Cross-Platform Rendering

Rendering on Mixed-OS farms or in Mixed-OS facilities

Many Qube customers have artists running on one operating system while the render farm runs on another. Some have render farms made up of multiple operating systems. In both cases, rendering becomes a problem as paths are different across operating systems. For example, the path to a Maya scene file on OS X may be "/Volumes/server/projects/foo/maya.mb" whereas the path to the same file on Windows is "X:\Projects\foo\maya.mb" and on Linux is "/media/projects/foo/maya.mb". To complicate matters, the path to the renderer is likely different across operating systems as well.

We have made use of or developed ways of working with heterogeneous facilities: OS-level adaptations, worker-side path translation maps, client-side path translation maps, and "appFinder" jobs. There is a time and a place for each. The details follow:

OS-Level Adaptations

This is a primary recommended solution. It works in almost all cases **except** when a job is submitted from Windows and run on another OS (due to the direction of the path separators and/or the leading drive letter) -- For that situation, you will need to set up some form of path translation as described in another section. In all other cases, this will work not only for resolving paths to project files, but also resolving references used within the project file - something that the other methods can not do. You will, however, need to perform these actions on every machine in your render farm individually.

The idea is that you want the path to any file to resolve the same way on all operating systems. For each OS, this is accomplished through the use of symbolic links. In our example above, we described the path to a file "maya.mb" on three OSs:

OS	Path
Windows	X:\Projects\foo\maya.mb \\server\Projects\foo\maya.mb
OS X	/Volumes/server/projects/foo/maya.mb
Linux	/media/projects/foo/maya.mb

Windows Paths

1. When windows evaluates a path without a drive letter, the root drive (typically C:) is assumed.
2. Most applications that run cross-platform will accept either forward or backward slashes as path separators

OS X to Windows example

When an artist is running OS X but the workers are running Windows, you will want to make Windows recognize the OS X path. The job will be sent with the path to the maya file as "/Volumes/server/projects/foo/maya.mb". Keeping in mind that Windows assumes the system drive when no drive letter is given, that path will get a C: prepended to it. We are now working with:

	path
evaluated	C:/Volumes/server/projects/foo/maya.mb
actual	X:\Projects\foo\maya.mb -or- \\server\Projects\foo\maya.mb

So the differences is c:/Volumes/server - vs - X:. Beyond that, the paths are the same. Our job, then, is to link the two paths through symbolic links. On windows, this is done on the command prompt with mklink:

```
mkdir c:\Volumes\server
cd /d c:\Volumes\server
mklink projects X:\Projects    - or -   mklink server \\server\Projects
```

Now you can:

```
dir X:\Projects\foo\maya.mb
dir \\server\Projects\foo\maya.mb
dir C:/Volumes/server/projects/foo/maya.mb
```

and get the same results each time.

OS X to Linux example

OS X and Linux integration is done in a similar manner. We want to make the OS X path work on the Linux machine. In our example above, the OS X path to maya.mb is "/Volumes/server/projects/foo/maya.mb" whereas the Linux path to that file is "/media/projects/foo/maya.mb". Both paths are the same starting at "projects", therefore, we need to make the OS X directories on the Linux machines with an appropriate symlink, as follows:

```
sudo mkdir -pv /Volumes/server
cd /Volumes/server
sudo ln -sv /media/projects
```

Now you can:

```
ls /Volumes/server/projects/
ls /media/projects
```

and get the same result (note the use of the trailing slash in the first example)

Note that Linux to OS X operates exactly the same way with the same commands - you'll just need to replicate the start of the Linux path on OS X through the terminal.

Worker-side Path Translation

Starting with Qube 6.4-2, path translation can happen at the worker level. This has an advantage over client-side path translation in that each worker can define the appropriate translation maps for itself rather than using a generic map for all machines.

The basic idea here is that we do simple string replacement for paths that are found in the submission dialog. Being that we are only converting submission parameters, any internal references to paths may still be broken. For this type of path translation to work, the application must be using relative paths for assets.

Worker-side path translation is configured in the worker's local qb.conf or in the supervisor's central qbwrk.conf through the worker_path_map variable.

For detailed instructions, see worker_path_map.

To centrally manage worker path maps, see Centralized Worker Configuration

Client-side Path Translation

Client-side path translation is similar to worker-side path translation in that it does a simple find and replace on paths in the submission dialog. Unlike worker-side path translation, client-side path translation happens when the job is submitting, which means you must submit from one OS and render on the same other OS - you cannot render on multiple OSs like you can work worker-side path translation.

Client-side path translation is configured in the GUI's "Path Translation Map" section in the preferences.

To centrally manage client-side path translation, see the section on "Studio wide preferenes" in WV Preferences

AppFinder jobs

The last topic for cross-platofrm rendering is the path to the application itself. Maya, for example, may be in either `/Applications/Autodesk/maya<ver>/Maya.app`, `C:/Program Files/Autodesk/Maya<ver>`, or `/usr/autodesk/maya<ver>`, or some other custom location based on the operating system.

Jobtypes have handled this situation for years, but applications that do not have or use jobtypes (like After Effects or Cinema 4D) have not been able to run simultaneously on multiple operating systems until now appFinder jobs were introduced in Qube 6.4-2

The basic idea is that you define (or use) an application key, then define the known location(s) for that application. When submitting, you use the key for the executable field & let the appFinder do the rest.

AppFinder jobs require Python 2.x be installed on the workers. OS X comes with python pre-installed. Linux users can install python through the built-in package manager. Windows users will need to install Python manually. See Installing Python.

For detailed instructions, see AppFinder Jobs