`kboot_sl` Linux System Loader Description

Michael Löhr and Swen Schillig

$Id: description.lyx,v 1.1.2.1 2006/06/27 14:41:41 loehr Exp $

Contents

1 Overview

This document describes the installation and configuration of the system loader. All features and options of the system loader configuration are explained in detail. In addition, the usage of the user interface is outlined and example configurations are provided in the appendix.

2 Description

As an extension of the `kboot` project by Werner Almesberger (kboot homepage `http://kboot.sourceforge.net/`) the system loader runs in a minimal Linux environment started by a platform specific first stage boot loader. At the start a configuration file is read, the setup is performed according to the configuration data and finally the boot selection menu is displayed. Based on the users selection, a boot configuration is loaded and the system is reinitialized via the `kexec` system call. As the result the minimal Linux system is replaced by the system selected in the boot configuration.

Up to this level of description `kboot` and `kboot_sl`, as the system loader extension will be called in the following text, offer similar functionality. From a more detailed view `kboot_sl` has been optimized for a different goal than the original `kboot` environment. The original `kboot` tries to provide a compact and minimal Linux boot system that will be built and customized for the target machine and application. `kboot_sl` tries to offer a more generic and feature rich Linux boot environment that can be provided in a ready to use form, e.g. as a rpm-package but gives up the goal to provide a very small boot environment.[1]

`kboot_sl` can be seen in the following ways:

1. *As a comfortable boot loader for systems that offer only a frugal first stage boot loader:*

   From this point of view `kboot_sl` solves the problem that some boot loaders have to be reinitialized whenever a component of the boot configuration changes. Examples for this type of boot loaders are `lilo` on the i386 platform or `zipl` on the s390 platform. With `kboot_sl` these boot loaders will be initialized only once to start `kboot_sl`. Any further changes to the boot configuration will be handled dynamically by `kboot_sl`. A potentially uncomfortable boot menu like in the case of the `zipl` boot loader is replaced by the more convenient boot selection menu of `kboot_sl`.

2. *As a boot loader that can be controlled from remote via the network:*

   Usual boot managers offer their boot seletion menu on the local console of the system to be booted. Therefore it is necessary to walk to the machine or to connect with a special terminal software if anything different than the default configuration has to be booted. With `kboot_sl` a standard ssh interface allows remote connections while a system is still in its boot phase.

3. *As a boot loader that supports boot configurations to be centralized and managed on the network:*

   Typical boot loaders have to be configured locally on the system to be booted. If kernel parameters or the kernel itself have to be changed on several systems an administrator has to login to all systems in order to change the settings or install the kernel. `kboot_sl`'s ability to access remote files through a network connection together with a mechanism that can handle system specific configurations is the basis to consolidate the boot configuration for several systems. Boot parameters, kernels, ramdisks, etc. can be provided by a centralized boot server. Especially in virtualization or cluster environments with many similar systems this allows to manage and change boot configurations very efficiently. The approach helps also to distribute new kernel versions with important bug fixes fast and keeps all systems booting from the same boot server in a consistent state.

[1] Dennis Merbach was so friendly to host our collected Linux system loader patches at `http://www.merbach.net/kboot_sl/`. Future additions and extensions may also appear on this site.

4. *As a rescue system that is always available in the boot sequence:*

   If the boot of a Linux system fails, a Linux rescue system that allows to fix the problem is quite useful. On a system that uses `kboot_sl` as its boot manager there is already a rescue system installed. From the shell environment of `kboot_sl` the problem that prevents the system from booting can be analyzed and fixed. As soon as the problem is solved the system can be booted into normal operation again. Special tools and requirements for a customized rescue system can be easily integrated into the `kboot_sl` ramdisk via the `kboot_admin` tool.

3 Building system loader executables

In most cases the end user of the system loader `kboot_sl` will use prebuilt components and has therefore no need to build the system loader from scratch. If it is necessary to rebuild the system loader components there are three possibilities.

3.1 The `kboot` Build Environment

The system loader is an extension to `kboot` and therefore integrated into the `kboot` build environment. As a result the execution of the `make` command downloads and compiles all components required by `kboot` and the system loader extension. Finally, `make` creates a system loader kernel and a ramdisk. This ramdisk contains the kboot base environment and all system loader add-ons, which are detailed out further down in this document. Additional information about the build environment can be found in the original `kboot` documentation by Werner Almesberger.

3.2 RPM Package Build

This build option is only available on RPM based Linux distributions. Executing `make redhat-rpm` in the `kboot` top level directory will first execute `make` in the `kboot` build environment and finally create source and binary RPM packages of the system loader.

3.3 System Loader Standalone Build

It is possible to build the system loader as a stand-alone application without the `kboot` kernel and ramdisk environment. Executing `make install` in the sub-directory `kboot/ui` is creating a stand-alone version of the system loader. This version can be used for debugging or to initiate a fast menu controlled reboot of the system. To ensure that the system loader will find all its components the environmant variable `KBOOT_PATH` has to be set similar to the following example:

```
export KBOOT_PATH=/root/systemloader/kboot/ui/usr/kboot
/root/systemloader/kboot/ui/usr/kboot/kboot -u linemode file:///boot/kboot.conf
```

4 Installation

There are two ways to install the system loader, automatically via a RPM-package (on supported systems only) or manually by copying the system loader kernel and ramdisk to the `/boot` directory. In both cases the system loader is not activated automatically.

