AutoYaST

Automatic Linux Installation and Configuration with YaST2

SUSE Linux Products GmbH

Anas Nashif,
Uwe Gansert, *SuSE* **Linux Products GmbH**

AutoYaST: Automatic Linux Installation and Configuration with YaST2

SUSE Linux Products GmbH
by Anas Nashif and Uwe Gansert

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Introduction

AutoYaST is a system for installing one or more SuSE Linux systems automatically and without user intervention. AutoYaST installations are performed using a control file with installation and configuration data. The control file can be created using the configuration insterface of AutoYaST and can be provided to YaST2 during installation in different ways.

1.1. Availability

AutoYaST is available with recent *SuSE* products starting from *SuSE Linux 8.0* and business products starting from *SLES 8*.

Products prior to SuSE Linux 8.0 and business products based on *SLES 7* have an auto-installation system based on *YaST1*. A configuration management system is provided by *ALICE* for these products.

Updated documentation

Updated documentation can always be found at the following URL: http://yast2.suse.com/autoinstall/

1.2. Motivation

The Linux Journal [http://www.linuxjournal.com/], in an article in issue 78 [http://www.linuxjournal.com/categories.php?op=newindex&catid=178] writes:

" A standard Linux installation asks many questions about what to install, what hardware to configure, how to configure the network interface, etc. Answering these questions once is informative and maybe even fun. But imagine a system engineer who has to set up a new Linux network with a large number of machines. Now, the same issues need to be addressed and the same questions answered repeatedly. This makes the task very inefficient, not to mention a source of irritation and boredom. Hence, a need arises to automate this parameter and option selection."

"The thought of simply copying the hard disks naturally crosses one's mind. This can be done quickly, and all the necessary functions and software will be copied without option selection. However, the fact is that simple copying of hard disks causes the individual computers to become too similar. This, in turn, creates an altogether new mission of having to reconfigure the individual settings on each PC. For example, IP addresses for each machine will have to be reset. If this is not done properly, strange and inexplicable behavior results."

Regular installation of SuSE Linux is semi-automated by default. The user is requested to select the necessary information at the beginning of the installation (In most cases language only), YaST2 then generates a proposal for the underlying system depending on different factors and system paramters. In most cases, and especially for new systems, such a proposal can be used to install the system and provides a usable installation.

The steps following the proposal are fully automated and the user is only prompted at the end of the installation to configure hardware and network services.

AutoYaST can be used where no user intervention is required or where customization is required. Using a control file, YaST2 prepares the system for a custom installation and avoids any interaction with the user, unless specified in the file controling the installation.

AutoYaST is not an automated GUI system. This means that in most cases many screen will be skipped, i.e. you will never see the language selection interface. AutoYaST will simply pass the language parameter to the sub-system without displaying any language related interface.

1.3. Overview and Concept

Using AutoYaST, multiple systems sharing the same environment and similar but not necesserily identical hardware performing similar tasks can easily be installed in parallel and in a short time. A configuration file (referred to as "control file") is created using existing configuration resources. The control file can be easily tailored for any specific environment.

Unlike autoinstallation systems available with older *SuSE* releases, AutoYaST is fully integrated and provides various options for installing and configuring a system. The main advantage over older systems and other auto-installation systems is the possibility to configure a computer by using existing modules and avoiding using custom scripts which are normally executed at the end of the installation.

This document will guide you through the three steps of auto-installation:

• Preparation: All relevant information about the target system are collected and turned into the appropriate directives of the control file. The control file is transferred onto the target system where its directives will be parsed and transformed to YaST2 conforming data.

• Installation: follows the instructions given in the control file and installs the base system.

• Configuration: YaST2 in addition to user-defined post-install scripts complete the system configuration

The complete and detailed process is illustrated in the following figure:

Figure 1.1. Auto-installation process

Chapter 2. The Control File

2.1. Introduction

The control file is in most cases a configuration description for a single system. It consists of sets of resources with properties including support for complex structures representations such as lists, records, trees and large embedded or referenced objects.

2.2. Format

The XML configuration format provides a consistent file structure, which is easier to learn and remember when attempting to configure a new system.

Using XML, you can eliminate (nearly) all of the configuration file parsing and error handling - an external XML parser can do that instead - (especially if it is a validating parser). To make sure the control file is well-formatted and that the syntax is valid, you can run the control file through a validating parser before it is actually used for automatic installation. This is especially required if you prefer to edit the profile manually.

The following example shows a control file in XML format:

Example 2.1. XML Control File (Profile)

```
<?xml version="1.0"?>
<!DOCTYPE profile SYSTEM
 "/usr/share/autoinstall/dtd/profile.dtd">
 <profile
 xmlns="http://www.suse.com/1.0/yast2ns"
 xmlns:config="http://www.suse.com/1.0/configns">
 <install> <!-- install is deprecated since SUSE Linux 10.0 -->
   <partitioning   config:type="list">
       <drive>
           <device>/dev/hda</device>
           <partitions config:type="list">
             <partition>
                <filesystem config:type="symbol">ext2</filesystem>
                <size>520Mb</size>
                <mount>/</mount>
             </partition>
             <partition>
                <filesystem config:type="symbol">reiser</filesystem>
                <size>1200Mb</size>
                <mount>/data</mount>
             </partition>
           </partitions>
       </drive>
   </partitioning>
 </install> <!-- install is deprecated since SUSE Linux 10.0 -->
 <configure> <!-- configure is deprecated since SUSE Linux 10.0 -->
   <scripts>
    <pre-scripts>
     <script>
        <interpreter>shell</interpreter>
        <filename>start.sh</filename>
         <source>

            <![CDATA[
#!/bin/sh
echo "Starting installation"
exit 0

]]>

         </source>
      </script>
     </pre-scripts>
   </scripts>
 </configure> <!-- configure is deprecated since SUSE Linux 10.0 -->
</profile>
```

2.3. Structure

Below is an example of a basic control file container, the actual content of which is explained later on in this chapter.

Example 2.2. Control file container

```
<?xml version="1.0"?>
<!DOCTYPE profile SYSTEM
 "/usr/share/autoinstall/dtd/profile.dtd">
 <profile
 xmlns="http://www.suse.com/1.0/yast2ns"
 xmlns:config="http://www.suse.com/1.0/confign s">

<!-- RESOURCES -->

</profile>
```

The profile element (root node) contains one or more distinct resource elements. The permissible resource elements are specified in the DTD.

The root element in the control file can for example contain the following sub-keywords:

- installation (Tag *<install>* this tag is deprecated since SUSE Linux 10.0)

  - Bootloader configuration: bootloader device, bootloader location (Tag *<bootloader>*)

  - Partitioning: drives and partition plans (Tag *<partitioning>*)

  - General: Installation instructions, including all variables related to the client i.e. display, languages, keyboard etc. (Tag *<general>*)

  - Software: Software and Package selections (Tag *<software>*)

- configuration (Tag *<configure>* this tag is deprecated since SUSE Linux 10.0)

  - Network: network configuration for the client and servers providing services to the target client (Tag *<networking>*)

  - Users: user administration, including first user and root. (Tag *<users>*)

  - User scripts: (Tag *<scripts>*)

2.3.1. Resources and Properties

A resource element either contains multiple and distinct property and resource elements or contains multiple instances of the same resource element or is empty. The permissible content of a resource element is specified in the DTD.

A property element is either empty or contains a literal value. The permissible property elements and values in each resource element are specified in the DTD.

An element can be either a container of other elements (a resource) or have a literal value (a property), it can never be both. This restriction is specified in the DTD. A configuration component with more than one value must either be represented as some kind of embedded list in a property value or as a nested resource.

2.3.2. Nested Resources

Nested resource elements allow a tree like structure of configuration components to be built to any level.

Example 2.3. Nested Resources

```
...
<drive>
  <device>/dev/hda</device>
  <partitions config:type="list">
    <partition>
        <size>1000mb</size>
        <mount>/</mount>
    </partition>
    <partition>
        <size>250mb</size>
        <mount>/tmp</mount>
    </partition>
  </partitions>
</drive>
....
```

In the example above the disk resource consists of a device property and a partitions resource. The partitions resource contains multiple instances of the partition resource. Each partition resource contains a size and mount property.

Although it is specified in the DTD that the partitions resource contains multiple instances, it is still required to specify this to avoid wrong data typing in YaST2. Using the example above, imagine having a drive with only one partition. This will result in interpreting the partition resource as a property. To avoid this the following syntax must be used when defining multiple instances. For more information about type attributes, see next section.

Example 2.4. Nested Resources with type attributes

```
...
<drive>
  <device>/dev/hda</device>
  <partitions config:type="list">
    <partition>
        <size>1000</size>
        <mount>/</mount>
    </partition>
    <partition>
        <size>250</size>
        <mount>/tmp</mount>
    </partition>
  </partitions>
</drive>
....
```

2.3.3. Attributes

Global profile attributes are used to define meta-data on resources and properties. Attributes are used to define context switching. They are also used for naming and typing properties as shown in earlier sectionons [1].

Profile attributes are defined in the configuration namespace and must always be prefixed with *config:* . All profile attributes are optional. Most can be used with both resource and property elements but some can only be used with one type of element which is specified in the DTD.

---

[1] Profile attributes are in a separate namespace so they don't have to be treated as reserved words in the default namespace. New ones can then be added without having to potentially alter existing profiles.

The type of an element is defined using the *config:type* attribute. The type of a resource element is always RESOURCE , although this can also be made explicit with this attribute (to ensure correct identification of an empty element for example when there is no DTD to refer to). A resource element cannot be any other type and this restriction is specified in the DTD. The type of a property element determines the interpretation of its literal value. The type of a property element defaults to *STRING* , as specified in the DTD. The full set of permissible types is specified in the DTD.

2.4. The XML Document Type Definition (DTD)

2.4.1. Introduction

The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in the XML document, or as an external reference.

XML provides an application independent way of sharing data. With a DTD, the application can use a standard DTD to verify that data that the user supplies is valid. A "Valid" XML document is a "Well Formed" XML document which conforms to the rules of a Document Type Definition (DTD).

In AutoYaST, a DTD should is available to allow users to validate the control files before the installation process is initiated. The DTD can be also used with XML editors while editing the control file to avoid later errors.

2.4.2. Example DTD

- A *drive* resource containing a *device* property and a *partitions* property represented as a nested resource.

- A *partitions* resource containing multiple instances of the *partition* property represented as a nested resource.

- A *partition* resource containing a *size* property and a *mount* property.

Below is the XML for an example node view profile for the above tree which includes a DTD which validates it.

Example 2.5. Example DTD

```
<?xml version="1.0"?>
<!DOCTYPE profile [
<!ELEMENT profile (install)>
<!ELEMENT install (partitioning)>
<!ELEMENT partitioning (drive+)>
<!ELEMENT drive (device,partitions)>
<!ELEMENT device (#PCDATA)>
<!ELEMENT partitions (partition*)>
<!ELEMENT partition (size,mount)>
<!ELEMENT size (#PCDATA)>
<!ELEMENT mount (#PCDATA)>
]>
<profile>
.....
  <install>
    <partitioning config:type="list">
      <drive>
        <device>
           /dev/hda
        </device>
        <partitions>
          <partition>
            <size>1000mb</size>
            <mount>/</mount>
          </partition>
          <partition>
            <size>250mb</size>
            <mount>/tmp</mount>
          </partition>
```

```
            </partitions>
         </drive>
      </partitioning>
   </install>
.....
</profile>
```

Chapter 3. Creating A Control File

3.1. Collect information

In order to create the control file, first you need to collect information about the systems your are going to install. This includes among other things hardware data and network information. Make sure you know the following about the machines you want to install:

- Hard disk types and sizes

- Graphic interface and attached monitor if any

- Network interface and MAC address if known (i.e. when using DHCP)

With these parameters you are ready to go and create a profile of your systems to control the auto-installation process.

3.2. Using the Configuration Management System

In order to create the control file for one or more computers, a configuration interface based on YaST2 is provided. This system depends on existing modules which are usually used to configure a computer in regular operation mode, i.e. after *SuSE* Linux is installed.

The configuration management system lets you create control files easily and additionally it lets you manage a repository of configurations for use in a networked environment and with multiple clients.

Figure 3.1. Configuration System

3.2.1. Creating a new Profile

With some exceptions, almost all resources of the control file can be configured using the configuration management system. The system offers flexibility and configuration of some resources is identical to this available in the YaST2 Control Center. In addition to the existing and familiar modules new interfaces were created for special and complex configurations, for example for partitioning, general options and software.

Furthermore, using the CMS guarantees that the resulting control file is valid and insures that it can be used directly to start automated installation.

Make sure the configuration system is installed (package *autoyast2*) and call it using the *YaST2 Control Center* or call it directly as root with the following command (make sure the *DISPLAY* variable is set correctly to start the graphical user interface instead of the text based one):

```
/sbin/yast2 autoyast
```

3.2.2. Import of Legacy and Foreign Configuration Files

AutoYaST offers the option to import ALICE configuration files from previous SuSE releases and foreign auto-installation systems (Kickstart). Consult the chapter dealing with these issues in this manual.

