Launching tools with invoke scripts

Overview

Invoke scripts are small programs, usually written in sh or bash, used to setup the application container environment so the tool can run properly. More specifically, invoke scripts are responsible for:

- Locating tool.xml for Rappture applications
- Setting up the PATH and other optional environment variables
- Starting the window manager
- Starting optional subprograms, like filexfer
- Starting the application

For most applications, the invoke script is a single command that calls the default HUBzero invoke script, named invoke_app, with a few options set. In some rare situations, the tool needs the application container setup in a manner that invoke_app cannot handle. In these cases, the tool developer can modify the tool's invoke script to appropriately setup the application container.

The sections below list out details regarding the options of invoke_app, how to launch Rappture tools using an invoke script that calls invoke_app, and how to launch non-Rappture tools using an invoke script that calls invoke_app.

invoke_app and its options

HUBZero's default tool invocation script is called invoke_app. It is a bash script, usually located in /usr/bin. When called with no options, the script tries to automatically find the needed information to start the applications. There are a number of options that can be provided to alter the script's behavior.

invoke_app accepts the following options:

```
-A tool arguments
-c execute command in background
-C command to execute for starting the tool
-d working directory
-e environment variable (${VERSION} substituted with $TOOL_VERSION)
-f No FULLSCREEN
-S No submit
-n nanowhim version
```

```
-p add to path          (${VERSION} substituted with $TOOL_VERSION)
-r rappture version
-t tool name
-T tool root directory
-u use environment packages
-w specify alternate window manager
```

Here is a detailed description of the options:

-A	Pass the provided enquoted arguments onto the tool. Example usage: `-A "-q blah1 -w blah2"` The options -q and -w are not parsed by invoke_app, but are passed on to the tool
-c	Commands to run in the background before the tool launches. Example usage: `-c "echo hi" -c "filexfer"` This prints "hi" to stdout and starts filexfer
-C	Command to execute for starting tool. Tool's command line arguments can be included in this option, or can be placed in the -A option. Example usage: Call a program, named myprog, located in the tool's bin directory: `-C @tool/bin/myprog` Call a program, named myprog, located in the tool's bin directory, with program arguments "-e val1" and "-b val2":

	`-C "@tool/bin/myprog -e val1 -b val 2"`
	
	Call a program, named myprog, located in the tool's bin directory with arguments -e val1 and -b val2, used in conjunction with invoke_app's -A option:
	
	`-C @tool/bin/myprog -A "-e val1 -b val2"`
	
	Call a program, named myprog, located in the tool's bin directory. We can omit the path of the program if it is an executable and located in the tool's bin directory because the tool's bin directory is added to the PATH environment variable. This would not work for calling a Perl script in a fashion similar to **perl myscript.pl** because in this case, **perl** is executable and **myscript.pl** is the argument.:
	
	`-C myprog`
	
	Call simsim with no arguments:
	
	`-C simsim`
	
	Call simsim with the options -tool and -values, to be parsed by simsim:
	
	`-C "simsim -tool driver.xml -values random"`
	
	Call simsim with the options -tool and -values, to be parsed by simsim:

	`-C simsim -A "-tool driver.xml -values random"`
-d	Change to this working directory. By default change to session directory.
-e	Set an environment variable. Example usage: `-e LD_LIBRARY_PATH=@tool/../${VERSION}/lib:${LD_LIBRARY_PATH}` Within the value part of this option's argument, the text ${VERSION} is automatically substituted with the value of the variable ${TOOL_VERSION}. Similarly, the text @tool is substituted with the value of ${TOOLDIR}. By setting the environment variable, you are overwritting its previous value.
-f	no full screen - disable FULLSCREEN environment variable, used by Rappture, to expand the window to the full available size of the screen.
-p	Prepend to the PATH environment variable. Example usage: `-p @tool/../${VERSION}/bin` Within the value part of this option's argument, the text ${VERSION} is automatically substituted with the value of the variable ${TOOL_VERSION}. Similarly, the text @tool is substituted with the value of ${TOOLDIR}. By setting this option the PATH environment variable is adjusted, but not overwritten. The directory @tool/bin is automatically added to the PATH environment variable.
-r	Sets ${rappture_version} which dictates which version of Rappture is used. If left blank the version will default to the special keyword "system", which represents whichever version is pointed to by the default Rappture environment in "use". A "use -e -r rappture" will be performed to figure out where Rappture is

	installed. If set to the special keyword "none", searching for Rappture executables (rappture, simsim, about) will be skipped and use of these executables will be disabled. This flag works well on hubs where multiple versions of rappture are installed. Users can specify their own version of Rappture to use by updating the PATH environment variable to include the directory where the "rappture" executable is installed.
-S	Disable submit client and run job locally. This flag takes no arguments and is used for debugging. It disables the use of submit client from the -C command that will be executed. The default behavior, when the flag is not given, is to run the command through the submit client unless the command is "rappture", "simsim", "getrappturexml", or "nanowhim", none of which are run through the submit client. Setting the flag on the command line will add your command to the list of commands that do not run with the submit client.
-t	sets ${toolname} which is used while setting up tool paths for TOOLDIR and TOOLXML. ${toolname} is the short name (or project name) of the tool. It is the same as the name used in the source code repository. With respect to the tool contribution process, it is the "toolname" in the path /apps/toolname/version/rappture/tool.xml. Setting this option will change the paths searched while trying to locate tool.xml and the bin directory.
-T	Tool root directory. This is the directory holding a checked out version of the code from the source code repository. It typically has the src, bin, middleware, rappture, docs, data, and examples directories underneath it. With respect to the tool contribution process, it is the "/apps/toolname/version" in the path /apps/toolname/version/rappture/tool.xml. Setting this option will change the paths searched while trying to locate tool.xml and the

	bin directory. Typically when testing this option is used to specify where the tool directory is. In this case, its the present working directory: `-T $PWD`
-u	Set use scripts to invoke before running the tool. Example usage: `-u octave-3.2.4 -u petsc-3.1-real-gnu` These would setup octave-3.2.4 and petsc-3.1 in the environment that your tool would launch in.