Instead of using the kernel provided with the `kboot` package it is also possible to use any other kernel available on the system. However, the kernel must be at least version 2.6.13 and the kexec system call has

to be supported. In the simplest scenario all required drivers (e.g. disk or network drivers) are compiled into the kernel. For more flexibility the `kboot_admin` tool (see section 8) allows to create and modify customized system loader ramdisks with module support. Kernels and ramdisks are different for 31/32-bit and 64-bit systems and must not be mixed.

To enable the system loader it has to be configured as a boot entry of the first stage boot loader. On Intel `lilo` or `grub` can be used, on s390/System z the system loader can be booted from the reader or use `zipl` as the first stage loader. Any minimal first stage loader is sufficient because the system loader will handle the boot selection menu and user interaction.

Before the system loader can be used, an appropriate configuration file has to be created. A detailed description of all elements of the system loader configuration file is given in section 6. Example configuration files can be found in sections A.2 and A.3.3. When the system loader configuration file is available the system loader kernel and initrd can be booted via the first stage boot loader. Sections A.1 and A.3.1 show example configurations for `grub` and `zipl`.

`kboot_sl` will be started if the kernel command line contains the additional parameter `kboot` that specifies the location of the configuration file. Otherwise the Werner Almesberger variant of the `kboot` environment will be started without the system loader extension.

Example:

```
kboot=dasd://(0.0.5c5e,1)/boot/kboot.conf
```

5 Boot Methods

Depending on the hardware platform and the available disk types the system loader supports several boot methods. On the i386 and s390 platform the system can be booted from a kernel image file. An initial ramdisk and a kernel command line or parmfile can be specified together with this kernel file. On the s390 platform `*.ins` files or the boot map information written by `zipl` can be used in addition.

In any case the location of kernel, ramdisk, parmfile, insfile or boot map will be defined as an URI. Which URI scheme has to be used depends on the system platform and available disk. The `block` URI is typically used on the i386 platform. The `dasd` and `zfcp` URI schemes are specific for the s390 platform. On every platform the `ftp` , `http` and `scp` URI schemes can be used to boot from the network.

All boot methods and URI schemes will be described in detail in the following chapter.

6 Elements of the Config File

The config file for `kboot_sl` consists of some global definitions which are followed by one or more boot entries. External files are always referenced by an URI. A detailed description of supported URIs will be given in section 6.5.

6.1 Global Definitions

Preceding the boot entries the system loader configuration file contains a number of definitions that are valid for the whole configuration file.

6.1.1 Comments

Comments are allowed in the global definitions part of the configuration file and in the boot entries. A comment starts with # and ends at the end of the line. Tabs or spaces in front of the # are possible but comments at the end of a line that contains a definition are not allowed.

Example:

```
# this is a comment
        # this is also a comment

<definition> # this is not allowed!
```

6.1.2 default

The `default` statement references the boot entry that will be selected automatically after the timeout. Valid references are the label of a boot entry or a number. Implicit numbering of the boot entries starts with 0. If the default statement is not used, entry number 0 will be used as the default.

Example:

```
default linux2
default 1
```

6.1.3 timeout

The `timeout` statement specifies the time in seconds until the default boot entry will be started automatically. If the statement is not used or the timeout is set to 0 no timeout will occur.

Example:

```
timeout 15
```

6.1.4 password

The `password` statement defines the password that has to be entered if a locked boot entry (see section 6.4.8) has been selected.

Example:

```
password topsecret
```

6.1.5 userinterface

The `userinterface` statement specifies which userinterface module should be started to display the boot selection menu. For every userinterface command a userinterface process will be started. This allows to start several instances of the same userinterface module listening on different input devices or ports. Additional options may be specified depending on the user interface module. Currently `linemode` and ssh are available as user interface modules.

Syntax:

```
userinterface <ui_module_name> [MODULE_OPTIONS]
```

6.1.6 userinterface linemode

For the linemode interface it is required to specify the device on which the boot selection menu will be displayed. Typical values for the device parameter are `/dev/console` , `/dev/tty` or `/dev/tty1` .

Syntax:

```
userinterface linemode <device>
```

Example:

```
userinterface linemode /dev/tty1
```

6.1.7 userinterface ssh

The `userinterface ssh` statement will start a process that is listening for incoming ssh connections in the background. For every incoming connection that is successfully established a new user interface instance will be started. The type of interface to be started has to be specified as the first module option. Currently only linemode can be specified here but for the future additional user interface types (e.g. ncurses) may become available. Additional module options may be used to specify the port on which the ssh connection can be established and to load RSA or DSS keys from a location defined by an URI scheme.

Connecting to the system loader via ssh will require a userid and password. During the creation of the system loader ramdisk a user kboot with password kboot will be created automatically. The tool `kboot_admin` allows to change the password on an existing ramdisk.

Syntax:

```
userinterface ssh <ui_module_name> [port=<portnumber>] [rsa_key=<key_uri>] [dss_key=<key_uri>]
```

Example:

```
userinterface ssh linemode port=2222
```

6.1.8 include

The `include` statement can be used anywhere in the system loader configuration file. The content of the file referenced by the URI will be included at the position of the `include` statement. Nested includes are allowed but limited to twenty levels. `include` statements that appear inside an inactive system section will not be applied.

Syntax:

```
include URI
```

Example:

```
include dasd://(0.0.5c5e,1)/boot/extra_menu.conf
```

6.2 Setup Commands

This group of commands allows to setup additional devices in the minimal Linux environment of the system loader. It is possible to load additional kernel modules, to specify network settings and to enable devices that are not enabled automatically. Each `setup` command is executed immediately. As parameters are identified by keywords they are not required to appear in any particular order. An additional short form allows to use `setup` commands on the kernel command line.