3.3. Creating/Editing a Control File Manually

If you edit the control file manually, make sure it has a valid syntax. To check the syntax, use some tools already available on the distribution. For example to verify that the file is well formed, use the utility **xmllint** available with the *libxml2* package:

```
xmllint <control file>
```

If the control file is not well formed, i.e. if a tag is not closed, **xmllint** will report about the errors.

Before going on with the auto-installation, please fix any errors resulting from such checks. The auto-installation process can't be started with an invalid and non-well formed control file.

You can use any XML editor available on your system or use your favorite text editor with XML support (i.e. Emacs, Vim). However, it is not quite optimal to create the control file manually for large number of machines and it should only be seen as an interface between the auto-installation engine and the Configuration Management System (CMS).

Figure 3.2. Editing the control file with kxmledit

3.4. Creating a Profile (control file) via Script with XSLT

For the case you have a template and just want to change a few things via script or command line, you can use a XSLT processor like *sablot* for this. Lets say you have an autoyast profile and you want to fillout the hostname via script for any reason (maybe because you have to do it so often, you want to script it)

First you have to create an XSL file

Example 3.1. Example file for replacing hostname/domain by script

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:y2="http://www.suse.com/1.0/yast2ns"
  xmlns:config="http://www.suse.com/1.0/configns"
  xmlns="http://www.suse.com/1.0/yast2ns"
  version="1.0">

  <xsl:output method="xml" encoding="UTF-8" indent="yes" omit-xml-declaration="no" cdata-section-elements="source"/>

  <!-- the parameter names -->
  <xsl:param name="hostnam"/>
  <xsl:param name="domain"/>

  <xsl:template match="/">
    <xsl:apply-templates select="@*|node()"/>
  </xsl:template>

  <xsl:template match="y2:dns">
    <xsl:copy>
      <!-- where to copy the parameters -->
      <domain><xsl:value-of select="string($domain)"/></domain>
      <hostname><xsl:value-of select="string($hostname)"/></hostname>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>


  <xsl:template match="@*|node()" >
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

As you can see, this file expects the "hostname" and the "domain" as parameters from the user.

```
<xsl:param name="hostname"/>
<xsl:param name="domain"/>
```

There will be a "copy" of those parameters in the "dns" section of the control file. That means, if there already is a domain element in the dns section, you'll get a second one (no good).

If you want to create a new autoyast profile now from the template plus the XSL file, run the following command:

```
sabcmd add_hostname.xsl \$hostname=myHost \$domain=my.domain  template.xml
```

You'll get a filled out autoyast profile then on STDOUT.

If you have multiple XSL files you want to apply to a template, do it like this

```
        sabcmd add_hd_vg.xsl \$device=/dev/sda \$partition=p2 \$vg=system \
        sabcmd add_harddisk.xsl \$device=/dev/system \$lvm=true \
        sabcmd ....
        sabcmd add_hostname.xsl \$hostname=myHost \$domain=my.domain
```

So you just pipe the output of each sabcmd to the next sabcmd.

For more information aout XSLT, go to the official webpage www.w3.org/TR/xslt [http://www.w3.org/TR/xslt]

Chapter 4. Configuration and Installation Options

This chapter introduces important parts of a control file for standard purposes. To have an idea about the other options available, use the configuration management system.

Note that for some of the configuration options to work, additional packages have to be installed, depending on the software selection you have configured. If you choose to install *Minimal* then some packages might be missing and those have to be added to the individual package selection.

YaST will install packages required by YaST modules in the second phase of the installation and before the post-installation phase of AutoYaST has started, however if the YaST modules are not available in the system, this will not happen. For example, no security settings will be configured if *yast2-security* is not installed.

4.1. General Options

General options include all the settings related to the installation process and the environment of the installed system. Among others it includes the following 4 properties which are required: *language*, *keyboard*, *clock* and *mouse* almost for any installation. If left out, default values will be used, which might not be in one hand with what you want.

Example 4.1. General Options

```
<general>
  <signature-handling>
    <accept_unsigned_file        config:type="boolean">true</accept_unsigned_file>
    <accept_file_without_checksum config:type="boolean">true</accept_file_without_checksum>
    <accept_verification_failed  config:type="boolean">true</accept_verification_failed>
    <accept_unknown_gpg_key      config:type="boolean">true</accept_unknown_gpg_key>
    <accept_non_trusted_gpg_key  config:type="boolean">true</accept_non_trusted_gpg_key>
  </signature-handling>
  <mode>
  <halt config:type="boolean">false</halt>
    <confirm config:type="boolean">true</confirm>
    <second_stage config:type="boolean">true<second_stage>
  </mode>
</general>
```

By default, the auto-installation process has to be confirmed by the user. The confirmation should be disabled if a fully unattended installation is desired. This option is used to view and change the settings on a target system before anything is committed and can be used for debugging. It is set to *true* by default to avoid recursive installs when the system schedules a reboot after initial system setup.

With *halt* you make autoyast to turn off the machine after all packages have been installed. So instead of the reboot into stage two, the machine is turned off. The bootloader is alreay installed and all your chroot scripts have run.

Change starting from SUSE Linux 9.1/SLES 9

The *reboot* property in the *mode* resource was used to force a reboot after initial system setup and before the system is booted for the first time. Currently after initial installation all systems must boot, which makes this option obsolete.

SLES9 Only Options

In SLES 9, it is possible to halt the system after the initial installation phase using the boolean property *halt*

Change starting from SUSE Linux 10.1/SLES10

The *language*, *keyboard* and *clock* properties in the *general* resource were moved to the root (*profile*) element of the autoyast profile. So don't use them in the general section anymore.

Since now you can use the *second_stage* property, which can turn off autoyast after the first reboot. So the complete second stage is a manual installation (default is true, which means that autoyast is doing a complete installation).

For the signature-handling, please read the *Software* chapter of this documentation.

4.2. Reporting

The *report* resource manages 3 types of pop-ups that may appear during installation.

- Messages Popups (Usually non-critical, informative messages)

- Warning Popups (If something might go wrong)

- Error Popups (In the case of an error)

Example 4.2. Reporting Behavior

```
<report>
  <messages>
    <show config:type="boolean">true</show>
    <timeout config:type="integer">10</timeout>
    <log config:type="boolean">true</log>
  </messages>
  <errors>
    <show config:type="boolean">true</show>
    <timeout config:type="integer">10</timeout>
    <log config:type="boolean">true</log>
  </errors>
  <warnings>
    <show config:type="boolean">true</show>
    <timeout config:type="integer">10</timeout>
    <log config:type="boolean">true</log>
  </warnings>
</report>
```

Depending on your experience, you can skip, log and show (with timeout) those messages. It is recommended to show all *messages* with timeout. Warnings can be skipped in some places but should not be ignored.

By default, the settings in auto-installation mode is to show all messages without logging and with a timeout of 10 seconds.

Critical system messages

Note that *not* all messages during installation are controlled by the *report* resource. Some critical messages concerning package installation and partitioning will still show up ignoring your settings in the *report* section. Mostly those messages will have to be

answered with *Yes* or *No*.

4.3. The Boot loader

It is now possible to specify which bootloader needs to be installed and it is possible to specify sections and more bootloader options.

If the bootloader resource is not configured at all, the system will propose a configuration based on the partitioning scheme. If no global variables are configured, the system will propose a configuration for the global variables based on the partitioning scheme. Similarly, if no sections are configured, the system will propose a section configuration. No changes or additions will be made to existing global variables or existing sections. Only for the grub bootloader, missing standard values in the global variables will be filled with default values.

In some cases you must decide where to install the bootloader (the Master Boot Record or the first sector of the /boot partition) and must specify additional options for the bootloader to be installed correctly.

Boot Partition on IA64 Systems

The boot partition on *IA64* is */boot/efi* and is set to this value if only */boot* is configured in the control file.

Additionally, you can configure special kernel and boot parameters, Kernel parameters can be configured using the *kernel_parameters* property.

Example 4.3. Bootloader configuration

```
 <?xml version="1.0"?>
<!DOCTYPE profile SYSTEM "/usr/share/autoinstall/dtd/profile.dtd">
<profile xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
    <bootloader>
      <global config:type="list">
        <global_entry>
          <key>prompt</key>
          <value config:type="boolean">false</value>
        </global_entry>
      </global>
      <loader_device></loader_device>
      <loader_type>lilo</loader_type>
      <location>mbr</location>
      <sections config:type="list"/>
    </bootloader>
</profile>
```

The *global* resource is used to configure additional boot loader options. Note that bootloaders can have different configurable options. Consult the documentation for the specific bootloader you want to use before adding any options.

For example, to configure the serial console with *grub*, you can add the following options:

Example 4.4. Serial console configuration with GRUB

```
    <bootloader>
      <global config:type="list">
        <global_entry>
          <key>serial</key>
          <value>--unit=1 --speed=115200</value>
        </global_entry>
        <global_entry>
          <key>terminal</key>
```

```
        <value>serial console</value>
      </global_entry>
    </global>
    <loader_type>grub</loader_type>
    <location></location>
  </bootloader>
```

Tip

The bootloader configuration interface offers more configuration options than one might need for a regular setup. The options described in the example above should be sufficient for most cases.

In the current version, bootloader configuration also supports different architectures and boards. The configuration of the bootloader can be done offline using AutoYaST. For example, the bootloader configuration for PPC has the following additional options:

Example 4.5. Bootloader configuration for PPC

```
  <?xml version="1.0"?>
<!DOCTYPE profile SYSTEM "/usr/share/autoinstall/dtd/profile.dtd">
<profile xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
    <bootloader>
      <board_type>iseries</board_type>
      <global config:type="list"/>
      <iseries_streamfile>/boot/suse_linux_bootfile</iseries_streamfile>
      <iseries_write_prepboot config:type="boolean">true</iseries_write_prepboot>
      <iseries_write_slot_a config:type="boolean">true</iseries_write_slot_a>
      <iseries_write_slot_b config:type="boolean">true</iseries_write_slot_b>
      <iseries_write_streamfile config:type="boolean">true</iseries_write_streamfile>
      <loader_device>B</loader_device>
      <loader_type>ppc</loader_type>
      <location></location>
      <prep_boot_partition>/dev/hda1</prep_boot_partition>
    </bootloader>
</profile>
```

Currently the following bootloaders can be configured offline using the AutoYaST interface:

• LILO

• GRUB

• ELILO

• zipl

• yaboot

4.4. Partitioning

4.4.1. drive configuration

The following elements must be between the <partitioning config:type="list"><drive> ... </drive></partitioning> tags in the <profile> section.

Table 4.1.

Attribute	Values	Description
device	the device you want to config-ure in this section. Since SUSE Linux 10.1 and SLES10, you can use persistent device names via id, like /dev/disk/by-id/edd-int13_dev80. With SLES10 SP1 and SUSE Linux 10.2, *by-path* is possible too like /dev/disk/by-path/pci-0001:00:03.0-scsi-0:0:0:0. `<device>/dev/hda</device>`	optional. If left out, autoyast tries to guess the device. A RAID must always have "/dev/md" as device
initialize	if set to true, the partition table gets wiped out before autoyast starts the partition calculation `<initialize config:type="boolean">true</initialize>`	optional. The default is false.
is_lvm_vg	This tells autoyast that this device is not a physical device but a LVM volume group (see LVM configuration below) `<is_lvm_vg config:type="boolean">true</is_lvm_vg>`	DEPRECATED since SLES10SP1 and SL10.2 - use *type* instead. Must be true if this device is a LVM volume group. The default is false.
is_evms_vg	this tells autoyast that this device is not a physical device but an EVMS volume group (see EVMS configuration be-low) `<is_evms_vg config:type="boolean">true</is_evms_vg>`	DEPRECATED since SLES10SP1 and SL10.2 - use *type* instead. Must be true if this device is an EVMS volume group. The default is false.
partitions	this is a list of <partition> entries (see table below) `<partitions config:type="list"><partition>...</partition>...</partitions>`	optional. If no partition is spe-cified, autoyast will create it's own idea of a nice partitioning (see Automated Partitioning be-low).
pesize	this value makes only sense with LVM/EVMS. `<pesize>8M</pesize>`	optional. Default is 4M for EVMS/LVM volume groups.
use	this parameter tells autoyast which strategy it shall use to partition the harddisc. You can choose between: • all (uses the whole device while calculating the new partitioning) • linux (only existing linux	this parameter should be provided

Attribute	Values	Description
	partitions are used) • free (only unused space on the device gets used. No other partitions gets touched) • 1,2,3 (a list of comma seperated numbers that indicates the partition numbers to use)	
type	this value describes the type of the *drive* and is a replacement for *is_lvm_vg* and *is_evms_vg* used in SLES10 and SL10.1 You can choose between: • CT_DISK for physical harddisks (default) • CT_LVM for LVM volume groups • CT_EVMS for EVMS volume groups `<type config:type="symbol">CT_LVM</type>`	optional. Default is CT_DISK for a normal physical harddisk.

