graphic elements that are not animated. These grouped, static graphics render as a single image at runtime. This translates into significant drops in call up time and static CPU usage. Consider the following two tanks.

The two tanks look identical; however, the tank on the left opens in half the time with half the static CPU load as the tank on the right. The only difference between the two is that the developer of the tank on the left grouped all the static elements, while the developer of the tank on the right left the elements ungrouped.
Therefore, it is a good practice to group these non-animated elements as much as possible.

**Hardware Choices**

Users of InTouch have been accustomed to using low-end hardware for their runtime systems. Traditional InTouch requires very little to run from a graphic perspective. However, ArchestrA graphics, due to the more robust feature set, do require more from a workstation than traditional InTouch; therefore it is more important to follow recommended guidelines.

1) A faster CPU clockspeed will provide a better experience than multiple core processors at a lower clock speed.
2) Allow for a maximum of 3Gb of RAM to be used on graphic-intense InTouch systems; when working with a Terminal Services environment, allow for 3Gb PER RUNNING SESSION.
3) Faster hard drives will provide for faster call up time.

**InTouch Window Memory Management**
InTouch 2012 introduced built-in memory management to allow for InTouch to maintain windows in memory even when windows are closed by the user. This results in faster loading times when opening the window the next time.

The In-Memory Window Caching is configured in WindowMaker by going to **Special->Configure->WindowViewer**.

There are two sections of the In-Memory caching:

1) **Use In-Memory Window Cache** – this sets the limit of RAM used by WindowViewer to cache windows, based on the total RAM on the PC, or 3Gb, whichever is smaller. Checking this box gives two options:
   a. **Memory Limit for In-Memory Windows** – this is the limit of RAM by percentage that WindowViewer will use for caching windows. The windows are cached in a First-In, First-Out
method. When the limit is reached, the oldest Window in memory will be released from the cache, and the newest one will be added.

b. In-Memory Window expiration time – this limits the window’s length to be in the cache when inactive – it will be released after the inactivity expiration time elapses. Setting this value to 0 will disable this functionality.

2) High Priority Window Caching – this functionality allows for certain windows to ALWAYS remain in the cache, regardless of the FIFO queue. This has a separate memory limit to allow for the high priority windows to stay active – its own FIFO queue takes effect when the high priority window memory limit is reached. To set a window up for High Priority, simply select it in the window below the caching selection.

Conclusion

ArchestrA Graphics provide the user with significant advantages over traditional InTouch graphics. However, due to their extended functionality set, they do require more resources than traditional graphics. The guidelines above will allow you to create visually advanced graphics that still function efficiently at runtime. As an example, the graphic below, while intricate and aesthetically pleasing, has a call-up time of under 1 second, and its static CPU load is under 1%.