Syntax:

```
setup <setup_item> { <paramlist> }
```

6.2.1 `setup module`

The `setup module` command allows to load additional kernel modules. Module dependencies are resolved automatically. The module to be loaded has to be present on the system loader ramdisk. Modules can be added to an existing ramdisk via the kboot_admin tool.

Syntax:

```
setup module {
  name          <name>
  param         <param>
  kernelversion <kernelversion>
}
```

Short form:

```
mod(<name>[,<param>[,<kernelversion>]]
```

Example:

```
setup module {
  name qeth
}
```

6.2.2 `setup network`

The `setup network` command allows to specify network settings and to initialize a network device with these settings. Static setup and dhcp are supported. Parameters for static network setup can also be used in dhcp mode and are used as fallback if dhcp fails. A correct network setup is required to use network based URI schemes like ftp, http and scp. A `setup module` command may be required to load the network device driver before the `setup network` command can be executed successfully. The `kboot_admin` tool can be used to copy the required driver modules into the ramdisk of the system loader.

Syntax:

```
setup network {
  interface  <interface>
  mode       <dhcp_or_static>
  address    <address>
  mask       <mask>
  gateway    <gateway>
  nameserver <nameserver>
}
```

Short form:

```
static(<interface>[,<address>[,<mask>[,<gateway>[,<nameserver>]]]]
  dhcp(<interface>[,<address>[,<mask>[,<gateway>[,<nameserver>]]]]
```

Example:

```
setup network {
  interface  eth0
  mode       dhcp
  address    9.155.23.65
  mask       255.255.255.128
  gateway    9.155.23.1
  nameserver 9.64.163.21
}
```

6.2.3 `setup qeth`

On the s390 platform the `setup qeth` command can be used to enable a qeth ethernet device with the given busids.

Syntax:

```
setup qeth {
  busid  <busid>
  busid  <busid>
  busid  <busid>
}
```

Short form:

```
qeth(<busid>,<busid>,<busid>)
```

Example:

```
setup qeth {
  busid 0.0.f5de
  busid 0.0.f5df
  busid 0.0.f5e0
}
```

6.2.4 `setup dasd`

On the s390 platform the `setup dasd` command can be used to enable a dasd disk device with the given busid.

Syntax:

```
setup dasd {
  busid  <busid>
}
```

Short form:

```
dasd(<busid>)
```

Example:

```
setup dasd {
  busid 0.0.5c60
}
```

6.2.5 `setup zfcp`

On the s390 platform the `setup zfcp` command can be used to enable a zfcp disk device.

Syntax:

```
setup zfcp {
  busid  <busid>
  wwpn   <wwpn>
  lun    <lun>
}
```

Short form:

```
zfcp(<busid>,<wwpn>,<lun>)
```

Example:

```
setup zfcp {
  busid 0.0.54ae
  wwpn  0x5005076300cb93cb
  lun   0x512e000000000000
}
```

6.3 System Dependent Sections

Based on the network capabilities of the system loader it is possible to provide kernels, ramdisks and kernel parameters on a centralized bootserver. System dependent sections in the system loader configuration file provide a mechanism that is especially useful for shared system loader configuration files. A system dependent section starts with a test, that determines if the configuration part is active on a specific system. Depending on the result of this test the following section is used or ignored. The identification of the system can be done via the MAC address of a network adapter, the UUID of the system, the name of the virtual machine or the name of the logical partition of the host system. A system dependent section is active if at least one of the system identifiers matches. If a system identifier is not available on the specific system it will never match. System dependent sections can be nested in order to specify settings that are common to a number of systems and to be able to change only some details for the specific system.

Syntax:

```
system <systemidlist> {
  <definitionlist>
}
```

Examples:

```
system mac(00:10:C6:DE:12:6C)
       uuid(C7CCA781-2DD5-11C6-93BC-AD439DC0988B)
{
  root block://(/dev/hda5,ext3)/boot/
}
system not(vmguest(linux41)) {
  include ftp://kboot:kboot@53v15g41.ibm.com/boot/standard_entries.conf
}
system vmguest(linux40,g53lp15)
       vmguest(linux41,g53lp15)
{
  setup network {
          mode static
          vmguest(linux40) {
            address   9.152.26.120
          }
          vmguest(linux41) {
            address   9.152.26.121
          }
          mask    255.255.252.0
          gateway 9.152.24.1
          nameserver 9.152.120.241
          interface eth0
  }
}
```

6.3.1 mac

The mac systemidentifier can be used on nearly every system that has a network adapter. The mac statement matches if the system has a network adapter with this mac address. It may be necessary to load a kernel module and to enable the network device before the MAC of the system can be determined. Therefore it is not the preferred method to identify a system via its mac. Nevertheless it is the only available method for older machines on the i386 platform.

6.3.2 uuid

On the i386 platform recent BIOS versions provide a UUID that allows to identify the specific system unambiguously. The `uuid` command matches when the given UUID matches the UUID of the system. If available this is the preferred method to identify a specific system on the i386 platform.

6.3.3 lpar

The concept of logical partitions allows to split the hardware of a computer system into several independent parts. Each part or logical partition can boot its own operating system. The `lpar` system identifier allows to identify the logical partition on which the system loader configuration file is currently used and matches when the given name matches the name of the current logical partition. In the system loader it is implemented for the s390 platform. In the future it could be extended to other platforms that offer logical partitions.