4.4.2. partition configuration

The following elements must be between the <partitions config:type="list"><partition> ... </partition></partitions> tags in the <drive> section.

Table 4.2.

Attribute	Values	Description
create	the "create" tells autoyast if this partition must be created or if it's already existing `<create config:type="boolean">false</create>`	if set to false, there must be some information for autoyast which partition this is (like with partition_nr)

mount	the mountpoint of this partition.
	
	`<mount>/</mount>`
	
	`<mount>swap</mount>`
		you should have at least a root partition (/) and a swap partition

fstopt	mount options for this partition	see "man mount" for the mounttoptions you can use

Attribute	Values	Description
	`<fstopt>ro,noatime,user,data=ordered,acl,user_xattr</fstopt>`	
label	the label the partition has (useful for the "mountby" parameter - see below). `<label>mydata</label>`	see "man e2label" for example.
uuid	the uuid the partition has (only useful for the "mountby" parameter - see below). `<uuid>1b4e28ba-2fa1-11d2-883f-b9a761bde3fb</uuid>`	see "man uuidgen"
size	the size for the partition like 4G, 4500M, ... The /boot partition and the swap partition can have "auto" as size too, to let autoyast calculate a reasonable size for them. On partition can have the value "max" to fillup all available space. with SUSE Linux 10.2 and SLES10 SP1, you can specify the the size in percentage. So 10% will use 10% of the size of the harddisk/VG. You can mix auto,max,sizes and percentage like you want. `<size>10G</size>`	
format	shall autoyast format the partition? `<format config:type="boolean">false</format>`	if "create" is true, then it's very likely that this is true too
filesystem	what filesystem is used on this partition? • reiser (the default) • ext2 • ext3 • xfs • jfs `<filesystem config:type="symbol">reiser</filesystem>`	optional. The default is reiser

Attribute	Values	Description
partition_nr	the partition_nr this partition has/will have. If you have set create=false, then you can tell autoyast which partition you mean by the partition_nr. You can force autoyast to create only primary partitions by configuring only partition numbers below 5. `<partition_nr config:type="integer">2</partition_nr>`	in most cases nr. 1-4 are primary partitions and 5-... are logical partitions
partition_id	the partition_id configures the id of the partition. If you want something else than 131 for linux partition or 130 for swap, you must configure that with partition_id. `<partition_id config:type="integer">131</partition_id>`	the default is 131 for linux partition. 130 for swap is set by autoyast itself too.
filesystem_id	look at partition_id above. For historical reasons they represent the same. `<filesystem_id config:type="integer">131</filesystem_id>`	since 10.1 and SLES10 it's recommended to use partition_id instead.
mountby	instead of a partition number, you can tell autoyast to mount a partition by label or by uuid `<mountby config:type="symbol">label</mountby>`	see "label" and "uuid" documentation above
lv_name	if this partition is in a logical volume in a volume group (LVM or EVMS) (see is_lvm_vg/is_evms_vg parameter in drive configuration) you must specifiy the logical volume name here. `<lv_name>opt_lv</lv_name>`	
lvm_group	if this is a physical partition that is used by (part of) a volume group (LVM), you have to specify the name of the volume group here. `<lvm_group>system</lvm_group>`	
evms_group	if this physical partition is used by a volume group (EVMS), you have to specify the name of	

Attribute	Values	Description
	the volume group here. `<evms_group>system</evms_group>`	
raid_name	this physical volume is part of a RAID and the name of the raid is specified here. `<raid_name>/dev/md0</raid_name>`	
raid_type	this physical volume is part of a RAID and the type of the raid is specified here.. `<raid_type>raid1</raid_type>`	
raid_options	special options for the raid are specified here. See below. `<raid_options>...</raid_options>`	
resize	This parameter is available since SLES10 SP1 and OpenSUSE 10.2. This boolean must be true if an existing partition should be resized. In this case, you want to set *create* to *false* too and in most cases you don't want to *format* the partition. You need to tell autoyast the *partition_nr* and the *size*. The size can be in percentage of the original size or as a number of the new size, like *800M*. *max* and *auto* don't work as size here. `<resize config:type="boolean">false</resize>`	The resize only works with physical disks. Not with LVM/ EVMS volumes.

4.4.3. raid options

The following elements must be between the `<partition><raid_options>` ... `</raid_options></partition>` tags.

Table 4.3.

Attribute	Values	Description
chunk_size	`<chunk_size>4</chunk_size>`	
parity_algorithm		

Attribute	Values	Description
	possible values are: left_asymmetric, left_symmetric, right_asymmetric, right_symmetric `<parity_algorithm>left_asymmetric</parity_algorithm>`	
raid_type	possible values are raid0,raid1 and raid5 `<raid_type>raid1</raid_type>`	the default is raid1

4.4.4. Automated Partitioning

For the automated partitioning to be completed, only the sizes and mount points of partitions can be provided. All other data needed for successful partitioning can be calculated during installation if they were not provided in the control file.

If no partitions are defined and the specified drive is also the drive where the root partition should be created, the following partitions are created automatically:

- */boot*

  Size of the */boot* is determined by the architecture of the target system.

- *swap*

  Size of the *swap* partitions is determined by the amount of memory available in the system.

- */* (root partition)

  Size of the */* (root partition) is the space left after creating *swap* and */boot*.

Depending on the initial status of the drive and how it was previously partitioned, it is possible to create the *default* partitioning in the following ways:

- *Use free space*

  If the drive is already partitioned, it is possible to create the new partitions using the available space on the hard drive. This requires the availability of enough space for all selected packages in addition to swap.

- *Reuse all available space*

  This option will lead to the deletion of all existing partitions (Linux and non-Linux partitions).

- *Reuse all available Linux partitions*

  This option will lead to the deletion of existing Linux partitions. All other partitions (i.e. Windows) will be kept. Note that this works only if the Linux partitions are at the end of the device.

- *Reuse only specified partitions*

  This option will lead to the deletion of the specified partitions. The selection of the partitions scheduled for deletion should be started from the last available partition.

Repartitioning using only some of the existing partitions can be accomplished only if the region selected to be partitioned exists at the end of the device and only with neighboring partitions. This means that you cannot repartition a region which contains a partition that should not be touched in the middle.

Important Notice

> The value provided in the *use* property determines how existing data and partitions are treated. The value *all* means that *ALL* data on the disk will be erased. Make backups and use the *confirm* property if you are going to keep some partitions with important data. This is automated installation and no pop-ups will notify you about partitions being deleted.

In case of the presence of multiple drives in the target system, all drives must be identified with their device names and how the partitioning should be performed.

Partition sizes can be given in Gigabytes, Megabytes or can be set to a flexible value using the keywords *auto* and *max*. *max* is used to fill a partition to the maximal available space on a drive (Which mean that the partition should be the last one on the drive). *auto* can be used to determine the size of a *swap* or *boot* partitions depending on the memory available and the type of the system.

A fixed size can be given as shown below:

1GB will create a partition with 1 GB size. *1500MB* will create a partition which is 1.5 GB big.

Example 4.6. Automated partitioning

The following is an example of a single drive system, which is not pre-partitioned and should be automatically partitioned according to the described pre-defined partition plan. If you leave the device out, an autodetection of the device will happen. So you don't have to do different profiles for /dev/sda or /dev/hda systems.

```
<partitioning  config:type="list">
  <drive>
    <device>/dev/hda</device>
    <use>all</use>
  </drive>
</partitioning>
```

A more detailed example shows how existing partitions and multiple drives are handled.

Example 4.7. Detailed automated partitioning

```
<partitioning  config:type="list">
<drive>
  <device>/dev/hda</device>
  <partitions config:type="list">
    <partition>
      <mount>/</mount>
      <size>5gb</size>
    </partition>
    <partition>
      <mount>swap</mount>
      <size>1gb</size>
    </partition>
  </partitions>
</drive>
<drive>
  <device>/dev/hdb</device>
  <use>all</use>
  <partitions config:type="list">
    <partition>
      <filesystem  config:type="symbol">reiser</filesystem>
```

```
            <mount>/data1</mount>
            <size>15gb</size>
          </partition>
          <partition>
            <filesystem  config:type="symbol">jfs</filesystem>
            <mount>/data2</mount>
            <size>auto</size>
          </partition>
        </partitions>
        <use>free</use>
      </drive>
</partitioning>
```

4.4.5. Advanced Partitioning features

4.4.5.1. Wipe out partition table

In the most cases this is not needed because autoyast can delete partitions one by one automatically but you have the option to let autoyast clear the partition table instead of deleting the partitions individually.

if you go into the "drive" section, you can add

```
<initialize config:type="boolean">true</initialize>
```

which tells Autoyast to delete the partition table before it starts to analyse the actual partitioning and calculates it's partition plan. Of course this means, that you can't keep any of your existing partitions.

4.4.5.2. Mount Options

By default a file system which is to be mounted is identified in /etc/fstab by the device name. This identification can be changed so the file system is found by searching for a UUID or a volume label. Note that not all file systems can be mounted by UUID or a volume label. To specify how a partition is to be mounted, use the *mountby* property which has the *symbol* type. Possible options are:

- device (default)

- label

- UUID

If you choose to mount the partition using a label, the name entered in the *label* property is used as the volume label.

Add any legal mount option allowed in the fourth field of /etc/fstab. Multiple options are separated by commas. Possible fstab options:

- *Mount Read-Only (ro):* No writable access to the file system is possible. Default is false.

- *No access time (noatime):* Access times are not updated when a file is read. Default is false.

- *Mountable by User (user):* The file system may be mounted by an ordinary user. Default is false.

- *Data Journaling Mode (ordered | journal | writeback) :* Specifies the journaling mode for file data. journal -- All data is committed into the journal prior to being written into the main file system. ordered -- All data is forced directly out to the main file system prior to its meta data being committed to the journal. writeback -- Data ordering is not preserved.

- *Access Control List (acl):* Enable access control lists on the file system.

- *Extended User Attributes (user_xattr):* Allow extended user attributes on the file system.

Example 4.8. Mount Options

```
<partitions config:type="list">
  <partition>
    <filesystem config:type="symbol">reiser</filesystem>
    <format config:type="boolean">true</format>
    <fstopt>ro,noatime,user,data=ordered,acl,user_xattr</fstopt>
    <mount>/local</mount>
    <mountby config:type="symbol">uuid</mountby>
    <partition_id config:type="integer">131</partition_id>
    <size>10gb</size>
  </partition>
</partitions>
```

4.4.5.3. Creating Primary and Extended Partitions

By default, AutoYaST will create an extended partition and will add all other new partitions as logical devices. It is possible however to instruct AutoYaST to create a certain partition as a primary or as extended partition. Additionally, it is possible to specify the size of a partition using sectors instead of the size in Mbytes.

The properties *partition_id* and *partition_type* control such behavior. To specify the size in sectors, the list resource *region* can be used.

Example 4.9. Advanced Automated partitioning

```
<partitioning config:type="list">
  <drive>
    <device>/dev/hdc</device>
    <partitions config:type="list">
      <partition>
        <partition_id config:type="integer">5</partition_id>
        <region config:type="list">
          <region_entry config:type="integer">0</region_entry>
          <region_entry config:type="integer">16858</region_entry>
        </region>
      </partition>
      <partition>
        <filesystem config:type="symbol">reiser</filesystem>
        <mount>/</mount>
        <partition_id config:type="integer">131</partition_id>
        <region config:type="list">
          <region_entry config:type="integer">0</region_entry>
          <region_entry config:type="integer">2081</region_entry>
        </region>
      </partition>
      <partition>
        <mount>swap</mount>
        <partition_id config:type="integer">130</partition_id>
        <region config:type="list">
          <region_entry config:type="integer">2081</region_entry>
          <region_entry config:type="integer">781</region_entry>
        </region>
      </partition>
    </partitions>
    <use>all</use>
  </drive>
</partitioning>
```

The last example only makes sense if you exactly know the boundaries of the partitions and it does only make sense if you are creating an exact copy of a system, for example when cloning. The first

region entry in the list is the beginning of the region, the second represents the length of the region.

The following example lets you create an extended partition with a custom size.

