Batch Simulations

With fluid and object dynamics it is often necessary to run several simulations to find the best solution. Wouldn’t it be convenient if you were able to complete all these different setups overnight without the need for manual intervention, like loading, resetting, and simulating the scene? A batch script can help you to automatize this process.

Name	BatchSimulator.rfs
Type	Batch script
Description	The script can be used to automatically load and calculate various projects ready to simulate. The results will be stored under the appropriate project folders. Export resources have to be determined manually before under "Export Central".

What the script should do:

- Use a given root path where all the simulations are stored
- Append the various project names to the root path
- Load and simulate the specified scenes

The very first thing you have to do is to find the path to the common folder, where all simulations are stored. This is not really necessary, because you can also specify custom paths to different locations, but it is much better to have everything stored in one place. If you are using the same path as specified under

Preferences > General > Scenes Folder

you can directly copy and paste it, as it already points to the correct directory. If you have them stored under a different location you have to find out manually. The most convenient and reliable way is to use an absolute path. As you can see it is not necessary to use the Windows-specific backslash for the path:

```
projectRoot = “E:/RF Projects/Batch Simulations/DensityTest/”
```

Under OS X the directory delimiter is a colon (:), but it can also be substituted by a slash:

```
projectRoot = “/Users/Next Limit/Batch Simulations/DensityTest/”
```

The next task is to determine the different project names. As in the previous examples, you have to store multiple values and this requires an appropriate data structure: a list. You simply have to write the different files names to the list and loop through it.

```
projectList = ["Density_0750","Density_1000","Density_1100","Density_1300"]
```

You can append as many projects as needed: there is virtually no limit. The advantage is that the names do not have to share a common prefix or suffix, and it is possible to enter any desired name. Now you have to go through the individual elements of the previously created list and load the files. RealFlow always establishes the same structure:

project name/project name.flw

With the concatenation operator “+” the elements can be joined together:

```
for projectFile in projectList:
    scene.load(projectRoot+"/"+projectFile+"/"+projectFile+".flw")
    scene.reset()
    scene.simulateRange(0, 200)
```

With `scene.simulateRange(start, stop)` you are able to specify the simulation range. In many cases the ranges are not the same for each project, and this will be an addition for this script. Assuming that the start frame is not subject to change, it is only a matter of different end frames. For this purpose another list is created containing all the stop frames. A counter is needed, too:

```
stopFrameList = [50,100,150,200]
counter       = 0
projectRoot   = "E:/RF Projects/Batch Simulations/Various/"
projectList   = ["Splash","filling_a_glass","rbd test_01","GridFluid_BeachScene"]

for projectFile in projectList:
    scene.load(projectRoot+"/"+projectFile+"/"+projectFile+".flw")
    scene.reset()
    stopFrame = stopFrameList[counter]
    scene.simulateRange(0, stopFrame)
    counter += 1
```

The statement `stopFrameList[counter]` is the important element here. Since the counter is incremented with each frame, you can read out the stop frames from the list. The counter is used as an index for each list entry. If you need different start frames, you have to create another list. To save the assignment of extra start/stop variables, the `simulateRange()` statement can then also be written as:

```
scene.simulateRange(startFrameList[counter], stopFrameList[counter])
```

To speed up the entire simulation process you should consider using "RealFlow -nogui" instead of the GUI application. The script has to be saved externally and can then be called with the “-script” flag.