NAME

rstartd - a sample implementation of a Remote Start rsh helper

SYNOPSIS

rstartd

rstartd.real [−c *configfilename*]

DESCRIPTION

Rstartd is an implementation of a Remote Start "helper" as defined in "A Flexible Remote Execution Protocol Based on **rsh**".

This document describes the peculiarities of *rstartd* and how it is configured.

OPTIONS

−c *configfilename*

This option specifies the "global" configuration file that *rstartd* is to read. Normally, *rstartd* is a shell script that invokes *rstartd.real* with the **-c** switch, allowing local configuration of the location of the configuration file. If *rstartd.real* is started without the -c option, it reads */X11/rstart/config*.

INSTALLATION

It is critical to successful interoperation of the Remote Start protocol that *rstartd* be installed in a directory which is in the "default" search path, so that default rsh requests and the ilk will be able to find it.

CONFIGURATION AND OPERATION

Rstartd is by design highly configurable. One would like things like configuration file locations to be fixed, so that users and administrators can find them without searching, but reality is that no two vendors will agree on where things should go, and nobody thinks the original location is "right". Thus, *rstartd* allows one to relocate **all** of its files and directories.

Rstartd has a hierarchy of configuration files which are executed in order when a request is made. They are:

```
global config
per-user ("local") config
global per-context config
per-user ("local") per-context config
config from request
```

As you might guess from the presence of "config from request", all of the config files are in the format of an *rstart* request. *Rstartd* defines a few additional keywords with the INTERNAL- prefix for specifying its configuration.

Rstartd starts by reading and executing the global config file. This file will normally specify the locations of the other configuration files and any systemwide defaults.

Rstartd will then read the user's local config file, default name $HOME/.rstart.

Rstartd will then start interpreting the request.

Presumably one of the first lines in the request will be a CONTEXT line. The context name is converted to lower case.

Rstartd will read the global config file for that context, default name /usr/lib/X11/rstart/contexts/<name>, if any.

It will then read the user's config file for that context, default name $HOME/.rstart.contexts/<name>, if any.

(If neither of these exists, *rstartd* aborts with a Failure message.)

Rstartd will finish interpreting the request, and execute the program specified.

This allows the system administrator and the user a large degree of control over the operation of *rstartd*. The administrator has final say, because the global config file doesn't need to specify a per-user config file. If it does, however, the user can override anything from the global file, and can even completely replace the global context config files.

The config files have a somewhat more flexible format than requests do; they are allowed to contain blank lines and lines beginning with "#" are comments and ignored. (#s in the middle of lines are data, not comment markers.)

Any commands run are provided a few useful pieces of information in environment variables. The exact names are configurable, but the supplied defaults are:

$RSTART_CONTEXT	
$RSTART_GLOBAL_CONTEXTS	the global contexts directory
$RSTART_LOCAL_CONTEXTS	the local contexts directory
$RSTART_GLOBAL_COMMANDS	the global generic commands directory
$RSTART_LOCAL_COMMANDS	the local generic commands directory

$RSTART_{GLOBAL,LOCAL}_CONTEXTS should contain one special file, @List, which contains a list of the contexts in that directory in the format specified for ListContexts. The supplied version of ListContexts will cat both the global and local copies of @List.

Generic commands are searched for in several places: (defaults)

```
per-user per-context directory ($HOME/.rstart.commands/<context>)
global per-context directory (/usr/lib/X11/rstart/commands/<context>)
per-user all-contexts directory ($HOME/.rstart.commands)
global all-contexts directory (/usr/lib/X11/rstart/commands)
```

(Yes, this means you can't have an all-contexts generic command with the same name as a context. It didn't seem like a big deal.)

Each of these directories should have a file called @List that gives the names and descriptions of the commands in that directory in the format specified for ListGenericCommands.

CONFIGURATION KEYWORDS

There are several "special" *rstart* keywords defined for *rstartd* configuration. Unless otherwise specified, there are no defaults; related features are disabled in this case.

INTERNAL-REGISTRIES name ...
Gives a space-separated list of "MISC" registries that this system understands. (Registries other than this are accepted but generate a Warning.)

INTERNAL-LOCAL-DEFAULT relative_filename
Gives the name ($HOME relative) of the per-user config file.