Example 4.10. Creating custom extended partitions

```
<partitioning config:type="list">
  <drive>
    <device>/dev/hdc</device>
    <partitions config:type="list">
      <partition>
        <filesystem config:type="symbol">ext2</filesystem>
        <mount>/boot</mount>
        <partition_id config:type="integer">131</partition_id>
        <partition_nr config:type="integer">2</partition_nr>
        <size>50mb</size>
      </partition>
      <partition>
        <mount>swap</mount>
        <partition_id config:type="integer">130</partition_id>
        <partition_nr config:type="integer">3</partition_nr>
        <size>100mb</size>
      </partition>
      <partition>
        <partition_id config:type="integer">15</partition_id>
        <partition_nr config:type="integer">4</partition_nr>
        <size>3000mb</size>
      </partition>
      <partition>
        <filesystem config:type="symbol">ext2</filesystem>
        <mount>/</mount>
        <partition_id config:type="integer">131</partition_id>
        <partition_nr config:type="integer">5</partition_nr>
        <size>1gb</size>
      </partition>
    </partitions>
    <use>free</use>
  </drive>
</partitioning>

</install>
```

4.4.5.4. Keeping Specific Partitions

In some cases you might choose to keep some partitions untouched and only format specific target partitions, rather than creating them from scratch. This might be the case of Linux installations have to co-exist with another operating system or if certain partitions contain data that you wish to keep untouched.

Such scenarios require certain knowledge about the target systems and hard drives. Depending on the scenario, you might need to know the exact partition table of the target hard drive with partition id's, sizes and numbers. With such data you can tell AutoYaST to keep certain partitions, format others and create new partitions if needed.

The following example will keep partitions 1, 2 and 5 and delete partition 6 to create two new partitions. All kept partitions will be only formatted.

Example 4.11. Keeping partitions

```
<partitioning config:type="list">
  <drive>
    <device>/dev/hdc</device>
    <partitions config:type="list">
      <partition>
        <create config:type="boolean">false</create>
        <format config:type="boolean">true</format>
        <mount>/</mount>
        <partition_nr config:type="integer">1</partition_nr>
      </partition>
      <partition>
        <create config:type="boolean">false</create>
```

```
                   <format config:type="boolean">false</format>
                   <partition_nr config:type="integer">2</partition_nr>
                   <mount>/space</mount>
                </partition>
                <partition>
                  <create config:type="boolean">false</create>
                  <format config:type="boolean">true</format>
                  <filesystem config:type="symbol">swap</filesystem>
                  <partition_nr config:type="integer">5</partition_nr>
                  <mount>swap</mount>
                </partition>
                <partition>
                  <format config:type="boolean">true</format>
                  <mount>/space2</mount>
                  <size>50mb</size>
                </partition>
                <partition>
                  <format config:type="boolean">true</format>
                  <mount>/space3</mount>
                  <size>max</size>
                </partition>
              </partitions>
              <use>6</use>
          </drive>
        </partitioning>
```

The last example requires exact knowledge about the existing partition table and about the partition numbers of those partitions that should be kept. In some cases however, such data might be not available, especially in a mixed hardware environment with different hard drive types and configurations. The following scenario is for a system with a non-Linux OS with a designated area for a Linux installation.

Figure 4.1. Keeping partitions

In this scenario and as shown in figure "Keeping partitions" , AutoYaST should not in any case create any new partitions, instead it should search for certain partition types on the system and use them according to the partitioning plan in the control file. No partition numbers are given in this case, only the mount points and the partition types (Additional configuration data can be provided, for example file system options, encryption and filesystem type)

Example 4.12. Auto-detection of partitions to be kept.

```
    <partitioning config:type="list">
      <drive>
        <partitions config:type="list">
          <partition>
            <create config:type="boolean">false</create>
            <format config:type="boolean">true</format>
            <mount>/</mount>
            <partition_id config:type="integer">131</partition_id>
          </partition>
          <partition>
            <create config:type="boolean">false</create>
            <format config:type="boolean">true</format>
            <filesystem config:type="symbol">swap</filesystem>
            <partition_id config:type="integer">130</partition_id>
            <mount>swap</mount>
          </partition>
        </partitions>
      </drive>
    </partitioning>
```

4.4.6. Using existing mount table (fstab)

New Feature

This option will allow the AutoYaST to use an existing `/etc/fstab` and use the partition data

from from a previous installation. All partitions are kept and no new partitions are created. The found partitions will be formatted and mounted as specified in `/etc/fstab` found on a Linux root partition.

Although the default behaviour is to format all partitions, it is also possible to leave some partitions untouched and only mount them, for example data partitions. If multiple installations are found on the system (multiple root partitions with different *fstab* files, the installation will abort, unless the desired root partition is configured in the control file. The following example illustrates how this option can be used:

Example 4.13. Reading existing `/etc/fstab`

```
<partitioning_advanced>
  <fstab>
    <!-- Read data from existing fstab. If multiple root partitions are
    found, use the one specified below. Otherwise the first root
    partition is taken -->
    <!-- <root_partition>/dev/hda5</root_partition> -->
    <use_existing_fstab config:type="boolean">true</use_existing_fstab>
    <!-- all partitions found in fstab will be formatted and mounted
    by default unless a partition is listed below with different
    settings -->
    <partitions config:type="list">
      <partition>
        <format config:type="boolean">false</format>
        <mount>/bootmirror</mount>
      </partition>
    </partitions>
  </fstab>
</partitioning_advanced>
```

4.4.7. Logical Volume Manager (LVM)

To configure LVM, first you need to create a *physical volume* using the normal partitioning method described above.

Example 4.14. Create LVM Physical Volume

The following example shows how to prepare for LVM in the *partitioning* resource:

```
<partitioning config:type="list">
  <drive>
    <device>/dev/sda</device>

    <partitions config:type="list">
      <partition>
        <lvm_group>system</lvm_group>
        <partition_type>primary</partition_type>
        <size>max</size>
      </partition>
    </partitions>
    <use>all</use>
  </drive>
</partitioning>
```

The last example will create a non-formatted partition on device `/dev/sda1` of the type *LVM* and with the volume group *system*. The partition created will use all available space on this drive.

Example 4.15. LVM Logical Volumes (New syntax)

```
<partitioning config:type="list">
  <drive>
```

```
      <device>/dev/sda</device>
      <partitions config:type="list">
        <partition>
          <lvm_group>system</lvm_group>
          <partition_type>primary</partition_type>
          <size>max</size>
        </partition>
      </partitions>
      <use>all</use>
    </drive>
    <drive>
      <device>/dev/system</device>
      <is_lvm_vg config:type="boolean">true</is_lvm_vg>
      <partitions config:type="list">
        <partition>
          <filesystem config:type="symbol">reiser</filesystem>
          <lv_name>user_lv</lv_name>
          <mount>/usr</mount>
          <size>500mb</size>
        </partition>
        <partition>
          <filesystem config:type="symbol">reiser</filesystem>
          <lv_name>opt_lv</lv_name>
          <mount>/opt</mount>
          <size>1500mb</size>
        </partition>
        <partition>
          <filesystem config:type="symbol">reiser</filesystem>
          <lv_name>var_lv</lv_name>
          <mount>/var</mount>
          <size>200mb</size>
        </partition>
      </partitions>
      <pesize>4M</pesize>
      <use>all</use>
    </drive>
</partitioning>
```

With SUSE Linux 10.1 and all following versions, it's possible to set the *size* to *max* for the logical volumes. Of course, you can only use *max* only for one(!) logical volumes. You can't have two logical volumes in one volume group with the *size* set to *max*

4.4.8. **Enterprise Volume Management System (EVMS)**

Since SLES10 autoyast has EVMS support.

Using EVMS is quite similar to using LVM (see above). So switching from LVM to EVMS is just a small change in the autoyast profile. All you have to do is to change the "is_lvm_vg" element into "is_evms_vg" and the "lvm_group" element into "evms_group".

With autoyast it's not possible to mix LVM and EVMS.

Using the LVM example from above for EVMS now looks like this:

Example 4.16. EVMS Logical Volumes

```
<partitioning config:type="list">
  <drive>
    <device>/dev/sda</device>
    <partitions config:type="list">
      <partition>
        <evms_group>system</evms_group>
        <partition_type>primary</partition_type>
        <size>max</size>
      </partition>
    </partitions>
    <use>all</use>
  </drive>
  <drive>
    <device>/dev/system</device>
    <is_evms_vg config:type="boolean">true</is_evms_vg>
    <partitions config:type="list">
      <partition>
        <filesystem config:type="symbol">reiser</filesystem>
        <lv_name>user_lv</lv_name>
        <mount>/usr</mount>
        <size>500mb</size>
      </partition>
      <partition>
```

```
            <filesystem config:type="symbol">reiser</filesystem>
            <lv_name>opt_lv</lv_name>
            <mount>/opt</mount>
            <size>1500mb</size>
          </partition>
          <partition>
            <filesystem config:type="symbol">reiser</filesystem>
            <lv_name>var_lv</lv_name>
            <mount>/var</mount>
            <size>200mb</size>
          </partition>
        </partitions>
        <pesize>4M</pesize>
        <use>all</use>
      </drive>
    </partitioning>
```

4.4.9. Software RAID

Using AutoYaST, you can create and assemble software RAID devices. The supported RAID levels are the following:

- *RAID 0:* This level increases your disk performance. There is *NO* redundancy in this mode. If one of the drives crashes, data recovery will not be possible.

- *RAID 1:* This mode has the best redundancy. It can be used with two or more disks. This mode maintains an exact copy of all data on all disks. As long as at least one disk is still working, no data is lost. The partitions used for this type of RAID should have approximately the same size.

- *RAID 5:* This mode combines management of a larger number of disks and still maintains some redundancy. This mode can be used on three disks or more. If one disk fails, all data is still intact. If two disks fail simultaneously, all data is lost.

- *Multipath:* This mode allow access to the same physical device over multiple controller for redundancy against a fault in a controller card. This mode can be used with at least two devices.

As with LVM, you need to create all *RAID* partitions first and assign the partitions to the RAID device you want to create and additionally you need to specify whether a partition or a device should be configured in the RAID or if it should configured as a *Spare* device.

The following example shows a simple RAID1 configuration:

Example 4.17. RAID1 configuration

```
<partitioning config:type="list">
  <drive>
    <device>/dev/sda</device>
    <partitions config:type="list">
      <partition>
        <partition_id config:type="integer">253</partition_id>
        <format config:type="boolean">false</format>
        <raid_name>/dev/md0</raid_name>
        <raid_type>raid</raid_type>
        <size>4gb</size>
      </partition>

      <!-- Here come the regular partitions, i.e. / and swap -->
    </partitions>
    <use>all</use>
  </drive>
  <drive>
    <device>/dev/sdb</device>

    <partitions config:type="list">
      <partition>
        <format config:type="boolean">false</format>
        <partition_id config:type="integer">253</partition_id>
        <raid_name>/dev/md0</raid_name>
        <raid_type>raid</raid_type>
        <size>4gb</size>
      </partition>
```

```
      </partitions>
      <use>all</use>
    </drive>
    <drive>
      <device>/dev/md</device>
      <partitions config:type="list">
        <partition>
          <filesystem config:type="symbol">reiser</filesystem>
          <format config:type="boolean">true</format>
          <mount>/space</mount>
          <partition_id config:type="integer">131</partition_id>
          <partition_nr config:type="integer">0</partition_nr>
          <raid_options>
            <chunk_size>4</chunk_size>
            <parity_algorithm>left-asymmetric</parity_algorithm>
            <raid_type>raid1</raid_type>
          </raid_options>

        </partition>
      </partitions>
      <use>all</use>
    </drive>

</partitioning>
```

The following has to be taken into consideration when configuring raid:

- The device for raid is always */dev/md*

- The property *partition_nr* is used to determine the MD device number. if *partition_nr* is equal to 0, then */dev/md0* is configured.

- All RAID specific options are contained in the *raid_options* resource.

4.5. Software

4.5.1. Package Selections until SUSE Linux 10.1 (not SLES10)

You can install software on the new system using a pre-defined package base selection, i.e. Minimal, Minimal+X11, default etc. in addition to several *Add-on* selections. Check the first CD for available selections in `suse/setup/descr/selections`, depending on the product you are installing, some "well-known" selections might not be available.

In the control file, packages and package selections are described as the following:

Example 4.18. Package selection in control file

```
<software>

  <addons  config:type="list">
    <addon>Kde</addon>
  </addons>
  <base>Minimal</base>
  <packages  config:type="list">
    <package>apache</package>
    <package>sendmail</package>
  </packages>
</software>
```

You can install one base selection and additionally one or multiple add-on selections.

When installing from a CD-ROM, needed packages from other CD-ROMs are installed after the initial boot of the system in the second installation phase. If you are installing packages from multiple

CD-ROMs, then auto-installation has to be interrupted for switching the CD-ROMs. In the case of NFS installation, all packages are installed at first stage of the installation only if the NFS repository is configured as a single medium.

It is often required, that a package should be installed in the second phase, especially custom packages which may contain scripts for configuring the system. This can be done using the *post-packages* resource.

4.5.2. Package Selections with patterns (SLES10 and SUSE Linux 10.2)

SLES10 no longer supports *selections* but uses *patterns* now. Autoyast is not be able to convert selections into patterns and so you have to do that on your own. If you want to use a SLES9 autoyast profile to install a SLES10 server, you have to remove all *addon* entries and the *base* entry. Patterns are configured like this:

Example 4.19. Package selection in control file with patterns

```
<software>
  <patterns config:type="list">
      <pattern>directory_server</pattern>
  </patterns>
  <packages  config:type="list">
    <package>apache</package>
    <package>sendmail</package>
  </packages>
</software>
```

As you can see, the *packages* section is still the same like on a SLES9. Just the *addon* and *base* is gone.