6.3.4 vmguest

The concept of virtual machines is an approach that allows to run several instances of operating systems under the control of a virtualization software. The `vmguest` system identifier allows to identify the virtual

machine on which a system loader configuration file is currently used. It matches when the given name matches the name of the virtual machine. An extended form allows to specify the name of a logical partition in addition and matches when the name of the virtual machine and the name of the logical partition are identical to the given parameter values. The `vmguest` system identifier is implemented for VM on the the s390 platform. In the future it could be extended to other environments that offer virtual machines.

6.4 Boot Entry Definitions

The global definitions are followed by a section of boot entries which describe the available boot configurations. A boot entry starts with the line

```
boot_entry {
```

and ends with a closing bracket

```
}
```

Every boot entry must contain exactly one statement that triggers a boot action. These boot action statements are `kernel`, `insfile`, `bootmap`, `halt`, `reboot` and `shell`. Other statements can be used optionally. Some of them will specify labels or variables to improve the handling of the configuration file, some will influence the behaviour of the boot menu and some will pass additional information to the booted system. All possible statements inside the boot entry are described in the following sections.

6.4.1 `title`

The `title` statement defines a text that will be displayed in the boot selection menu and should give some meaningful description of the boot configuration. This statement is mandatory for all boot entries.

Example:

```
title Debian GNU/Linux, latest kernel
```

6.4.2 `label`

The `label` statement allows to assign a symbolic label to a boot entry. This label can be used in the default statement to reference the default boot entry (see section 6.1.2). If no label definition is present the boot entry can still be referenced numerically by its position in the configuration file.

Example:

```
label linux3
```

6.4.3 `root`

The `root` statement defines an URI path prefix and will be prepended to all URIs[2] specified in the same boot entry. Typically it is used to specify a common path for kernel, initrd and parmfile. Note that the path prefix and the rest of the URI will be concatenated as they are specified. There is no automatic insertion of a `'/'` character and no syntax checking.

Example:

```
root dasd://(0.0.5c5e,1)/boot/
```

[2]The root definition will not be applied to `bootmap` URIs because a path definition makes no sense for a structure that is not part of any filesystem.

6.4.4 kernel

The `kernel` statement specifies the kernel file that will be booted if this boot entry is selected. All supported URI formats are allowed to specify the location of the kernel file. If a root statement is given in the same boot entry, it will be prepended to the specified kernel path.

Example:

```
kernel vmlinuz
```

6.4.5 initrd

The `initrd` statement specifies the initial ramdisk that will be used if this boot entry is selected. All supported URI formats are allowed to specify the location of the ramdisk file. If a root statement is given in the same boot entry, it will be prepended to the specified ramdisk path.

Example:

```
initrd initrd.img
```

6.4.6 cmdline

The cmdline statement specifies the kernel commandline that is used to start the kernel if this boot entry is selected.

Example:

```
cmdline ro root=/dev/ram0 ramdisk_size=100000
```

6.4.7 parmfile

As an alternative to the cmdline statement the `parmfile` statement can be used to specify the file that will be used as the kernel command line if this boot entry is selected. All supported URI formats are allowed to specify the location of the parmfile. If a root statement is given in the same boot entry, it will be prepended to the specified parmfile path. If parmfile and cmdline are specified at the same time they will be concatenated as `parmfile + ' ' + cmdline` .

Example:

```
parmfile parmfile.txt
```

6.4.8 lock

The `lock` command allows to have password protected boot entries. If a locked boot entry is selected the user has to enter the password specified in the `password` statement (see section 6.1.4) to execute this boot selection entry.

Example:

```
lock
```

6.4.9 pause

The pause statement displays a message and waits for user input before the boot entry will be started. This can be used to ask the user to prepare the system for booting (e.g. by inserting a boot CD to the CD drive).

Example:

```
pause Please insert your boot floppy!
```

6.4.10 insfile

The insfile statement can be used as an alternative method to specify a boot configuration. An *.ins file contains the definitions of a kernel, initial ramdisk and parmfile, therefore it makes no sense to specify these components . The insfile statement is only available on the s390 platform and is provided with several Linux distributions for this platform.

Example:

```
insfile dasd://(0.0.5c5e,1)/usr/local/insfile_test/redhat/generic.ins
```

6.4.11 bootmap

The bootmap command can be used as an alternative method to specify a boot configuration. It is only available on the s390 platform and boots the system using the boot information from the boot map of the specified disk. This boot information is written by the tool zipl. Only bootmaps in the format created by zipl version 1.2 or newer are supported. The bootmap command uses a modified URI format that is described in section 6.5.8 and 6.5.9.

Example:

```
bootmap dasd://(0.0.5e2a,0)
```

6.4.12 halt

The halt statement can be used instead of a real boot selection. It will halt the Linux environment of the system loader.

6.4.13 reboot

The reboot statement can be used instead of a real boot selection. This statement will reboot the system. If a first stage bootloader is installed the system will be restarted via this bootloader.

6.4.14 shell

The shell statement can be used instead of a real boot selection. If the resulting boot entry is selected, the user will get a shell prompt. Leaving the shell prompt via exit will redisplay the boot selection menu.

6.5 URI Definitions

URIs provide a generic mechanism to describe the location of files. They are used by the system loader to locate boot and configuration files. The following sections describe the supported URI schemes in detail.

