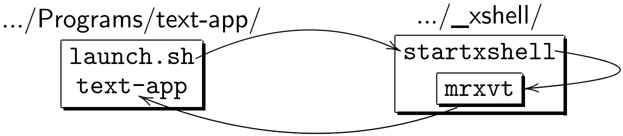
```
___   ___      _______. __    __   _______  __       __
\  \ /  /     /       ||  |  |  | |   ____||  |     |  |
 \  V  /     |   (----`|  |__|  | |  |__   |  |     |  |
  >   <       \   \    |   __   | |   __|  |  |     |  |
 /  .  \  .----)   |   |  |  |  | |  |____ |  `----.|  `----.
/__/ \__\ |_______/    |__|  |__| |_______||_______||_______|


                                  _  _ ___ _  _ _  _ ___
          with                   |\/| |__/  \/  |  |  |
                                 |  | |  \ _/\_  \/   |
```

Q: What is xshell?
A: Xshell is a collection of scripts which allow users to easily run text-based applications from the iLiad's menu system.

Q: What's the purpose of xshell?
A: Since the iLiad has no built-in command-line terminal, xshell is designed to install and keep track of user-installed terminals.

Q: How does xshell differ from mrxvt?
A: Mrxvt is a windowed terminal program, and the xshell installer includes mrxvt. The xshell scripts provide an interface to this installation of mrxvt.

Q: Why did you make xshell?
A: The iLiad can't handle a lone text-based application. One possible solution would be to package the mrxvt terminal with each separate text-based application, but this would waste valuable storage. Xshell implements a different solution: package text-based applications with a tiny launcher script. This launcher script locates and runs the terminal for the text-based application.



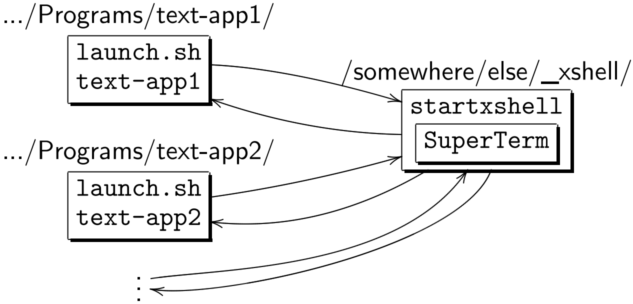
Q: How does it work, in short?
A: Each text-based application comes bundled with the `xshell` launcher script. When run, this launcher script locates and connects up with the `xshell` base installation, containing `mrxvt`. Then `startxshell` launches our text-based application in a `mrxvt` window.

Q: What are the advantages of this approach?
A: 1. Multiple applications share the same terminal program. This prevents wasting space. The terminal program can be moved to a different location.

2. It's possible to move `mrxvt` to a different location. It's probably best to keep it in internal memory, but some might prefer to move it onto a card.

3. Terminal programs other than `mrxvt` could easily be used.

4. All these changes are possible without reinstalling or reconfiguring the applications.



Information for developers

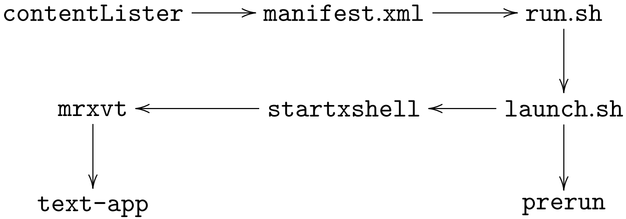
We now elaborate on the oversimplifed execution model summarized above.

The typical text-based application will live in a directory containing a `manifest.xml`, a `run.sh`, the application (which we shall call `text-app`) together with its supporting libraries, and a copy of the `launch.sh` script from `xshell`.

The `xshell` directory (typically named `_xshell/` to keep it hidden from the `contentLister`) contains `prerun`, `startxshell`, and `mrxvt`.

```
/usr/local/programs/text-app/      /mnt/free/_xshell/
   manifest.xml                       prerun
   run.sh                             startxshell
   text-app                           mrxvt
   launch.sh
```

The execution chain is summarized in the diagram below.



The `contentLister` is the menu from which a user selects an application. When the user selects the menu option corresponding to `text-app`, the `contentLister` executes `manifest.xml`, which contains the line `<startpage>run.sh</startpage>`, causing `run.sh` to execute.

The typical run.sh will be

```
#!/bin/sh
export scriptdir=`/usr/bin/dirname $0`
cd $scriptdir
./launch.sh --execute ./text-app
```

Next, launch.sh locates the `_xshell/` directory. It then runs `startxshell`, passing on the "`--execute ./text-app`" arguments. After setting up the fonts and libraries, `startxshell` calls "`mrxvt -e ./text-app`" finally launching `text-app` inside of an `mrxvt` window.

Upgrading `launch.sh`

Before `launch.sh` runs `startxshell`, it calls `prerun` with its version number and location. For example, if we are running `launch.sh` v0.5.0, it would execute something like

```
prerun 050 /usr/local/programs/text-app/launch.sh
```

Now suppose we just upgraded to `xshell` v0.5.1, which came with a new `launch.sh` v0.5.1. Our application only has v0.5.0. But when we execute `prerun`, it will replace the old `launch.sh` with the new one. When we restart our application, it will run with the new `launch.sh`.

How `launch.sh` finds `_xshell/`

First, `launch.sh` checks to see if a location was specified by the `--location` argument. If so, it will use that. Otherwise, it will check the location specified in `/home/root/.xshell`. If this is not valid, it searches `/mnt/free/`, `/usr/local/programs/`, `/mnt/card/`, `/mnt/cf/`, and `/mnt/usb/` for the subdirectories `_xshell/` and `Programs/_xshell/`. It selects the `_xshell/` directory with the highest version number. If no such directory is found, it fails.

Note The original search algorithm was to use `find /mnt/x/ -name startxshell`. However, I figured this was a security risk since an attacker might cause an application to create a temporary file named `startxshell`. Since FAT32 defaults to enabling execute permissions, this could lead to arbitrary code execution.

Command line arguments

`launch.sh`

`--help`	Displays a summary of these command line arguments
`--launcherversion`	Prints the version in a human-readable format
`--launcherversionnum`	Prints the version as a number
`--location`	Prints the location of `_xshell/`
`--location` *directory*	Uses the `_xshell/` directory in the specified location
arguments	All unrecognized arguments are passed on to `startxshell`

`startxshell`

`--help` or `--xshellhelp`	Displays a summary of these command line arguments
`--xshellversion`	Prints the version in a human-readable format
`--xshellversionnum`	Prints the version as a number
`--terminalinfo`	Prints the version of the terminal being used
`--execute` *command*	Executes the *command* in the terminal
`--hold`	Waits for the user to press a key after the program terminates
`--working-directory` *directory*	Starts the terminal in the specified *directory*
`--passargs` *arguments*	Pass all remaining *arguments* to the terminal program

Examples

```
launch.sh --hold --execute /sbin/ifconfig
```

This will open a window displaying `ifconfig`. It will remain open until the user presses a key.

```
launch.sh --passargs -title "My terminal"
```

This sets the title of the `mrxvt` window.

```
cat ‘launch.sh --location‘
```

Prints the code for the `startxshell` script currently being used.

```
launch.sh --help ; launch.sh --xshellhelp
```

Prints command line arguments available to both `launch.sh` directly, and those available indirectly via `startxshell`.

Ben Mares, June 2008