4.5.3. Custom Package Selections

In addition to the pre-defined selections, you can create custom selections by providing a selection file in the selection directory. (`suse/setup/descr`) The selection files have a special format and any additional selection file must conform to this format, otherwise YaST2 will not be able to read it.

As an example for the selection file, take a look at the files available in the directory `/suse/setup/descr/` on the CD-ROMs.

After creating a selection file, you can add it to the configuration as described earlier in this section. The selection name, for example *My.sel* has to be added to the index files `selections` and `directory.yast` to make it visible to the installer.

The file `My.sel` should have the following format:

Example 4.20. Customized Package selection

```
  # SuSE-Linux-Package-Selection 3.0 -- (c) 2004 SuSE Linux AG
# generated on Thu Apr 15 19:49:04 UTC 2004

=Ver: 3.0

=Sel:  LSB

=Sum: LSB Runtime Environment
=Sum.de: LSB-Laufzeitumgebung

=Cat: addon
```

```
=Vis: true

=Ord: 108

+Ins:
XFree86-Mesa
XFree86-libs
expect
fontconfig
freetype2
gettext
glibc-i18ndata
libgcj
lsb
make
makedev
patch
pax
rsync
-Ins:
```

To use the above selection, the following should be added in the control file:

Example 4.21. Package selection file

```
  ...
 <software>
     <base>My</base>
 </software>
 ...
```

4.5.4. **Installing additional and customized Packages**

In addition to the packages available for installation on the CD-ROMs, you can add external pack-
ages including customized kernels. Customized kernel packages must be compatible to the *SuSE*
packages and must install the kernel files to the same locations.

Unlike earlier versions, to install custom and external packages there is no need for a special re-
source in the control file. Instead you need to re-create the package database and update it with any
new packages or new package versions in the source repository.

A script is provided for this task which will query packages available in the repository and create the
required package database.

Creating a new package database is only needed if new RPMs (i.e. update RPMs) were added. To
re-create the database, use the **/usr/bin/create_package_descr** command. For example, use this
command line to create the package database. (When creating the database, all languages will be re-
set to English).

Example 4.22. Creating package database

```
        cd /usr/local/CDs/LATEST/suse
        create_package_descr  -x PATH_TO_EXTRA_PROV -d /usr/local/CDs/LATEST/suse
```

Change starting from SUSE Linux 9.1/SLES 9

To provide extra dependencies which can not be extracted from the rpm files, an extra
file with missing dependencies is available in the directory suse/setup/descr.

The file `EXTRA_PROV` can be used when recreating the package database using the *-x* option.

In the above example, the directory `/usr/local/CDs/LATEST/suse` contains the architecture dependent and independent packages, i.e. *noarch* and *i586*. This might look different on other architectures.

The advantage of this method is that you can keep an up-to-date repository with fixed and updated package (i.e. from *SuSE* FTP server). Additionally this method makes the creation of custom CD-ROMs easier.

Change starting from SUSE Linux 10.1/SLES 10

With SLES10/SL10.1, the concept of adding own RPMs to an installation source has changed. The *yast/order* and *yast/instorder* is no longer supported. Neither by AutoYaST nor by YaST. To add own RPMs to an installation source (that includes add-on products like the SDK) you have to add a file *add_on_products* to the CD1 of the main product.

```
media_url [path_on_media [product_1 [product_2 [....]]]
```

media_url is URL of the media itself path_on_media is path of the catalog on the media. If not present, / (root) is assumed product_1 and following are the names for products, which should be marked for installation. If no product is mentioned, all products found on the media are selected for installation. For example:

```
http://192.168.66.6/SLES10/sdk/CD1
http://192.168.66.6/SLES10/CD1/updates
```

Besides that *add_on_products* file, you can use the autoyast profile to specify add-on products. For example:

```
<add-on>
 <add_on_products config:type="list">
   <listentry>
     <media_url>http://192.168.66.6/SLES10/CD1/updates</media_url>
     <product>SuSE-Linux-Updates</product>
     <product_dir>/</product_dir>
   </listentry>
 </add_on_products>
</add-on>
```

With that entry in the autoyast profile, the *add_on_products* file is not necessary.

YaST checks the signatures of files on the installation source now. If a *content* file is not signed, during a manual installation YaST asks the user what to do. During an autoinstallation, the installation source gets rejected silently.

If you want to use unsigned installation sources with autoyast, you can turn of the checks with the following configuration in your autoyast profile (part of the *general* section.

The following elements must be between the <general><signature-handling> ... </signature-handling></general> tags.

Table 4.4.

Attribute	Values	Description
accept_unsigned_file	the installer will accept unsigned files like the content file	optional. If left out, autoyast lets yast decide what to do
	`<accept_unsigned_file config:type="boolean">true</accept_unsigned_file>`	

Attribute	Values	Description
accept_file_without_checksum	the installer will accept files without a checksum in the content file `<accept_file_without_checksum config:type="boolean">true</accept_file_without`	optional. If left out, autoyast lets yast decide what to do
accept_verification_failed	the installer will accept files where the verification of the signature failed. So the file was signed but the check failed. `<accept_verification_failed config:type="boolean">true</accept_verification_f`	optional. If left out, autoyast lets yast decide what to do
accept_unknown_gpg_key	the installer will accept new gpg keys on the installation source that are used to sign the content file for example `<accept_unknown_gpg_key config:type="boolean">true</accept_unknown_gpg_key>`	optional. If left out, autoyast lets yast decide what to do
accept_non_trusted_gpg_key	This basically means, we know the key, but it is not trusted `<accept_non_trusted_gpg_key config:type="boolean">true</accept_non_trusted_gp`	optional. If left out, autoyast lets yast decide what to do
import_gpg_key	the installer will accept and import new gpg keys on the installation source in it's database. `<import_gpg_key config:type="boolean">true</import_gpg_key>`	optional. If left out, autoyast lets yast decide what to do

4.5.5. Kernel packages

Kernel packages are not part of any selection. The required kernel is determined during installation. If the kernel package is added to any selection or to the individual package selection, installation will mostly fail due to conflicts.

To force the installation of a specific kernel, use the *kernel* property. The following is an example forcing the installation of the default kernel. In this example this kernel will be installed in any case, even if an SMP or other kernel is required

Example 4.23. Package selection in control file

```
<software>
  <addons  config:type="list">
    <addon>Kde</addon>
  </addons>
  <base>Minimal</base>
  <kernel>kernel-default</kernel>
  <packages  config:type="list">
    <package>apache2</package>
  </packages>
</software>
```

4.5.6. Removing automatically selected packages

Some packages are selected automatically either because of a dependency or because it available in a selection.

Removing such packages might break the system consistency and it is not recommended to remove basic packages unless a replacement which provides same services is provided. Best example for this case are MTA packages. By default, *postfix* will be selected and installed. If you wish however to use another MTA like *sendmail*, then postfix can be removed from the list of selected package using a list in the software resource. The following example shows how this can be done:

Example 4.24. Package selection in control file

```
<software>
  <addons  config:type="list">
    <addon>Kde</addon>
  </addons>
  <base>Minimal</base>
  <packages  config:type="list">
    <package>sendmail</package>
  </packages>
  <remove-packages  config:type="list">
    <package>postfix</package>
  </remove-packages>
</software>
```

4.6. Services and Run-levels

With the run-level resource you can set the default run-level and specify in detail which system services you want to be started in which run-level.

The default property specifies the *default* run level of the system. Changes to the default run-level will take effect the next time you boot the system. After installation is completed, the system has run-level 5, which is *Full multiuser with network and XDM*. If you have configured a system with no X11, then it is recommended to reboot the system after the first stage using the *reboot* property in the *general* resource.

A service should run in using a space delimited list of the run-levels as shown in the following example. An alternative to specifying the exact run-levels is to change the status of the service by either enabling or disabling it using the *service_status* property.

Example 4.25. Run-level Configuration

```
....
  <runlevel>
   <default>3</default>
   <services config:type="list" >
    <service>
     <service_name>at</service_name>
     <service_start>3 5</service_start>
    </service>
    <service>
     <service_name>portmap</service_name>
     <service_status>enable</service_status>
    </service>
    <service>
     <service_name>hwscan</service_name>
     <service_status>disable</service_status>
    </service>
   </services>
  </runlevel>
....
```

4.7. Network configuration

4.7.1. Network devices, DNS and Routing.

Network configuration is used to connect a single *SuSE* Linux workstation to an Ethernet-based LAN or to configure dial-up connection. More complex configuration (multiple network cards, routing, etc.) is also provided. With this module it's possible to configure and setup Ethernet Controllers and Token-Ring Controllers.

To configure network settings and activate networking automatically, one global resource is used to store the whole network configuration.

Example 4.26. Network configuration

```
.....
    <networking>
      <dns>
        <dhcp_hostname config:type="boolean">true</dhcp_hostname>
        <dhcp_resolv config:type="boolean">true</dhcp_resolv>
        <domain>local</domain>
        <hostname>linux</hostname>
      </dns>
      <interfaces config:type="list">
        <interface>
          <bootproto>dhcp</bootproto>
          <device>eth0</device>
          <startmode>onboot</startmode>
        </interface>
      </interfaces>
      <routing>
        <ip_forward config:type="boolean">false</ip_forward>
        <routes config:type="list">
          <route>
            <destination>default</destination>
            <device>-</device>
            <gateway>192.168.1.240</gateway>
            <netmask>-</netmask>
          </route>
        </routes>
      </routing>
      <modules config:type="list">
        <module_entry>
          <device>eth0</device>
          <module>e100</module>
          <options></options>
        </module_entry>
      </modules>
    </networking>
....
```

4.7.2. Proxy

Configure your Internet proxy (caching) settings using this resource.

HTTP proxy is the name of the proxy server for your access to the world wide web (WWW). *FTP proxy* is the name of the proxy server for your access to the file transfer services (FTP). *No proxy* domains is a list of domains for which the requests should be done directly without caching.

If you are using a proxy server with authorization, fill in Proxy user name and Proxy password.

Example 4.27. Netwrok configuration: Proxy

```
<?xml version="1.0"?>
<!DOCTYPE profile SYSTEM "/usr/share/autoinstall/dtd/profile.dtd">
<profile xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
    <proxy>
        <enabled config:type="boolean">true</enabled>
        <ftp_proxy>http://192.168.1.240:3128</ftp_proxy>
        <http_proxy>http://192.168.1.240:3128</http_proxy>
```

```
          <no_proxy>localhost</no_proxy>
          <proxy_password>testpw</proxy_password>
          <proxy_user>testuser</proxy_user>
       </proxy>
</profile>
```

4.7.3. (X)Inetd

The profile has elements to specify which superserver should be used (netd_service), whether it should be enabled (netd_status) and how the services should be configured (netd_conf).

A service description element has conceptually two parts: key and non-key. When writing the configuration, services are matched using the key fields and to the matching service, non-key fields are applied. If no service matches, it is created. If more services match, a warning is reported. The key fields are script, service, protocol and server.

Service and protocol are matched literally. script is the base name of the config file: usually a file in `/etc/xinetd.d`, for example "echo-udp", or "inetd.conf". For compatibility with 8.2, server is matched more loosely: if it is `/usr/sbin/tcpd`, the real server name is taken from server_args. After that, the basename of the first whitespace-sparated word is taken and these values are compared.

Example 4.28. Inetd Example

```
<profile>
  ...
  <inetd>
    <netd_service config:type="symbol">xinetd</netd_service>
    <netd_status config:type="integer">0</netd_status>
    <netd_conf config:type="list">
      <conf>
        <script>imap</script>
        <service>pop3</service>
        <enabled config:type="boolean">true</enabled>
      </conf>
      <conf>
        <server>in.ftpd</server>
        <server_args>-A</server_args>
        <enabled config:type="boolean">true</enabled>
      </conf>
      <conf>
        <service>daytime</service>
        <protocol>tcp</protocol>
      </conf>
      ....
      <conf>...</conf>
    </netd_conf>
  </inetd>
  ...
</profile>
```

4.7.4. NIS

Using the *nis* resource, you can configure the target machine as a *NIS client*. The following example shows a detailed configuration using multiple domains.