6.5.1 block

The block URI can be used to reference a file on any block device containing a supported filesystem. This is the typical method to access the boot files on the i386 platform and on other non s390 platforms. If no filesystem type is specified, the filesystem type will be autodetected.

Syntax:

```
block://(<device node>[,<filesystem type>])/<path to file>
```

Example:

```
block://(/dev/hda5,ext3)/boot/vmlinuz
```

6.5.2 ftp

The ftp URI can be used to reference a file on a ftp-server. This URI type is available on all platforms but requires a network connection. If the remote server requires a UID and a password, it has to be provided the usual way (see the example below). In case the UID contains the "@" , as it happens in e-mail addresses, it has to be replaced with "%40" to avoid any misunderstanding with the separator between the UID:PW combination and the host-name.

Syntax:

```
ftp://[UID[:PW]@]hostname[:PORT]/<path to file>
```

Example:

```
ftp://my_name%40ibm.com:my_secret@ftp.server.com/boot/vmlinuz
```

6.5.3 http

The http URI can be used to reference a file on a http-server. This URI type is available on all platforms but requires a network connection. If the remote server requires a UID and a password, it has to be provided like for the 6.5.2ftp URI (see example below). In case the UID contains the "@" , as it happens in e-mail addresses, it has to be replaced with "%40" to avoid any misunderstanding with the separator between the UID:PW combination and the host-name.

Syntax:

```
http://[UID[:PW]@]hostname[:PORT]/<path to file>
```

Example:

```
http://my_name%40ibm.com:my_secret@http.server.com/boot/vmlinuz
```

6.5.4 scp

The scp URI can be used to reference a file on a server which is accessible via the SSH protocol. As with the 6.5.2 ftp- and the 6.5.3 http-URI the scp-URI is available on all platforms but requires a network connection. The encoding of a UID and a password is identical to the above described protocols (see example below). In case the UID contains the "@" , as it happens in e-mail addresses, it has to be replaced with "%40" to avoid any misunderstanding with the separator between the UID:PW combination and the hostname. Beside the UID:PW combination, encoded in the URI, the public key authentication is supported as well. Therefor the local public key has to be part of the servers authorized_keys file and the public key authentication has to be enabled on the server.

Syntax:

```
scp://[UID[:PW]@]hostname[:PORT]/<path to file>
```

Example:

```
scp://my_name%40ibm.com:my_secret@ssh.server.com/boot/vmlinuz
```

6.5.5 file

The file URI can be used to reference files on an already mounted filesystem. This type of URI is available on all platforms. In an unmodified system loader boot environment it can only be used to reference files on the initial ramdisk. If the system loader is running as a program on an already booted Linux system it can be used instead of the other URIs.

Example:

```
file:///boot/vmlinuz
```

6.5.6 dasd

The dasd URI can be used to reference files on zSeries ESCON/FICON attached storage. This type of URI is only available on the s390 platform. If no filesystem type is specified, the filesystem type will be autodetected.

Syntax:

```
dasd://(<bus id>,[<partition>[,<filesystem type>]])/<path to file>
```

Example:

```
dasd://(0.0.5c5e,1)/usr/local/insfile_test/SuSE/suse.ins
```

6.5.7 zfcp

The zfcp URI can be used to reference files on s390 (zSeries) FCP attached storage. This type of URI is only available on the s390 platform. If no filesystem type is specified, the filesystem type will be autodetected.

Syntax:

```
zfcp://(<bus id>,<WWPN>,<LUN>,[<partition>[,<filesystem type>]])/<path to file>
```

Example:

```
zfcp://(0.0.54e0,0x5005076303000104,0x4011400500000000,1)/boot/initrd.img
```

6.5.8 `dasd` URI for the bootmap command

Because the bootmap information is not stored inside a file system the bootmap command (see section 6.4.11) uses a reduced form of the dasd URI.

Syntax:

```
dasd://(<bus id>[,<program number>])
```

Example:

```
dasd://(0.0.5e89,1)
```

6.5.9 `zfcp` URI for the bootmap command

Because the bootmap information is not stored inside a file system the bootmap command (see section 6.4.11) uses a reduced form of the zfcp URI.

Syntax:

```
zfcp://(<bus id>,<WWPN>,<LUN>[,<program number>])
```

Example:

```
zfcp://(0.0.04ae,0x500507630e01fca2,0x4010404500000000,2)
```

7 Using the Command Line Interface

The command line interface is the basic user interface of the system loader. It is designed to run on all platforms and without any special requirements regarding the capabilities of the user interface device. Therefore it will be usable on line mode interfaces like the 3270 terminal on the s390 platform or via a serial line on typical open systems platforms.

The command line interface displays the title information from all boot entries found in the system loader configuration file. The default entry is marked by an arrow symbol `->`, locked entries are displayed in brackets `[]`.

```
kboot user interface is starting.
Configuration file source: file:///boot/boot_menu.config

Welcome to kboot!
The following boot options are available:
-> 1    latest kernel from DASD
   2    latest kernel from EVMS (matching system)
   3    rescue kernel from DASD
   4    kboot (development version)
   5    Linux 2.6.13-14.x from FCP disk
   6    System Reboot
   7    System Halt
   8    start a shell!
   d<n> Display boot parameters of the selected entry
   m<n> Modify and boot selected entry
   i    Enter boot parameters interactively
Please enter your selection:
```

The selection of a specific entry is done by entering its number. If the menu is not completely visible it can be redisplayed by entering an empty input. If a timeout occurs while one or more user interfaces are waiting for input, all user interfaces will be terminated and the default entry will be executed. If a timeout is defined it will be displayed by the user interface. In addition to the selection of predefined menu entries the user interface allows to display existing boot entries, to modify them or to enter new entries from scratch.

