Image Preview

This feature allows users to display the image created by a particular task directly from the Tractor Dashboard interface by selecting "Preview Output Image" from the context menu (right-mouse). The images can be displayed directly by the browser itself, or in some cases through an external application running on the user's local desktop system. In either case, some site configuration is required.

Summary -- to enable a site-specific policy regarding image preview, administrators should review and likely modify the default preview handlers in the config file "trSiteDashboardFunctions.js" -- look for the callback function named "trSiteOpenPreviewLink".

Background

There are a few system considerations to keep in mind when thinking about image previews through Tractor Dashboard:

- Where are the image files located? Applications launched from a job might run on the user's desktop or out on the farm somewhere. When those applications run, where do they write their output? to the local disk on the machine that they happen to be running on? to a central file server? are these files "globally visible" through some sort of file sharing?
- Which image file formats are in use? Can the usual output images be displayed directly by your browser's built-in "img" support, or will special pre-processing or external display programs be needed?
- Who is using the Dashboard interface? Are the people interested in seeing preview images always the artists themselves, working on their own desktop workstations, with their Tractor Dashboard browser running there on the same machine? Preview schemes may require different configuration schemes if there are wranglers, TDs, or administrators that will access images from different user's jobs, while accessing Tractor's web interface from various network locations and on differing devices.
- By default, security features in browsers prevent web page applications like Tractor Dashboard from launching external image display programs, or any other process. Doing so would open serious security holes since it would mean that the javascript embedded in random web pages could "do things" to your local machine, like run executables or read and write files. There are obviously site-specific considerations to be made when evaluating the trade-off between the practical value and risks of such access.

Preview Solutions

There are two basic approaches to displaying images through the web-browser-based Tractor Dashboard. The simplest and safest one is for your web brower itself to fetch the images itself, directly from a web server on your network, like any other image on the web. The other approach is to install a custom image viewing plug-in on each user's desktop (or other web-browsing device), and then direct the plug-in to read load the image using its own custom mechanism for finding and displaying the pixels.

In the first case, javascript code in Tractor Dashboard causes the web browswer to request the image from Tractor Engine or another **web server** somewh ere the local network. Once the browser has been given the URL to the image, it will load the image itself through http, in a separate browser window. This approach requires that all previewable renders must generate output images that the browser can display natively; note for example that TIFF images are not widely supported on most browsers. This scheme also requires your renderers must write their output images into locations that can be read by the given web server -- that is, the renderings must appear to be within the webserver's document tree, from the web server's point of view. Job scripts that specify the preview image names must be careful to refer to the images using full network paths to the output image *from the web server's point of view.* At sites where assets are stored on central fileservers, it is likely that such a network path is already in use and that the chaser/preview paths used by the engine will be the same as those used by artists on their own workstations.

You can cause network mounts to appear "virtually" in the tractor-engine web document root by adding entries to the SiteURLMap configuration table, located in tractor.config. Each pair of paths in the table represents a mapping of reference relative to the engine's document root, to a location elsewhere on the filesystem. Frequently the added mappings take a simple "duplicate" form:

> *"/net/assets/previews", "/net/assets/previews",*

which maps http requests for images in "/tractor/net/assets/previews" to the filesystem location "/net/assets/previews".

In the **browser plug-in** approach, the Tractor Dashboard JavaScript code will ask the user's browser to open the chaser URI specified in the job script. These will have a general format like this:

> *scheme_name:scheme_parameters*

A familiar example of such a thing is a web page containing a "mailto" URI like "mailto:myfriend@example.com" in a link. When someone clicks on that link the browser has a special handler for the "mailto" scheme, which usually involves asking the underlying operating system to launch the user's favorite mail application.

Other built-in or 3rd-party browser extensions can also be invoked using this URI scheme mechanism. Tractor simply invokes the browser's "open" method on the URI supplied in the job's chaser command and assumes that the site administrators have coordinated both the job script generation and the available browser plug-ins to cause the desired application invocation on the user's desktop. These sorts of plug-ins can do many arbitrary things including reading local files on the browser's host, which of course can be very useful, and easily abused.

So with the URI handler approach, preview renders could write temporary images somewhere on the user's local desktop, for example, and the dashboard + plug-in combination could cause a full-function native display application to launch and display the image directly on the desktop. Globablly shared files are not necessarily required in this case, although the launched app could certainly use them if they are available. The disadvantages of the plug-in approach is that you will need a browser-specific version of the plug-in compiled for each workstation platform or other device on which you will be viewing the Tractor Dashboard web page. Also, if you are trying to preview a rendering while away from your desk, the local plug-in on your device must be able to access the output image by itself somehow, wherever it may be on the web.