The Rembo Wizard 2.0

KickStart Support

How To

Petri Mäkijärvi

May 19, 2006

Abstract:
The Rembo Wizard 2.0 is a free plug-in module for the Rembo Toolkit 2.0, a PXE-enabled, Pre-OS platform for the system hard disk management for Windows and Linux PC-computers.
This document explains how system administrator can use The Rembo Wizard to plan and execute fully automated installation (FAI) of Red Hat Enterprise Linux 4 WS using PXE, DHCP and RedHat KickStart installation program.

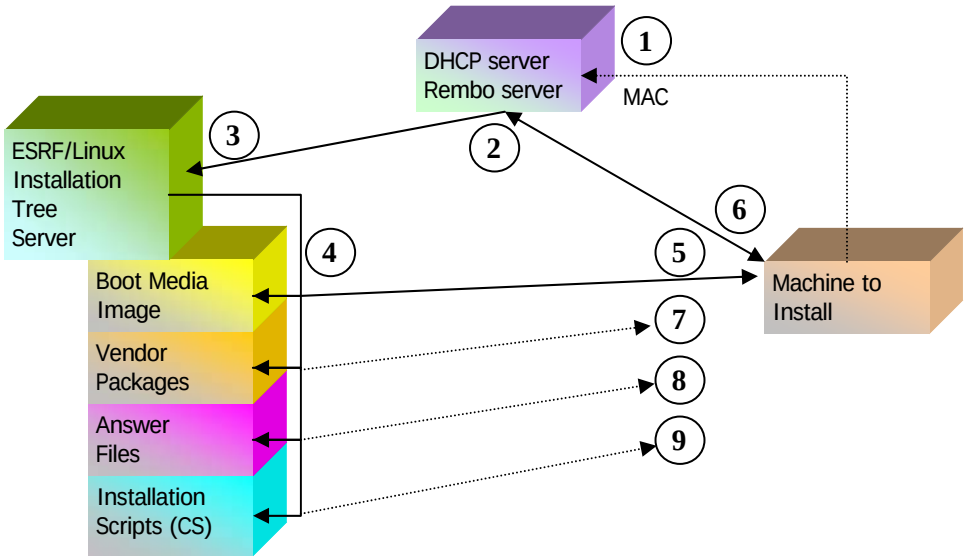
Introduction

- The Rembo Wizard is a free disk management plug-in module for the Rembo Toolkit.
- ESRF/Linux is a system deployment and version management scheme for Red Hat Enterprise Linux 4 WS installations used at the European Synchrotron Radiation Facility.
- KickStart is a Fully Automated Installation (FAI) method developed by Red Hat.

The Rembo Wizard package contains an additional plug-in module that allows the system installation to be started with the KickStart method. The launch of the KickStart installation is somewhat specific to the ESRF/Linux version management scheme. Nothing prevents to use, however The Rembo Wizard's KickStart support to launch any Red Hat Enterprise Linux installation using KickStart.

This document gives detailed information about the KickStart support in the Rembo Wizard. The given information helps the administrators of the ESRF/Linux to understand the configuration aspects of the Rembo Wizard. For everybody else the information allows the system administrator to adapt the KickStart infrastructure to The Rembo Wizard's KickStart support.

Operating principle



1. PXE boot request is sent to the DHCP server
2. The machine is started with The Rembo Wizard using PXE network boot
3. Available ESRF/Linux (RedHat) versions are stored on a (ESRF/Linux) Installation Tree Server (ex. *myserver1-1.mycompany.com*)
4. ESRF/Linux (RedHat) versions are stored on the server as a set of KickStart installation CD-ROMs
5. The Rembo Wizard illustrates a simple menu that allows the administrator to select either the default installation or to install an earlier version of the ESRF/Linux (RedHat).
    - A selection between different installation types, such as "workstation" or "server" is also displayed.
6. Once the installation is launched The Rembo Wizard searches a KickStart Linux kernel from the Rembo server and launches it
    - KickStart Linux kernel is fetched from a local network Rembo server

- Once launched it works with a centralized ESRF/Linux (RedHat) Installation Tree Server using HTTP-protocol
- It gets all its arguments about which installation to do, which server to use and other parameters from The Rembo Wizard

7. Vendor (*RedHat*) packages (*.rpm files*) gets installed from the selected ESRF/Linux (RedHat) version
8. Instead of asking the person who is doing the installation about specific decisions regarding the hard disk partitioning, network configuration, KickStart Answer Files contains the configuration information adapted to organization's infrastructure
9. Once the system is up and running, the person who is doing the installation can run the post-installation scripts which further tailor the system to better adapt to organization's infrastructure according to the specific function of the machine

Those who are familiar with the RedHat KickStart installation method will spot the missing component of PXELinux in the above scheme. The advantages of using the existing DHCP server and Rembo Toolkit server infrastructure instead of centralized PXELinux/KickStart server are:

- DHCP/PXE/Rembo services can be distributed on sub-networks, accessible directly from the machines to be installed. This is would avoid the need to route low level network protocols to a centralized KickStart server
- Centralized KickStart server (ESRF/Linux Installation Tree server) requires but one network service for file distribution. Usually an Apache server is installed to provide HTTP services on the organization's network. The HTTP protocol is usually routed everywhere in the enterprise's intranet network.