-w	Set the window manager. The default value is to use the ratpoison window manager if it exists. If ratpoison is not installed on the system, look for the icewm captive window manager setup. Use this flag to choose an alternative window manager. If your application does not require a window manager specify headless. The possible options are headless, ratpoison, captive, and icewm. If multiple options are specified the first one listed is selected.
	
	Examples:
	
	Use the icewm captive window manager.
	
	`-w captive`
	
	Use no window manager.
	
	`-w headless`

invoke_app is called from within a tool's invoke script. The invoke script is stored in the middleware directory of the tool's source code repository.

Using invoke_app with Rappture tools

Invoke scripts should be placed in the middleware directory of the tool's source code repository. A typical invoke script for a Rappture application looks similar to this:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t calc \
                         -C rappture
```

In the invoke script above, invoke_app, located in the directory /usr/bin, is called with "$@" , "-t calc", and "-C rappture". "$@" represents all options that the invoke script itself received. "-t calc" tells invoke_app that the toolname is "calc". "-C rappture" tell invoke_app to execute the rappture command. This information is used by invoke_app to figure out which tool it is supposed to be launching and where that tool is installed.

For most Rappture applications, the invoke script is very simple. The above is enough for invoke_app to start looking for a tool.xml file. invoke_app looks for the file named tool.xml. It uses the TOOLDIR variable to help decide where to look. If the tool.xml file is not found in the ${TOOLDIR}/rappture directory, invoke_app will exit explaining that it could not find the tool.xml file. The TOOLDIR variable can be set from the command line using the -T flag:

```
/usr/bin/invoke_app "$@" -t calc \
                         -C rappture \
                         -T ${PWD}
```

Actually, it is more common to see the -T flag provided to a tool's invoke script, and the option is forwarded to invoke_app by "$@":

```
./middleware/invoke -T ${PWD}
```

In the above example, the TOOLDIR variable is set to the present working directory, which is stored in the variable PWD. Specifying the -T option is usually not needed, but can help when invoke_app is confused on what it is supposed to be launching.

Using invoke_app with Jupyter Notebook tools

Invoke scripts should be placed in the middleware directory of the tool's source code repository. A typical invoke script for a Jupyter Notebook application looks similar to this:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t calc \
                         -C "start_jupyter -t -A -T @tool/bin calc.ipy
nb" \
                         -u anaconda-7 \
                         -r none \
                         -w headless
```

In the invoke script above, invoke_app, located in the directory /usr/bin, is called with "$@", "-t calc", "-C start_jupyter ...", "-c filexfer", "-w captive". "$@" represents all options that the invoke script itself received. "-t calc" tells invoke_app that the toolname is "calc". This information is used by invoke_app to figure out which tool it is supposed to be launching and where that tool is installed. "-C start_jupyter ..." tells invoke_app that the command to run to start the tool is "start_jupyter". "start_jupyter has several typical arguments as shown. "-r none" tells invoke_app that Rappture is not required. "-w headless" tells invoke_app not to start a window manager.

Using invoke_app with GUI tools

Invoke scripts should be placed in the middleware directory of the tool's source code repository. A typical invoke script for a non-Rappture GUI application looks similar to this:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t calc \
                         -C calc \
                         -c filexfer \
                         -w captive
```

In the invoke script above, invoke_app, located in the directory /usr/bin, is called with "$@", "-t calc", "-C calc", "-c filexfer", "-w captive". "$@" represents all options that the invoke script itself received. "-t calc" tells invoke_app that the toolname is "calc". This information is used by invoke_app to figure out which tool it is supposed to be launching and where that tool is

installed. "-C calc" tells invoke_app that the command to run to start the tool is "calc". In this case calc is UI program built using something other than Rappture. Possible GUI builders include but are not limited to PyQt and MATLAB. "-c filexfer" tells invoke_app to start up the filexfer program before starting the tool's graphical user interface. "-w captive" tells invoke_app to use the icewm captive window manager. For non-rappture applications the icewm captive window manager may be preferred over the ratpoison window manager if there are multiple graphical user interface windows that could popup.

The invoke script above could be made more svelte if the we did not want to start filexfer and we wanted to use the ratpoison window manager. After all, not all applications require files from the user, so they don't need the filexfer program. Here's an example of the tool named calc (the "-t calc" option), that is started by the executable named calc (the "-C calc" option), and uses the default window manager which is ratpoison.

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t calc \
                         -C calc
```

Other invoke script examples

Here are a few common invoke scripts examples that demonstrate using invoke_app options.

Use the -u option to setup Octave-3.2.4 in the path before starting the tool's graphical user interface. The -u option sources a "use" script (octave-3.2.4 in this example) from the /apps/environ directory.

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t calc \
                         -C calc \
                         -u octave-3.2.4
```

Use the -A option to send additional arguments to the command to be executed:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t calc \
                         -C calc \
```

```
                        -A "-value 13 -value 5 -op add"
```

Or:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t calc \
                         -C "calc -value 13 -value 5 -op add"
```

Launching a Matlab tool (named app-fermi) with a Rappture graphical user interface:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t app-fermi \
                         -C rappture
```

Launching a Python tool (named app-fermi) with a Rappture graphical user interface:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t app-fermi \
                         -C rappture
```

Launching a Java tool (named app-fermi) with a Rappture graphical user interface:

```
#!/bin/sh

/usr/bin/invoke_app "$@" -t app-fermi \
                         -C rappture
```