Entering the boot entry number with prefix `d` will display the associated boot entry like shown in the following example.

```
d2

*** BOOT ENTRY PRINTOUT ***
title     latest kernel from EVMS (matching system)
label     evms
root      block://(/dev/evms/evmsvol)/boot/
kernel    vmlinuz
cmdline   dasd=5c60-5c61 root=/dev/dasda1 ro noinitrd selinux=0
action    KERNEL_BOOT
*** Press RETURN to continue ***
```

With prefix `m` the boot entry will first be displayed completely and will then be offered line by line to allow the input of modified lines. Empty user input will leave the respective line unchanged. The modified boot entry will be displayed again and can be booted finally.

```
m2

*** BOOT ENTRY PRINTOUT ***
title     latest kernel from EVMS (matching system)
label     evms
root      block://(/dev/evms/evmsvol)/boot/
kernel    vmlinuz
cmdline   dasd=5c60-5c61 root=/dev/dasda1 ro noinitrd selinux=0
action    KERNEL_BOOT

*** Please enter the new values for each line ***
*** or hit ENTER to leave it unchanged.       ***
root
kernel    vmlinuz.rescue
initrd
cmdline
parmfile

*** BOOT ENTRY PRINTOUT ***
root      block://(/dev/evms/evmsvol)/boot/
kernel    vmlinuz.rescue
cmdline   dasd=5c60-5c61 root=/dev/dasda1 ro noinitrd selinux=0
action    KERNEL_BOOT

*** Press 'B' to boot this entry or any other key to cancel ***
```

The `i` command allows to enter a boot entry interactively from scratch in cases where no similar entry is already present in the boot selection menu.

```
i

*** Please enter the values for manual boot***
ACTION   => 1->KERNEL_BOOT 2->INSFILE_BOOT 3->BOOTMAP_BOOT
action    1
root      block://(/dev/evms/evmsvol)/boot/
kernel    vmlinuz.rescue
initrd
```

```
cmdline   dasd=5c60-5c61 root=/dev/dasda1 ro noinitrd selinux=0
parmfile

*** BOOT ENTRY PRINTOUT ***
root      block:/(/dev/evms/evmsvol)/boot/
kernel    vmlinuz.rescue
cmdline   dasd=5c60-5c61 root=/dev/dasda1 ro noinitrd selinux=0
action    KERNEL_BOOT

*** Press 'B' to boot this entry or any other key to cancel ***
```

8 The `kboot_admin` Tool

The administration tool was created to modify a ramfs to suite all environment and user requirements while running the minimal system contained within the ramfs. The functionality includes the support to add any executable, library or module to the ramfs. In addition all dependencies are detected and resolved by adding those files to the ramfs as well.

E.g. if an added executable is dynamically linked and has therefor certain library dependencies, this will be detected and the required libraries will be copied to the ramfs.

The same applies for library (e.g. library_a requires library_b) and module (mod_a requires mod_b) dependencies. If modules are added to the ramfs the required modules.dep file is generated in the appropriate location.

The second field of application is to merge two ramfs files into one. This might be the case if a special utility ramfs exists and the extra features of this ramfs should be added to the default initial ramfs of a distribution (e.g.: merge the kboot ramfs with SuSEs initrd). The resulting ramfs contains all files from both sources whereas the content of the master ramfs, usually the distribution ramfs, takes precedence.

In addition the tool can be used to change the super-users password within the ramfs. This functionality is primarily used when the ramfs is supporting remote-access while running the minimal system.

The individual settings of the admin tool are configured through a configuration file which can be specified by the '-c <file>' command-line parameter. The default is /etc/kboot_admin.conf.

On first program initiation the configuration file is created automatically with default values for all settings.

```
GZIP         /usr/bin/gzip
FIND         /usr/bin/find
CPIO         /usr/bin/cpio
CUT          /usr/bin/cut
RSYNC        /usr/bin/rsync
GET_K_VERS   /sbin/get_kernel_version
MASTER_RAMFS /boot/initrd
KBOOT_RAMFS  /boot/kboot-root-glibc.cpio.gz
IMAGE_RAMFS  /boot/initrd-kboot.cpio.gz
TEMP_DIR     /tmp/KBOOT
############### the module dependent part ##################
KERNEL       /boot/vmlinuz
MODULE       e100
MODULE       reiserfs
############### executables and libraries ##################
EXEC         uname
EXEC         losetup
EXEC         ldd
LIBRARY      /lib/libpam_misc.so.0
```

These settings have to be modified manually to map the local requirements.

In addition the admin tool supports the following options and commands.

```
Commands:
  merge    join content from -s <ramfs> and -m <ramfs> to -o <ramfs>
  passwd   change password for super-user ID in -m <ramfs>
  add      add executables, libraries and/or modules to the ramfs
Options:
  -h|--help              this help text
  -v|--version           print version information
  -c|--config <file>     configuration file name (default /etc/kboot_admin.conf)
  -m|--master <file>     master ramfs superseding other ramfs' content.
  -s|--secondary <file>  second ramfs which is merged with the master ramfs.
  -o|--output <file>     file name for the resulting ramfs image
  -k|--kernel <file>     kernel file name
  -l|--lib <file>        add DLL(s) <file,...> to ramfs (full qualified path)
  -m|--module <module>   add module(s) <module,...> to ramfs (e.g. qeth )
  -x|--executable <file> add executable <file,...> to ramfs (e.g. losetup)
```

Notice: if one or more of the command-line switches `-l, -m, -x` are used, none of the configured file values are used for libraries, modules and executables. This means that either the configuration file or the command line is used to reference the files which are to be added to the ramfs.