Activating the KickStart support module

First step in the KickStart support activation is to do the configuration of The Rembo Wizard for the host to be installed. From this point on the activation of the KickStart support module can be done by adding the following declaration into the *autoload* file of the host to be installed:

```
int KickStart = (int)"1";
```

Please note that the activation is done at ***host level***, *i.e.* in an *autoload* file that it located in the directory named with the system's MAC address. You can define the KickStart support activation at ***group level*** (and even at global level) to say "*All new hosts that belongs to this group should be installed using the KickStart*".

int KickStart	*Description*
0	No KickStart module interaction
1	Start of KickStart module in interactive mode
2	KickStart installation's reboot coming next
3	KickStart installation terminated
101	KickStart installation has failed
1000	Launch KickStart automatically with default architecture (Fully Automated Installation, FAI)
1001	Launch KickStart automatically with x86 (32-bit) installation
1002	Launch KickStart automatically with x86_64 (64-bit) installation

Table 1 - KickStart states

The FAI option of the KickStart support module can be achieved by using the special declaration

```
int KickStart = (int)"1000";
```

With this declaration, no questions will be asked but the default version and default configuration type gets installed on the target host, allowing the installation to take place without the presence of the system administrator. Because the installation method is potentially dangerous, it is available only at the ***host level***. Declaring the FAI option at any level, will clear the *KickStart* declaration both at group and at global levels.

> **Tip:** KickStart is not designed to mass installations. A much better strategy would be to install a master system with KickStart and then clone it using the disk image installation techniques available in The Rembo Wizard. The most important aspect of this technique would be to use Rembo Toolkit's Multicast protocol possibilities to limit the network bandwidth usage.

Configuring the Installation Tree server

This chapter explains what is required on the Installation Tree server side that it would work with The Rembo Wizard's KickStart support module. An existing ESRF/Linux Installation Tree Server is used as an example. Although you are not installing from an ESRF/Linux managed Red Hat server, please continue reading because it is relatively easy to modify any file system to be compatible with The Rembo Wizard KickStart support module.

Making the KickStart installation tree available

An excerpt from the Red Hat Enterprise Linux System Administration Guide: "*The Kickstart installation must access an installation tree. An installation tree is a copy of the binary Red Hat Enterprise Linux CD-ROMs with the same directory structure.*".

The above is of course true for The Rembo Wizard's KickStart support module as well. The recommended protocol for the KickStart installation is HTTP, and in this explanation we will talk about configuring an Apache server. Other protocols, such as FTP can be used but they will not be discussed in this document.

In an intranet environment Apache server's security settings can be relaxed. Notably it is much easier to point to the ESRF/Linux (Red Hat) distribution tree, when the *FollowSymLinks* directive is set for the *httpd* Apache daemon (see Options section of the Apache manual).

If symbolic links can be used with the web server, the actual installation tree can be located anywhere on the server's file system. Let's suppose that it has been installed on a file system, such as

/dist/mykickstart/rhel4ws

In the above path *mykickstart* is your specific version of the Red Hat Enterprise Linux CD-ROMs.

The Rembo Wizard's KickStart support module makes reference to the distribution tree using a format

r*[release #]****v****[version #]****p****[patch level #]*.

For example, *r1v0p0* for the distribution release 1.0.0.

You do not need to set up a version numbering scheme in order to use The Rembo Wizard's KickStart support module. In this case you would use a symbolic link to your actual installation tree and tell

KickStart support module that the version 1.0.0 is the one and the only version available, as explained in the configuration file example in *Appendix A: kickconfig.rbc for single version distribution.* Supposing that you have the Apache server installed on a Red Hat Enterprise Linux server, you would have to create a symbolic link to your installation tree:

cd /var/www/html
ln –s /dist/mykickstart/rhel4ws r1v0p0

Now the KickStart installation program can have an access to your installation tree using HTTP protocol.

Make available a KickStart answer file

You would need to make a KickStart answer file (or files) available within the installation tree. In our example, we would create a directory *mycfg* and place a single answer file in it for a default workstation configuration. The full path to the file in our example would be

/dist/mykickstart/mycfg/workstation.cfg

Please refer to the Red Hat's documentation about the KickStart file to get information how to create the answer file.