Example 4.29. Network configuration: NIS

```
...
    <nis>
      <nis_broadcast config:type="boolean">true</nis_broadcast>
      <nis_broken_server config:type="boolean">true</nis_broken_server>
      <nis_by_dhcp config:type="boolean">false</nis_by_dhcp>
```

```
        <nis_domain>test.com</nis_domain>
        <nis_local_only config:type="boolean">true</nis_local_only>
        <nis_options></nis_options>
        <nis_other_domains config:type="list">
          <nis_other_domain>
            <nis_broadcast config:type="boolean">false</nis_broadcast>
            <nis_domain>domain.com</nis_domain>
            <nis_servers config:type="list">
              <nis_server>10.10.0.1</nis_server>
            </nis_servers>
          </nis_other_domain>
        </nis_other_domains>
        <nis_servers config:type="list">
          <nis_server>192.168.1.1</nis_server>
        </nis_servers>
        <start_autofs config:type="boolean">true</start_autofs>
        <start_nis config:type="boolean">true</start_nis>
      </nis>
...
```

4.7.5. LDAP client

The installed machine can be set up as an > *LDAP client* to authenticate users with an OpenLDAP;
server. Required data are the name of the search base (base DN, e.g, dc=mydomain,dc=com) and the
IP address of the LDAP server (e.g., 10.20.0.2).

If LDAP is activated, *NSS* and *PAM* will be configured accordingly to use LDAP for user authentication.

Example 4.30. Network configuration: LDAP client

```
...
    <ldap>
        <ldap_domain> dc=mydomain,dc=com</ldap_domain>
        <ldap_server>10.10.0.1</ldap_server>
        <ldap_tls config:type="boolean">true</ldap_tls>
        <ldap_v2 config:type="boolean">true</ldap_v2>
        <pam_password>crypt</pam_password>
        <start_ldap config:type="boolean">true</start_ldap>
    </ldap>
...
```

4.7.6. NFS Client and Server

Configuration of a system as an NFS client or an NFS server is possible and can be done using the
configuration system. The following example shows how both NFS client and server can be configured.

Example 4.31. Network configuration: NFS client

```
  ...
    <nfs config:type="list">
      <nfs_entry>
        <mount_point>/home</mount_point>
        <nfs_options>defaults</nfs_options>
        <server_path>192.168.1.1:/home</server_path>
      </nfs_entry>
    </nfs>
...
```

Example 4.32. Network configuration: NFS Server

```
    ....
    <nfs_server>
      <nfs_exports config:type="list">
        <nfs_export>
          <allowed config:type="list">
            <allowed_clients>*(ro,root_squash,sync)</allowed_clients>
          </allowed>
          <mountpoint>/home</mountpoint>
        </nfs_export>
        <nfs_export>
          <allowed config:type="list">
            <allowed_clients>*(ro,root_squash,sync)</allowed_clients>
          </allowed>
          <mountpoint>/work</mountpoint>
        </nfs_export>
      </nfs_exports>
      <start_nfsserver config:type="boolean">true</start_nfsserver>
    </nfs_server>
....
```

4.7.7. NTP Client

Select whether to start the NTP daemon when booting the system. The NTP daemon resolves host names when initializing. The first synchronization of the clock is performed before the NTP daemon is started. To use this host for initial synchronization configure the property *initial_sync*.

To run NTP daemon in chroot jail, set *start_in_chroot*. Starting any daemon in a chroot jail is more secure and strongly recommended. To adjust NTP servers, peers, local clocks, and NTP broadcasting, add the appropriate entry to the control file. an example of various configuration options is shown below.

Example 4.33. Network configuration: NTP Client

```
<?xml version="1.0"?>
<!DOCTYPE profile SYSTEM "/usr/share/autoinstall/dtd/profile.dtd">
<profile xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
    <ntp-client>
      <configure_dhcp config:type="boolean">false</configure_dhcp>
      <peers config:type="list">
        <peer>
          <address>ntp1.example.com</address>
          <initial_sync config:type="boolean">true</initial_sync>
          <options></options>
          <type>server</type>
        </peer>
      </peers>
      <start_at_boot config:type="boolean">true</start_at_boot>
      <start_in_chroot config:type="boolean">true</start_in_chroot>
    </ntp-client>
</profile>
```

4.8. Mail Configuration (Sendmail or Postfix)

For the mail configuration of the client this module lets you create a detailed mail configuration. The module contains various options and it is recommended to use it at least for the initial configuration.

Example 4.34. Mail Configuration

```
...
  <mail>
    <aliases config:type="list">
      <alias>
        <alias>root</alias>
        <comment></comment>
        <destinations>foo</destinations>
      </alias>
      <alias>
        <alias>test</alias>
        <comment></comment>
        <destinations>foo</destinations>
      </alias>
    </aliases>
    <connection_type config:type="symbol">permanent</connection_type>
    <fetchmail config:type="list">
      <fetchmail_entry>
        <local_user>foo</local_user>
        <password>bar</password>
        <protocol>POP3</protocol>
        <remote_user>foo</remote_user>
        <server>pop.foo.com</server>
      </fetchmail_entry>
      <fetchmail_entry>
        <local_user>test</local_user>
        <password>bar</password>
        <protocol>IMAP</protocol>
        <remote_user>test</remote_user>
        <server>blah.com</server>
      </fetchmail_entry>
    </fetchmail>
    <from_header>test.com</from_header>
    <listen_remote config:type="boolean">true</listen_remote>
    <local_domains config:type="list">
      <domains>test1.com</domains>
    </local_domains>
    <masquerade_other_domains config:type="list">
      <domain>blah.com</domain>
    </masquerade_other_domains>
    <masquerade_users config:type="list">
      <masquerade_user>
        <address>joe@test.com</address>
        <comment></comment>
        <user>joeuser</user>
      </masquerade_user>
      <masquerade_user>
        <address>bar@test.com</address>
        <comment></comment>
        <user>foo</user>
      </masquerade_user>
    </masquerade_users>
    <mta config:type="symbol">postfix</mta>
    <outgoing_mail_server>test.com</outgoing_mail_server>
    <postfix_mda config:type="symbol">local</postfix_mda>
    <smtp_auth config:type="list">
      <listentry>
        <password>bar</password>
        <server>test.com</server>
        <user>foo</user>
      </listentry>
    </smtp_auth>
    <use_amavis config:type="boolean">true</use_amavis>
    <virtual_users config:type="list">
      <virtual_user>
        <alias>test.com</alias>
        <comment></comment>
        <destinations>foo.com</destinations>
      </virtual_user>
      <virtual_user>
        <alias>geek.com</alias>
        <comment></comment>
        <destinations>bar.com</destinations>
      </virtual_user>
    </virtual_users>
  </mail>
...
```

4.9. Security settings

Using the features of this module, you will be able to change the local security settings on the target system. The local security settings include the boot configuration, login settings, password settings, user addition settings, and file permissions.

Configuring the security settings automatically corresponds to the *Custom Settings* in the security module available in the running system which lets you create your own, customized configuration.

Example 4.35. Security configuration

See the reference for the meaning and the possible values of the settings in the following example.

```
...
   <security>
      <console_shutdown>ignore</console_shutdown>
      <cwd_in_root_path>no</cwd_in_root_path>
      <displaymanager_remote_access>no</displaymanager_remote_access>
      <fail_delay>3</fail_delay>
      <faillog_enab>yes</faillog_enab>
      <gid_max>60000</gid_max>
      <gid_min>101</gid_min>
      <kdm_shutdown>root</kdm_shutdown>
      <lastlog_enab>yes</lastlog_enab>
      <encryption>md5</encryption>
      <obscure_checks_enab>no</obscure_checks_enab>
      <pass_max_days>99999</pass_max_days>
      <pass_max_len>8</pass_max_len>
      <pass_min_days>1</pass_min_days>
      <pass_min_len>6</pass_min_len>
      <pass_warn_age>14</pass_warn_age>
      <passwd_use_cracklib>yes</passwd_use_cracklib>
      <permission_security>secure</permission_security>
      <run_updatedb_as>nobody</run_updatedb_as>
      <uid_max>60000</uid_max>
      <uid_min>500</uid_min>
   </security>
...
```

4.9.1. Password Settings Options

Change various password settings. These settings are mainly stored in the `/etc/login.defs` file.

Use this resource to activate one of the *encryption* methods currently supported. If not set, *DES* is configured.

DES, the Linux default method, works in all network environments, but it restricts you to passwords no longer than eight characters. *MD5* allows longer passwords, thus provides more security, but some network protocols don't support this, and you may have problems with NIS. *Blowfish* is also supported.

Additionally, you can setup the system to check for password plausibility and length etc.

4.9.2. Boot Settings

Use the security resource, you can change various boot settings.

- *How to interpret Ctrl + Alt + Del*

  When someone at the console has pressed the CTRL + ALT + DEL key combination, the system usually reboots. Sometimes it is desirable to ignore this event, for example, when the system serves as both workstation and server.

- *Shutdown behavior of KDM*

  Set who is allowed to shut down the machine from KDM.

4.9.3. Login Settings

Change various login settings. These settings are mainly stored in the '/etc/login.defs' file.

4.9.4. New user settings (useradd settings)

Set the minimum and maximum possible user ID and set the minimum and maximum possible group ID.

4.10. Monitor and X11 Configuration

FIXME

Example 4.36. X11 and Monitor configuration

```
...
  <x11>
    <color_depth>16</color_depth>
    <configure_x11 config:type="boolean">true</configure_x11>
    <display_manager>kde</display_manager>
    <enable_3d config:type="boolean">false</enable_3d>
    <monitor>
      <display>
        <frequency config:type="integer">60</frequency>
        <max_hsync config:type="integer">97</max_hsync>
        <max_vsync config:type="integer">180</max_vsync>
        <min_hsync config:type="integer">30</min_hsync>
        <min_vsync config:type="integer">50</min_vsync>
        <width config:type="integer">1024</width>
      </display>
      <monitor_device>G90F</monitor_device>
      <monitor_vendor>VIEWSONIC</monitor_vendor>
    </monitor>
    <resolution>1600x1200,1280x1024,1024x768,800x600,640x480</resolution>
    <window_manager>kdm</window_manager>
  </x11>

...
```

4.11. Users

The root user and at least one normal user can be added during install using data supplied in the control file. User data and passwords (encrypted or in clear text) are part of the *configure* resource in the control file.

At least the root user should be configured during auto-installation, which will insure you will be able to login after installation is finished and of course it will insure nobody else can login into the system (in case the password is not set).

The two users in the following example are added during system configuration.

Example 4.37. User configuration

```
...
  <users config:type="list">
    <user>
      <username>root</username>
      <user_password>password</user_password>
      <encrypted config:type="boolean">true</encrypted>
      <forename/>
      <surname/>
    </user>
    <user>
      <username>nashif</username>
      <user_password>password</user_password>
      <encrypted config:type="boolean">true</encrypted>
      <forename>Anas</forename>
      <surname>Nashif</surname>
    </user>
  </users>
...
```

The last example shows the minimal information required for adding users. More options are available for a more customized user account management. The data in `/etc/default/useradd` is used to determine the home directory of the user to be created in addition to other parameters.

4.12. Custom user scripts

By adding scripts to the auto-installation process you can customize the installation for your needs and take control in different stages of the installation.

In the auto-installation process, four types of scripts can be executed and they will be described here in order of "appearance" during the installation.

- *pre-scripts* (very early, before anything else really happened)

- *chroot-scripts* (after the package installation, before the first boot)

- *post-scripts* (during the first boot of the installed system, no services running)

- *init-scripts* (during the first boot of the installed system, all servies up and running)

4.12.1. Pre-Install Scripts

Executed before YaST2 does any real change to the system (Before partitioning and package installation but after the hardware detection)

You can use the pre-script to modify your profile and let autoyast read it again. If you want to do that, you can find your profile in "/tmp/profile/autoinst.xml". Do what you want to do with that file and store the modified version in "/tmp/profile/modified.xml". Autoyast will read that modified script then again after the pre-script is done.

With SUSE Linux 10.0 and all following versions it's possible to change the partitioning with fdisk in your pre-script. With pre 10.0 versions of SUSE Linux (like SLES9), that was not possible.

Pre-Install Scripts with confirmation

Pre-scripts are executed at an early stage of the installation. This means if you have requested to confirm the installation, the pre-scripts will be executed before the confirmation screen shows up. (*profile/install/general/mode/confirm*)

The following elements must be between the <pre-scripts config:type="list"><script> ... </script></pre-scripts> tags

Table 4.5. pre script XML representation

Element	Description	Comment
location	you can define a location from where the script gets fetched. Locations can be the same like for the profile (http,ftp,nfs,...). `<location>http://10.10.0.1/myPreScript.sh</location>`	either <location> or <source> must be defined
source	the script itself. The source code	Either <location> or <source>

Element	Description	Comment
	of the script if you want so. Encapsulated in a CDATA tag. If you don't want to put the whole shell script into the XML profile, look at the location parameter. ```\n<source>\n<![CDATA[\necho "Testing the pre script" >\n]]>\n</source>\n```	must be defined /tmp/pre-script_out.txt
interpreter	the interpreter that must be used for the script. Supported options are shell and perl. `<interpreter>perl</interpreter>`	optional (default is shell)
filename	the filename of the script. It will be stored in a temporary directory under /tmp/... `<filename>myPreScript5.sh</filename>`	optional. The default is the type of the script (pre-scripts) in this case
feedback	if this boolean is true, stdout and stderr of the script will be shown in a popup that the user has to confirm via ok-button. If stdout and stderr are empty, no popup is shown and so no confirmation is needed. `<feedback config:type="boolean">true</feedback>`	optional. The default is false. This option was invented with SL 10.1 / SLES10
debug	if this is true, every single line of a shell script is logged. Perl scripts are run with warnings turned on. `<debug config:type="boolean">true</debug>`	optional. The default is true. This option was invented with SL 10.1 / SLES10

4.12.2. Chroot environment scripts

Chroot scripts are executed before the machine reboots for the first time. Actually chroot scripts are two differnt kind of script with one name. You can execute chroot script before the installation chroots into the installed system and configures the boot loader and you can execute a script after the chroot into the installed system has happend (look at the "chrooted" parameter for that). Both types of scripts are executed before yast2 boots for the first time.