To describe the admin tool's functionality best, verify the following examples:

1. merge two ramfs files into one and use all the settings provided by the configuration file test.conf .

   ```
   admin_tool merge
   ```

2. add the executables ldd and /home/frank/my_exec to the ramfs /boot/my_initrd and store the resulting ramfs in the default file.

   ```
   admin_tool -m /boot/my_initrd -x ldd,/home/frank/my_exec add
   ```

3. add all executables, libraries and modules specified in the configuration file /home/frank/my.conf and store the resulting ramfs in /home/frank/my_initrd.new .

   ```
   admin_tool -f /home/frank/my.conf -o /home/frank/my_initrd.new add
   ```

4. change the super-users password in the ramfs /boot/my_initrd and store the resulting ramfs in the same file.

   ```
   admin_tool -m /boot/my_initrd -o /boot/my_initrd passwd
   ```

9 Using Online Documentation

Man pages are available for the `kboot` executable and for the system loader configuration file `kboot.conf`. These man pages are meant as a short overview and refer to this specification and to the system loader design document for a detailed description of this software.

10 Extending the System Loader

The system loader is designed to be extensible. The extension can be done without changing the code of the existing components. User provided loader modules can provide support for additional URI schemes (e.g. nfs, smb). Additional user interface modules can provide more comfortable user interfaces (e.g. graphical or web-based).

A Examples

A.1 `menu.lst` to Start the System Loader via `grub`

This example shows how the system loader can be booted on the i386 platform using grub as the first stage boot loader. Relevant for the system loader is the `kboot=` parameter in the kernel command line which references the system loader configuration file. Without this parameter the unmodified kboot variant as provided by Werner Almesberger will be started. The `quiet` parameter minimizes the screen output while the system loader environment is started.

```
## default num
# Set the default entry to the entry number NUM. Numbering starts from 0, and
# the entry number 0 is the default if the command is not used.
default 1

## timeout sec
# Set a timeout, in SEC seconds, before automatically booting the default entry
# (normally the first entry defined).
timeout 5

# Pretty colours
color cyan/blue white/blue
gfxmenu (hd0,4)/boot/message

title Debian GNU/Linux, kernel
root (hd0,4)
kernel /boot/vmlinuz root=/dev/hda5 ro
initrd /boot/initrd.img
boot

title kboot system loader  environment
root (hd0,4)
kernel /boot/vmlinuz ro quiet root=/dev/ram0 kboot=block://(/dev/hda5,ext3)/boot/boot_menu.config
initrd /boot/kboot-root-glibc.cpio.gz
boot
```

A.2 Simple `kboot.conf` for i386

The following example shows a simple system loader configuration for the i386 platform. Two userinterface instances are started to listen on two different devices simultaneously.

```
#
# This is a sample config file for kilo
#

default linux1
timeout 0
password secret

userinterface linemode /dev/tty1
userinterface linemode /dev/tty

boot_entry {
        title Debian GNU/Linux, latest kernel
        label linux1

        root block://(/dev/hda5,ext3)/boot/
        kernel vmlinuz
        initrd initrd.img
        cmdline root=/dev/hda5 ro ramdisk_size=100000 lang=de apm=power-off nomce lapic vga=normal
}

boot_entry {
        title Debian GNU/Linux, rescue kernel
        label linux2

        root block://(/dev/hda5,ext3)/boot/
        kernel vmlinuz-2.6.11-kanotix-7
        initrd initrd.img-2.6.11-kanotix-7
        cmdline root=/dev/hda5 ro ramdisk_size=100000 lang=de apm=power-off nomce vga=0x317
```

```
}

boot_entry {
        title Debian GNU/Linux, rescue kernel (locked)
        label linux3

        lock
        root block://(/dev/hda5,ext3)/boot/
        kernel vmlinuz-2.6.11-kanotix-7
        initrd initrd.img-2.6.11-kanotix-7
        cmdline root=/dev/hda5 ro ramdisk_size=100000 lang=de apm=power-off nomce vga=0x317
}

boot_entry {
        title kboot environment
        label kboot

        root block://(/dev/hda5,ext3)/boot/
        kernel vmlinuz
        initrd kboot-root-glibc.cpio.gz
        cmdline root=/dev/hda5 ro ramdisk_size=100000 lang=de apm=power-off nomce lapic vga=normal

}

boot_entry {
        title System Reboot
        label reboot

        reboot
}

boot_entry {
        title System Halt

        halt
}
```

A.3 An Example Using Advanced Configuration Features

The following example shows how the same system loader configuration files can be used on two different systems. The platform for the examples is s390 which gets the boot configuration of the first stage boot loader in the file zipl.conf. Both systems share the same configuration files.