Configuring the Rembo Toolkit server

Configuring The Rembo Wizard's KickStart support module

After the installation of The Rembo Wizard 2.0.10 (or greater) the Rembo Toolkit server has a new directory, called *kickstart* at the global level. It contains an example configuration file, called *kickconfig_example.rbc* (which can be found from the distribution files as well). Copy the example under the name *kickconfig.rbc* and edit it. The configuration options are self explanatory.

The configuration example is for a rather complicated versioning scheme with multiple releases and answer file types. See below (*Appendix A: kickconfig.rbc for single version distribution*) for the most simple configuration example.

Installing the KickStart ISO-Linux kernel and initd-ramdisk

Once The Rembo Wizard's KickStart support module is ready to launch the installation, it will start the KickStart installation procedure by launching the ISO-Linux kernel from the KickStart installation CD-ROM. We need to copy the kernel and its initd-ramdisk on the Rembo Toolkit server and give the files specific names. Using our example installation tree path, we would have to copy

Full example installation tree path	*Path on the Rembo Toolkit server*
/dist/mykickstart/rhel4ws/isolinux/vmlinuz	kickstart/kslinuz_r1v0p0
/dist/mykickstart/rhel4ws/isolinux/initrd.img	kickstart/ksinitrd_r1v0p0

Table 2 - ISO-Linux kernel and ramdisk names

The above table illustrates the naming convention used to search for ISO-Linux kernel and its initrd-ramdisk. If you would have but one version, you would always use the extension (*_r1v0p0*). For multiple versions, there must be an ISO-Linux/initrd pair available for each version, with appropriate extension (*ex. _r1v1p3* for *v1.1.3*).