INTERNAL-GLOBAL-CONTEXTS absolute_directory_name
Gives the name of the system-wide contexts directory.

INTERNAL-LOCAL-CONTEXTS relative_directory_name
Gives the name ($HOME relative) of the per-user contexts directory.

INTERNAL-GLOBAL-COMMANDS absolute_directory_name
Gives the name of the system-wide generic commands directory.

INTERNAL-LOCAL-COMMANDS relative_directory_name
Gives the name ($HOME relative) of the per-user generic commands directory.

INTERNAL-VARIABLE-PREFIX prefix
Gives the prefix for the configuration environment variables *rstartd* passes to its kids.

INTERNAL-AUTH-PROGRAM authscheme program argv[0] argv[1] ...
Specifies the program to run to set up authentication for the specified authentication scheme. "program argv[0] ..." gives the program to run and its arguments, in the same form as the EXEC keyword.

INTERNAL-AUTH-INPUT authscheme
Specifies the data to be given to the authorization program as its standard input. Each argument is passed as a single line. $n, where n is a number, is replaced by the n'th argument to the

"AUTH authscheme arg1 arg2 ..." line.

INTERNAL-PRINT arbitrary text

Prints its arguments as a Debug message. Mostly for *rstartd* debugging, but could be used to debug config files.

NOTES

When using the C shell, or any other shell which runs a script every time the shell is started, the script may get run several times. In the worst case, the script may get run **three** times:

By rsh, to run *rstartd*
By *rstartd*, to run the specified command
By the command, eg *xterm*

rstartd currently limits lines, both from config files and requests, to BUFSIZ bytes.

DETACH is implemented by redirecting file descriptors 0,1, and 2 to /dev/null and forking before executing the program.

CMD is implemented by invoking $SHELL (default /bin/sh) with "-c" and the specified command as arguments.

POSIX-UMASK is implemented in the obvious way.

The authorization programs are run in the same context as the target program - same environment variables, path, etc. Long term this might be a problem.

In the X context, GENERIC-CMD Terminal runs xterm. In the OpenWindows context, GENERIC-CMD Terminal runs cmdtool.

In the X context, GENERIC-CMD LoadMonitor runs xload. In the OpenWindows context, GENERIC-CMD LoadMonitor runs perfmeter.

GENERIC-CMD ListContexts lists the contents of @List in both the system-wide and per-user contexts directories. It is available in all contexts.

GENERIC-CMD ListGenericCommands lists the contents of @List in the system-wide and per-user commands directories, including the per-context subdirectories for the current context. It is available in all contexts.

CONTEXT None is not implemented.

CONTEXT Default is really dull.

For installation ease, the "contexts" directory in the distribution contains a file "@Aliases" which lists a context name and aliases for that context. This file is used to make symlinks in the contexts and commands directories.

All **MISC** values are passed unmodified as environment variables.

One can mistreat *rstartd* in any number of ways, resulting in anything from stupid behavior to core dumps. Other than by explicitly running programs I don't think it can write or delete any files, but there's no guarantee of that. The important thing is that (a) it probably won't do anything REALLY stupid and (b) it runs with the user's permissions, so it can't do anything catastrophic.

@List files need not be complete; contexts or commands which are dull or which need not or should not be advertised need not be listed. In particular, per-user @List files should not list things which are in the system-wide @List files. In the future, perhaps ListContexts and ListGenericCommands will automatically suppress lines from the system-wide files when there are per-user replacements for those lines.

Error handling is OK to weak. In particular, no attempt is made to properly report errors on the exec itself. (Perversely, exec errors could be reliably reported when detaching, but not when passing the stdin/out socket to the app.)

If compiled with -DODT1_DISPLAY_HACK, *rstartd* will work around a bug in SCO ODT version 1.(1.1?) (The bug is that the X clients are all compiled with a bad library that doesn't know how to look host names up using DNS. The fix is to look up a host name in $DISPLAY and substitute an IP address.) This

is a trivial example of an incompatibility that *rstart* can hide.

SEE ALSO

rstart(1), rsh(1), A Flexible Remote Execution Protocol Based on **rsh**

AUTHOR

Jordan Brown, Quarterdeck Office Systems