The following elements must be between the <chroot-scripts config:type="list"><script> ... </script></chroot-scripts> tags

Table 4.6. chroot script XML representation

Element	Description	Comment
location	you can define a location from where the script gets fetched.	either <location> or <source> must be defined

Element	Description	Comment
	Locations can be the same like for the profile (http,ftp,nfs,...). `<location>http://10.10.0.1/myChrootScript.sh</location>`	

source	the script itself. The source code of the script if you want so. Encapsulated in a CDATA tag. If you don't want to put the whole shell script into the XML profile, look at the location parameter.
	
	`<source>`
	`<![CDATA[`
	`echo "Testing the chroot script"`
	`]]>`
	`</source>`
		either `<location>` or `<source>` must be defined
		
		`> /tmp/chroot_out.txt`

chrooted	this value can be true or false. "False" means that the installed system is still mounted at "/mnt" and no chrooting has happened till now. The bootloader is not installed too at that stage. "True" means, we did a chroot into /mnt, so we are now in the installed system. The bootloader is installed and if you want to change anything in the installed system, you don't have to use the "/mnt/" prefix anymore. `<chrooted config:type="boolean">true</chrooted>`	optional (the default is false)
interpreter	the interpreter that must be used for the script. Supported options are shell and perl.and if you are in a chrooted=true condition, you can use python too if it's installed. `<interpreter>perl</interpreter>`	optional (default is shell)
filename	the filename of the script. It will be stored in a temporary directory under /tmp/... `<filename>myPreScript5.sh</filename>`	optional. The default is the type of the script (pre-scripts) in this case
feedback	if this boolean is true, stdout and stderr of the script will be shown in a popup that the user has to confirm via ok-button. If stdout and stderr are empty, no popup is shown and so no confirmation is needed. `<feedback config:type="boolean">true</feedback>`	optional. The default is false. This option was invented with SL 10.1 / SLES10
debug	if this is true, every single line	optional. The default is true.

Element	Description	Comment
	of a shell script is logged. Perl scripts are run with warnings turned on. `<debug config:type="boolean">true</debug>`	This option was invented with SL 10.1 / SLES10

4.12.3. Post-Install Scripts

These scripts are executed after AutoYaST has completed the system configuration and after it has booted the system for the first time.

Starting from SLES9, network is not available during post-installation script execution. To access the network, network device has to be configured in the script.

It is possible to execute the post scripts in an earlier phase while the installation network is still up and before AutoYaST configures the system. To run network enabled post scripts, the boolean property *network_needed* has to be set to true.

The following elements must be between the <post-scripts config:type="list"><script> ... </script></post-scripts> tags

Table 4.7. post script XML representation

Element	Description	Comment
location	you can define a location from where the script gets fetched. Locations can be the same like for the profile (http,ftp,nfs,...) but then you need a running network interface of course `<location>http://10.10.0.1/myPostScript.sh</location>`	either <location> or <source> must be defined

source	the script itself. The source code of the script if you want so. Encapsulated in a CDATA tag. If you don't want to put the whole shell script into the XML profile, look at the location parameter.
	
	`<source>`
	`<![CDATA[`
	`echo "Testing the chroot script" > /tmp/chroot_out.txt`
	`]]>`
	`</source>`
		either <location> or <source> must be defined

network_needed	this value can be true or false. On "false" the script will run after the yast modules like the user configuration and everything else are done. The network is configured but still not up and running. With this value on "true", the script runs before(!) all yast modules are configured. So there is no local user and no network is con-	optional (the default is false)

Element	Description	Comment
	figured but the installation network is still up and running (of course only if you did a network installation). `<network_needed config:type="boolean">true</network_needed>`	
interpreter	the interpreter that must be used for the script. Supported options are shell, perl and python if it's installed. `<interpreter>perl</interpreter>`	optional (default is shell)
filename	the filename of the script. It will be stored in a temporary directory under /tmp/... `<filename>myPostScript5.sh</filename>`	optional. The default is the type of the script (post-scripts) in this case
feedback	if this boolean is true, stdout and stderr of the script will be shown in a popup that the user has to confirm via ok-button. If stdout and stderr are empty, no popup is shown and so no confirmation is needed. `<feedback config:type="boolean">true</feedback>`	optional. The default is false. This option was invented with SL 10.1 / SLES10
debug	if this is true, every single line of a shell script is logged. Perl scripts are run with warnings turned on. `<debug config:type="boolean">true</debug>`	optional. The default is true. This option was invented with SL 10.1 / SLES10

4.12.4. Init Scripts

Note

Available starting from SLES9 only.

These scripts are executed during the initial boot process and after YaST2 has finished. The final scripts are executed using a special *init.d* script which is executed only once. The final scripts are executed toward the end of the boot process and after network has been intialized.

Init scripts are configured using the tag *init-scripts* and are run using the special purpose *init.d* script `/etc/init.d/autoyast`.

The following elements must be between the <init-scripts config:type="list"><script> ... </script></init-scripts> tags

Table 4.8. init script XML representation

Element	Description	Comment
location	you can define a location from where the script gets fetched. Locations can be the same like for the profile (http,ftp,nfs,...) but then you need a running network interface of course `<location>http://10.10.0.1/myInitScript.sh</location>`	either <location> or <source> must be defined

source	the script itself. The source code of the script if you want so. Encapsulated in a CDATA tag. If you don't want to put the whole shell script into the XML profile, look at the location parameter.
	
	`<source>`
	`<![CDATA[`
	`echo "Testing the init script" > /tmp/init_out.txt`
	`]]>`
	`</source>`
		either <location> or <source> must be defined

filename	the filename of the script. It will be stored in a temporary directory under /tmp/... `<filename>mynitScript5.sh</filename>`	optional. The default is the type of the script (init-scripts) in this case

When added to the control file manually, the scripts have to be included in a *CDATA* element to avoid confusion with the file syntax and other tags defined in the control file.

4.12.5. Script example

Example 4.38. Post script configuration

```
    <?xml version="1.0"?>
<!DOCTYPE profile SYSTEM "/usr/share/autoinstall/dtd/profile.dtd">
<profile xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
        <scripts>
            <chroot-scripts config:type="list">
                <script>
                    <chrooted config:type="boolean">true</chrooted>
                    <filename>chroot.sh</filename>
                    <interpreter>shell</interpreter>
                    <source><![CDATA[
#!/bin/sh
echo "Testing chroot (chrooted) scripts"
ls
]]>
                    </source>
                </script>
                <script>
                    <filename>chroot.sh</filename>
                    <interpreter>shell</interpreter>
                    <source><![CDATA[
#!/bin/sh
echo "Testing chroot scripts"
df
cd /mnt
ls
]]>
                    </source>
                </script>
            </chroot-scripts>
            <post-scripts config:type="list">
                <script>
                    <filename>post.sh</filename>
                    <interpreter>shell</interpreter>
                    <source><![CDATA[
#!/bin/sh
```

```
echo "Running Post-install script"
/etc/init.d/portmap start
mount -a 192.168.1.1:/local /mnt
cp /mnt/test.sh /tmp
umount /mnt
]]>
                </source>
              </script>
              <script>
                 <filename>post.pl</filename>
                 <interpreter>perl</interpreter>
                 <source><![CDATA[
#!/usr/bin/perl
print "Running Post-install script";

]]>
                </source>
             </script>
           </post-scripts>
           <pre-scripts config:type="list">
               <script>
                 <interpreter>shell</interpreter>
                 <location>http://192.168.1.1/profiles/scripts/prescripts.sh</location>
               </script>
               <script>
                 <filename>pre.sh</filename>
                 <interpreter>shell</interpreter>
                 <source><![CDATA[
#!/bin/sh
echo "Running pre-install script"
]]>
                 </source>
              </script>
           </pre-scripts>
        </scripts>
</profile>
```

After installation is finished, the scripts and the output logs can be found in the directory `/var/adm/autoinstall`. The scripts are located in `scripts` and the output logs of the scripts are located in the `log` directory.

The log is the output resulting when executing the shell scripts using the following command:

```
/bin/sh -x <script_name> 2&> /var/adm/autoinstall/logs/<script_name>.log
```

4.13. System variables (Sysconfig)

Using the sysconfig resource, it is possible to define configuration variables in the sysconfig repository (`/etc/sysconfig`) directly. Sysconfig variables, offer the possibility to fine-tune many system components and environment variables exactly to your needs.

Refer to the handbook for more details about the many configuration options available in `/etc/sysconfig`

The following example shows how a variable can be set using the sysconfig resource.

To configure a variable in a sysconfig file, the following syntax can be used:

Example 4.39. Sysconfig Configuration

```
 <sysconfig config:type="list" >
  <sysconfig_entry>
   <sysconfig_key>XNTPD_INITIAL_NTPDATE</sysconfig_key>
   <sysconfig_path>/etc/sysconfig/xntp</sysconfig_path>
   <sysconfig_value>ntp.host.com</sysconfig_value>
  </sysconfig_entry>
  <sysconfig_entry>
   <sysconfig_key>HTTP_PROXY</sysconfig_key>
   <sysconfig_path>/etc/sysconfig/proxy</sysconfig_path>
```

```
    <sysconfig_value>proxy.host.com:3128</sysconfig_value>
   </sysconfig_entry>
   <sysconfig_entry>
    <sysconfig_key>FTP_PROXY</sysconfig_key>
    <sysconfig_path>/etc/sysconfig/proxy</sysconfig_path>
    <sysconfig_value>proxy.host.com:3128</sysconfig_value>
   </sysconfig_entry>
  </sysconfig>
```

Both relative and absolute pathes can be provided. If no absolute path is given, it is treated as a sysconfig file under the /etc/sysconfig directory.

4.14. Adding complete configurations

For many applications and services you might have prepared a configuration file which should be copied in a complete form to some location in the installed system. This is for example if you are installing a web server and have a *ready to go* server configuration file (httpd.conf).

Using this resource, you can embed the file into the control file by specifying the final path on the installed system. YaST2 will copy this file to the specified location.

Example 4.40. Dumping files into the installed system

```
  <files config:type="list">
     <config_file>
       <file_path>/etc/httpd/httpd.conf</file_path>
       <file_contents>

<![CDATA[
some content
]]>

       </file_contents>
     </config_file>
  </files>
```

A more advanced example is shown below. This configuration will create a file using the content supplied in *file_contents* and will change the permissions and ownership of the file. After the file has been copied to the system, a script is executed which can be used to manipulate the file and prepare it for the environment of the client.

Example 4.41. Dumping files into the installed system

```
  <files config:type="list">
     <config_file>
       <file_path>/etc/someconf.conf</file_path>
       <file_contents>

<![CDATA[
some content
]]>

       </file_contents>
       <file_owner>nashif.users</file_owner>
       <file_permissions>444</file_permissions>
       <file_script>
         <interpreter>shell</interpreter>
         <source>

<![CDATA[
#!/bin/sh

echo "Testing file scripts" >> /etc/someconf.conf
df
cd /mnt
ls
]]>
```

```
        </source>
      </file_script>
    </config_file>
 </files>
```

4.15. Miscellaneous hardware and system components

In addition to the core component configuration, like network authentication and security, AutoY-aST2 offers a wide range of hardware and system configuration which is available by default on any system installed manually and in an interactive way. For example, it is possible to configure print-ers, sound devices, TV cards and any other hardware components which have a module within YaST2.

Any new configuration options that will be added to YaST2 will be automatically available as an auto-installation resource.

4.15.1. Printer

Although Printer configuration, like other configurations can be done manually, it is recommended to use the Configuration System to create such a configuration because of the complexity and the range of options offered by such modules.

Using the configuration system will guarantee that the options provided are consistent. The follow-ing is an example of a configuration section which was created using the configuration system.