❖

Appendix A: *kickconfig.rbc* for single version distribution

In the below example, the example configuration file has been modified so that it contains but a single version (1.0.0) and a single KickStart configuration type (workstation.cfg). You do not want to support 64-bit versions for the time being so we would just skip it in the configuration file. Your modifications would be marked with blue color in the below configuration file example.

```
/* The Rembo Wizard KickStart Configuration File (Example) */

/* --------------- do not touch starts ---------------- */
bool kickconfigLoaded;
if (kickconfigLoaded)
     goto kickconfigEnd;
Printf("$Id: kickconfig_example.rbc,v 1.2 2006/05/18 13:49:10 petri Exp $<br>");
var kcSelect[], kcServers[];
var kcDefaultRelease[], kcDefaultVersion[], kcDefaultPatch[];
int kcRamSize;
int kcDefaultRelease_a0,kcDefaultVersion_a0,kcDefaultPatch_a0;
int kcDefaultRelease_a1,kcDefaultVersion_a1,kcDefaultPatch_a1;
int kcDefaultRelease_a2,kcDefaultVersion_a2,kcDefaultPatch_a2;
int kcNofPatch_r1_a0[],kcNofPatch_r2_a0[],kcNofPatch_r3_a0[];
int kcNofPatch_r1_a1[],kcNofPatch_r2_a1[],kcNofPatch_r3_a1[];
int kcNofPatch_r1_a2[],kcNofPatch_r2_a2[],kcNofPatch_r3_a2[];
var kcVerPatch_a0[],kcVerPatch_a1[],kcVerPatch_a3[];
str kcServerIPs_r1_a0[],kcServerIPs_r2_a0[],kcServerIPs_r3_a0[];
str kcServerIPs_r1_a1[],kcServerIPs_r2_a1[],kcServerIPs_r3_a1[];
str kcServerIPs_r1_a2[],kcServerIPs_r2_a2[],kcServerIPs_r3_a2[];
var kcArchServers_a0[],kcArchServers_a1[],kcArchServers_a2[];
str kcWelcomeText,kcDefaultConfig,kcProtocol, kcHostIP,kcDNSIP;
str kcConfigFiles[],kcConfigNames[],kcArchNames[];
str kcConfigRelPath,kcDefaultArch;
with (kickconfigErrorHandler) try {
/* --------------- do not touch ends ---------------- */

// Welcome text (ex. your company name) (string)
kcWelcomeText = "My Company's Name";

// Default architecture to install, if empty "default default" used (string)
kcDefaultArch = "";

// Default configuration file name to use (string)
kcDefaultConfig  = "workstation.cfg";

// Relative path to reach the configuration file name (string)
kcConfigRelPath = "mycompanydist/mycnf";

// List of available configuration files (must contain the Default above)
// The below list corresponds to "default default" system architecture.
// For other architectures you need to provide, ex. "workstation_x86_64.cfg".
// You need to have one dot (".") (but no more) in the file name.
kcConfigFiles = { "workstation.cfg" };

// Explicative names for the above list of configuration files
kcConfigNames = { "Workstation" };

// Protocol to reach the Kickstart servers (ex. http://) (string)
kcProtocol = "http://";

// Target's IP address (usually leave this to "dhcp") (string)
kcHostIP = "dhcp";

// Domain name server's IP address (ex. 192.168.10.12) (string)
kcDNSIP = "192.168.10.12";

// Kickctart RAM disk size (MB) (int)
kcRamSize = 8192;

// List of available target architectures (first one is "default default")
// Note: maximum target architectures is three (3)
```

```
kcArchNames = { "x86",          // Architecture 0
                "x86_64" };     // Architecture 1 (leave this for the future)

// ------------- ARCHITECTURE 0 : x86 ------------------
// Number of available patch levels in release 1
kcNofPatch_r1_a0 = { 0 };    // version 0 has patch levels 0

// Number of available patch levels in release 2
// kcNofPatch_r2_a0 = { 5,      // version 0 has patch levels 0,1,2,3,4,5
//                      3 };    // version 1 has patch levels 0,1,2,3

// Default internal release.version.patch number to install (int)
kcDefaultRelease_a0 = 1;
kcDefaultVersion_a0 = 0;
kcDefaultPatch_a0   = 0;

// Release 1: Kickstart server IP-addresses (ex. 192.168.1.102) (string)
kcServerIPs_r1_a0 = { "192.168.1.102" };    // version 0 server IP-address

// Release 2: Kickstart server IP-addresses (ex. 192.168.1.105) (string)
// kcServerIPs_r2_a0 = { "192.168.1.105",      // version 0 server IP-address
//                       "192.168.1.106" };    // version 1 server IP-address

// ------------- ARCHITECTURE 1 : x86_64 ------------------
// NOTE: We define this just as with the x86 above, just in case.
// Note: it is recommended (but not required) to have same number of
//       of versions and patch levels as in the default architecture
// Number of available patch levels in release 1
kcNofPatch_r1_a1 = { 0 };    // version 0 has patch levels 0

// Number of available patch levels in release 2
// kcNofPatch_r2_a1 = { 5,      // version 0 has patch levels 0,1,2,3,4,5
//                      3 };    // version 1 has patch levels 0,1,2,3

// Default internal release.version.patch number to install (int)
kcDefaultRelease_a1 = 1;
kcDefaultVersion_a1 = 0;
kcDefaultPatch_a1   = 0;

// Release 1: Kickstart server IP-addresses (ex. 192.168.2.102) (string)
kcServerIPs_r1_a1 = { "192.168.2.103" };    // version 0 server IP-address

// Release 2: Kickstart server IP-addresses (ex. 192.168.2.105) (string)
// kcServerIPs_r2_a1 = { "192.168.2.105",      // version 0 server IP-address
//                       "192.168.2.106" };    // version 1 server IP-address

// ---------- Collect the architecture definitions together ----------
// Uncomment/comment according to the actual number of releases
// and architectures

// KickStart server tables
kcArchServers_a0[0] = kcServerIPs_r1_a0;    // release 1, default architecture
// kcArchServers_a0[1] = kcServerIPs_r2_a0;    // release 2, default architecture
kcArchServers_a1[0] = kcServerIPs_r1_a1;    // release 1, second architecture
// kcArchServers_a1[1] = kcServerIPs_r2_a1;    // release 2, second architecture

kcServers[0] = kcArchServers_a0;    // default architecture
kcServers[1] = kcArchServers_a1;    // second architecture

// Version and patch level tables
kcVerPatch_a0[0] = kcNofPatch_r1_a0;  // release 1, default architecture
// kcVerPatch_a0[1] = kcNofPatch_r2_a0;  // release 2, default architecture
kcVerPatch_a1[0] = kcNofPatch_r1_a1;  // release 1, second architecture
// kcVerPatch_a1[1] = kcNofPatch_r2_a1;  // release 2, second architecture

kcSelect[0] = kcVerPatch_a0;    // default architecture
kcSelect[1] = kcVerPatch_a1;    // second architecture

kcDefaultRelease[0] = kcDefaultRelease_a0;
// kcDefaultRelease[1] = kcDefaultRelease_a1;
kcDefaultVersion[0] = kcDefaultVersion_a0;
// kcDefaultVersion[1] = kcDefaultVersion_a1;
```

```
kcDefaultPatch[0]   = kcDefaultPatch_a0;
kcDefaultPatch[1]   = kcDefaultPatch_a1;

//* --------------- do not touch starts ---------------- */
} // try
kickconfigLoaded = true;
kickconfigEnd:    ;
/* --------------- end of module - add nothing below ---------------- */
```