A.3.1 zipl.conf on System 1

System 1 is a virtual machine called `linux41`. The configuration for the first stage bootloader `zipl` references a file `parmfile_linux41.kboot` which contains the kernel commandline that is used to boot the initial system loader linux environment.

```
# This is an expample zipl.conf file
[defaultboot]
default = kboot
[kboot]
target     = "/home/kboot/boot"
image      = "/home/kboot/boot/s390/vmlinuz"
ramdisk    = "/home/kboot/boot/s390/initrd_sl2"
parmfile   = "/home/kboot/boot/s390/parmfile_linux41.kboot"
```

The kernel command line in `parmfile_linux41.kboot` contains the parameter `quiet` to suppress unnecessary messages while the system loader is booted. The second parameter references the system loader configuration file `kboot.conf` (shown in A.3.3 on the following page) on a disk of the virtual machine. Note that on the i386 platform usually the `block` URI scheme would be used instead of the `dasd` URI scheme.

```
quiet kboot=dasd://(0.0.5c60,1)/home/kboot/boot/s390/kboot.conf
```

A.3.2 zipl.conf on System 2

System 2 is another virtual machine called `linux40`. This machine has a kernel and the initial ramdisk of the system loader installed locally but gets the boot configuration via the network from system 1. The initial setup of the network interface is done via the `kset=` parameter on the kernel commandline. The `kboot=` parameter references the boot configuration via ftp.

```
# This is an expample zipl.conf file
[defaultboot]
default = kboot
[kboot]
target     = "/boot"
image      = "/boot/vmlinuz-2.6.16.18"
ramdisk    = "/boot/initrd_sl2"
parameters = "quiet
  kset=mod(qeth),qeth(0.0.f5db,0.0.f5dc,0.0.f5dd),static(eth0,9.152.26.120,255.255.252.0,9.152.24.1,9.152.120.241)
  kboot=ftp://kboot:kboot@53v15g41.boeblingen.de.ibm.com/home/kboot/boot/s390/kboot.conf"
```

A.3.3 System Loader Configuration Used by Both Systems

The following system loader configuration is used by both systems and shows various examples for system dependent definitions, device setup and included configuration parts and how these capabilities can be combined.

```
#
# My first system loader config file
#

default linux1
timeout 600

#
# define userinterfaces to be started
# 1. start a linemode ui on /dev/console
# 2. start a second linemode interface for incoming ssh connections
#

userinterface linemode /dev/console
userinterface ssh linemode

#
# device setup
#

# enable two fcp devices on the vmguest named 'linux40'

system vmguest(linux40) {

  setup zfcp {
    busid 0.0.54ae
    wwpn 0x5005076300cb93cb
    lun 0x512e000000000000
  }

  setup zfcp {
    busid 0.0.54ae
    wwpn 0x5005076300cb93cb
    lun 0x512f000000000000
  }
}

# load the qeth kernel module and enable a qeth device
# on the vmguest 'linux41' on lpar 'g53lp15'

system vmguest(linux41,g53lp15) {

  setup module {
    name qeth
  }

  setup qeth {
    busid 0.0.f5de
```

```
    busid 0.0.f5df
    busid 0.0.f5e0
  }
}

#
# network setup 'linux41' only
# setup for other systems is done via the kernel command line
#

system vmguest(linux41,g53lp15) {

  setup network {
    mode       static
    address    9.152.26.121
    mask       255.255.252.0
    gateway    9.152.24.1
    nameserver 9.152.120.241
    interface  eth0
  }
}

#
# here we have the entries for the boot menu
#

boot_entry {
  title latest kernel from DASD
  label linux1

  system vmguest(linux41) {
    root dasd://(0.0.5c60,1)/home/kboot/boot/s390/
  }

  system not(vmguest(linux41)) {
    root ftp://kboot:kboot@53v15g41.boeblingen.de.ibm.com/home/kboot/boot/s390/
  }

  kernel vmlinuz

  system vmguest(linux40) {
    cmdline dasd=5c5e-5c5f root=/dev/dasda1 ro noinitrd selinux=0
  }

  system vmguest(linux41) {
    cmdline dasd=5c60-5c61 root=/dev/dasda1 ro noinitrd selinux=0
  }
}

system vmguest(linux40) {

  boot_entry {
    title latest kernel from EVMS
    label evms

    root block://(/dev/evms/evmsvol)/boot/
    kernel vmlinuz
    cmdline dasd=5c5e-5c5f root=/dev/dasda1 ro noinitrd selinux=0
  }
}

boot_entry {
  title rescue kernel from DASD
  label rescue

  system vmguest(linux41) {
    root dasd://(0.0.5c60,1)/home/kboot/boot/s390/
  }

  system not(vmguest(linux41)) {
    root ftp://kboot:kboot@53v15g41.boeblingen.de.ibm.com/home/kboot/boot/s390/
  }

  kernel vmlinuz.rescue

  system vmguest(linux40) {
    cmdline dasd=5c5e-5c5f root=/dev/dasda1 ro noinitrd selinux=0
  }

  system vmguest(linux41) {
```

```
    cmdline dasd=5c60-5c61 root=/dev/dasda1 ro noinitrd selinux=0
  }
}

system vmguest(linux41) {
  include dasd://(0.0.5c60,1)/home/kboot/boot/s390/standard_entries.conf
}

system not(vmguest(linux41)) {
  include ftp://kboot:kboot@53v15g41.boeblingen.de.ibm.com/home/kboot/boot/s390/standard_entries.conf
}
```

To show the include mechanism the definition of the generic boot entries `reboot`, `halt` and `shell` is done via the following include file:

```
boot_entry {
  title System Reboot
  label reboot

  reboot
}

boot_entry {
  title System Halt

  halt
}

boot_entry {
  title start a shell!

  shell
}
```