Example 4.42. Printer configuration

```
....
    <printer>
      <default>lp</default>
      <printcap config:type="list">
        <printcap_entry>
           <cups-state>void</cups-state>
           <ff config:type="boolean">true</ff>
           <info></info>
           <location></location>
           <lprng-state>changed</lprng-state>
           <name>lp</name>
           <options>
             <job-sheets>none,none</job-sheets>
           </options>
           <raw config:type="boolean">true</raw>
           <type>yast2</type>
           <uri>parallel:/dev/lp0</uri>
        </printcap_entry>
      </printcap>
    </printer>
....
```

4.15.2. Sound devices

An example of sound configuration created using the configuration system is shown below.

Example 4.43. Sound configuration

```
....
    <sound>
      <autoinstall config:type="boolean">true</autoinstall>
```

```
    <modules_conf config:type="list">
      <module_conf>
        <alias>snd-card-0</alias>
        <model>M5451, ALI</model>
        <module>snd-ali5451</module>
        <options>
          <snd_enable>1</snd_enable>
          <snd_index>0</snd_index>
          <snd_pcm_channels>32</snd_pcm_channels>
        </options>
      </module_conf>
    </modules_conf>
    <volume_settings config:type="list">
      <listentry>
        <Master config:type="integer">75</Master>
      </listentry>
    </volume_settings>
  </sound>

....
```

4.16. Ask the user for values during installation

This feature is only available since SUSE Linux 10.1 and SLES10.

You have the option to let the user decide the values of specific parts of the profile during the installation. If you use that feature, a popup will come up during the installation and will ask the user to enter a specific part of the profile. So if you want a full auto installation but you want the user to set the password of the local account, you can do this via the *ask* directive in the profile.

The following elements must be between the <ask-list config:type="list"><ask> ... </ask></ask-list> tags in the <general> section.

Table 4.9. XML representation

Element	Description	Comment
question	The question you want to ask the user. `<question>Enter the LDAP server</question>`	The default value is the path to the element (the path often looks strange, so I recommend to enter a question)
default	you can set a pre-selection for the user. A textentry will be filled out with this value, a checkbox will be "true" or "false" and a selection will have this default "value" pre-selected. `<default>dc=suse,dc=de</default>`	optional
help	An optional helptext that is shown on the left side of the question. `<help>Enter the LDAP server address.</help>`	optional
title	An optional title that is shown above the question. `<title>LDAP server</title>`	optional
type	the type of the element you	optional. The defaul is string. If

Element	Description	Comment
	want to change. Possible values are "symbol","boolean","string" and "integer". The filesystem in the partition section is a symbol, while the "encrypted" element in the user configuration is a boolean. You can see the type of that element if you look in your profile at the config:type="...." attribute. `<type>symbol</type>`	type is "symbol" you must provide the selection element too (see below)
password	if this boolean is set to "true", a password dialog pops up instead of a simple text entry. Setting this to "true" makes only sense if "type" is string. `<password config:type="boolean">true</password>`	optional. The default is "false"
path	The path to the element in the profile. It's a comma seperated list of elements that describes the path to the element you want to change. For example, the ldap server element can be found in the profile in the <ldap><ldap_server> section. So if you want to change that value, you have to set the path to "ldap,ldap_server". If you want to change the password of the first user in the profile, you have to set the path to "users,0,user_password". The "0" indicates the first user in the <users config:type="list"> list of users in the profile. `<path>networking,dns,hostname</path>`	this information is optional but you should at least provie *path* or *file*
file (available since SLES10 SP1 and SL 10.2)	you can store the answer to a question in a file, to use it in one of your scripts later. `<file>/tmp/answer_hostname</file>`	this information is optional but you should at least provie *path* or *file*
password	if this boolean is set to "true", a password dialog pops up instead of a simple text entry. Setting this to "true" makes only sense if "type" is string. `<password config:type="boolean">true</password>`	optional. The default is "false"
stage	stage configures the installation stage where the question pops up. You can set this value to "cont" or "initial". "initial" means the popup comes up very	optional. The default is "initial"

Element	Description	Comment
	early in the installation, short after the pre-script has run. "cont" means, that the dialog with the question comes after the first reboot, when the system boots for the very first time. Questions you answer during the "inital" stage, will write their answer into the profile on the harddisk. You should know that if you enter cleartext passwords during "initial". Of course it does not make sense to ask for a filesystem to use in the "cont" phase. The harddisk is already partitioned at that stage and the question will have no effect. `<stage>cont</stage>`	
selection	the selection element contains a list of <entry> elements. Each entry represents a possible option for the user to choose. So the user can't enter a value in a textfield, but he can choose from a list of values.	optional for type=string, not possible for type=boolean and a must have for type=symbol

```
<selection config:type="list">
  <entry>
    <value>
        reiser
    </value>
    <label>
        Reiser Filesystem
    </label>
  </entry>
  <entry>
    <value>
        ext3
    </value>
    <label>
        Extended3 Filesystem
    </label>
  </entry>
</selection>
```

Below you can see an example of the usage of the "ask" feature.

```
<general>
  <ask-list config:type="list">
    <ask>
        <path>ldap,ldap_server</path>
        <stage>cont</stage>
        <help>choose your server depending on your department</help>
        <selection config:type="list">
            <entry>
                <value>ldap1.mydom.de</value>
                <label>LDAP for development</label>
            </entry>
            <entry>
                <value>ldap2.mydom.de</value>
                <label>LDAP for sales</label>
            </entry>
        </selection>
        <default>ldap2.mydom.de</default>
    </ask>
    <ask>
        <path>networking,dns,hostname</path>
        <question>Enter Hostname</question>
        <stage>initial</stage>
```

```
                <default>enter your hostname here</default>
        </ask>
        <ask>
            <path>partitioning,0,partitions,0,filesystem</path>
            <question>Filesystem</question>
            <type>symbol</type>
            <selection config:type="list">
                <entry>
                        <value config:type="symbol">reiser</value>
                        <label>default Filesystem (recommended)</label>
                </entry>
                <entry>
                        <value config:type="symbol">ext3</value>
                        <label>Fallback Filesystem</label>
                </entry>
            </selection>

        </ask>
  </ask-list>
  ...
</general>
```

Chapter 5. Network Based Installation

The installation method using AutoYaST provides a way to automatically and identically install groups of systems. The first step when preparing AutoYaST installations is deciding how you want the systems at your site to be installed. For example, the following scenario would be ideal to set up and perform automated installations:

- You need to install SuSE Linux on 50 new systems.

- The development department owns 30 out of the 50 new dual processor and SCSI systems, and its systems must be installed as clients with development software.

- The sales department owns 20 out of the 50 new, uni-processor IDE based systems and its systems must be installed as clients with end user software and office tools.

Prerequisites:

- A boot server on the same Ethernet segment

- An install server with the SuSE Linux OS

- An AutoYaST configuration server that defines rules and profiles.

5.1. Configuration Server

A configuration repository holds the control files for multiple machines. The control files can have any file names, which have to specified at the boot time of a client. To avoid supplying the profile name for every client, you can only define the directory of the control files. If a directory is specified, then the client tries to load a file with a name matching it's IP address in HEX mode. This has the advantage that you will be dealing with consistent file names rather than IPs as file names which might lead to some confusion.

The configuration repository is the same directory you have to define if you are using the configuration system for creating control files.

5.1.1. HTTP Repository

To be able to use the HTTP protocol to retrieve control file while auto-installing, you need a working HTTP server on the server side. Install *Apache* or your favorite web server and enable it using YaST2. Normally the the web server root directory resides in `/srv/www/htdocs` so you need to create a subdirectory below the root directory of the web server which will be your configuration repository.

5.1.2. NFS Repository

Create a directory and make it available via NFS to the clients by exporting it. This directory may for example be in the same place where you have copied the CDs. (i.e. `/usr/local/SuSE`)

5.1.3. TFTP Repository

By default the TFTP directory is available under `/tftpboot` which can also contain boot images if you are booting over network. Do not forget to enable TFTP in the Inetd configuration file (`/etc/inetd.conf`). *Inetd* configuration can be done using YaST2.

Chapter 6. Rules and Classes

6.1. Rule based auto-installation

Rules offer the possibility to configure a system depending on system attributes by merging multiple control file during installation. The rules based installation is controlled by a rules file.

The rules file is an XML based file that contains rules for each group of systems (or single systems) that you want to automatically install. A set of rules distinguish a group of systems based on one or more system attributes, after passing all rules, it links each group of rules to a profile. Both the rules file and the profiles must be located in a pre-defined and accessible location.

The rules file is retrieved only if no specific control is supplied using the *autoyast* keyword. For example, if the following is used, the rules file wont be evaluated:

```
autoyast=http://10.10.0.1/profile/test.xml
```

Figure 6.1. Rules

If more than one rule apply, the final profile for each group is generated on the fly using a merge script. The merging process is based on the order of the rules and later rules override configuration data in earlier rules.

The use of a rules file is optional. If the rules file is not found, system installation proceeds in the classic way by just using the supplied profile or by searching for the profile depending on the *MAC* or the *IP* address of the system.

6.1.1. Rules File explained

Example 6.1. Simple rules file

The following simple example illustrates how the rules file is used to retrieve the configuration for a client with known hardware.

```xml
<?xml version="1.0"?>
<!DOCTYPE autoinstall SYSTEM "/usr/share/autoinstall/dtd/rules.dtd">
<autoinstall xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
  <rules config:type="list">
    <rule>
        <disksize>
            <match>/dev/hdc 1000</match>
            <match_type>greater</match_type>
        </disksize>
        <result>
            <profile>machine1.xml</profile>
            <continue config:type="boolean">false</continue>
        </result>
    </rule>
    <rule>
        <disksize>
            <match>/dev/hda 1000</match>
            <match_type>greater</match_type>
        </disksize>
        <result>
            <profile>machine2.xml</profile>
            <continue config:type="boolean">false</continue>
        </result>
    </rule>
  </rules>
</autoinstall>
```

The last example defines 2 rules and provides a different profile for every rule. The rule used in this case is *disksize*. After parsing the rules file, YaST2 attempts to match the system being installed to the rules in the `rules.xml` file in the following order: first rule through the last rule. A rule match occurs when the system being installed matches all of the system attributes defined in the rule. As soon as a system matches a rule, the result resource is added to the stack of profiles AutoYaST will be using to create the final profile. The *continue* property tells AutoYaST if it should continue with other rules or not after a match has been found.

If the first rule does not match, next rule in the list is examined until a match is found.

Using the *disksize* attribute, you can provide different configurations for different hard drives with different size. First rule checks if the device */dev/hdc* is available and if it is greater than 1 GB in size using the *match* property.

A rule must have at least one attribute to be matched. If you need to check more attributes, i.e. memory or architectures, you can add more attributes in the rule resource as shown in the next example.

Example 6.2. Simple rules file

The following simple example illustrates how the rules file is used to retrieve the configuration for a client with known hardware.

```
<?xml version="1.0"?>
<!DOCTYPE autoinstall SYSTEM "/usr/share/autoinstall/dtd/rules.dtd">
<autoinstall xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
  <rules config:type="list">
    <rule>
      <disksize>
          <match>/dev/hdc 1000</match>
          <match_type>greater</match_type>
      </disksize>
      <memsize>
          <match>1000</match>
          <match_type>greater</match_type>
      </memsize>
      <result>
          <profile>machine1.xml</profile>
          <continue config:type="boolean">false</continue>
       </result>
    </rule>
    <rule>
      <disksize>
          <match>/dev/hda 1000</match>
          <match_type>greater</match_type>
      </disksize>
      <memsize>
          <match>256</match>
          <match_type>greater</match_type>
      </memsize>
      <result>
          <profile>machine2.xml</profile>
          <continue config:type="boolean">false</continue>
       </result>
    </rule>
  </rules>
</autoinstall>
```

The rules directory must be located in the same referenced directory used with the *autoyast* keyword on boot time, so if the client was booted using *autoyast=http://10.10.0.1/profiles/*, AutoYaST will search for the rules file in *http://10.10.0.1/profiles/rules/rules.xml*.

6.1.2. Custom Rules

If the attributes autoyast provides for rules are not enough for you, you can use custom rules. Custom rules are more or less a shell script you have to write has and whose output on STDOUT is being used to know which autoyast profile should be fetched. STDERR will be ignored.

Here is an example for the use of a custom rules:

```
<rule>
    <custom1>
        <script>
if grep -i intel /proc/cpuinfo > /dev/null; then
echo -n "intel"
else
echo -n "non_intel"
fi;
        </script>
        <match>*</match>
        <match_type>exact</match_type>
    </custom1>
    <result>
        <profile>@custom1@.xml</profile>
        <continue config:type="boolean">true</continue>
    </result>
</rule>
```

The script in this rule can echo either "intel" or "non_intel" to STDOUT (the output of the grep command must be directed to /dev/null in this case). Autoyast will catch that output and will fetch a file with the name "intel.xml" or "non_intel.xml". This file can contain the autoyast profile part for the software selection for example, in the case you want to do a different software selection on intel hardware than on others. So the output of the rule script will be filled between the two '@' characters, to determine the filename of the profile to fetch.

The number of custom rules is limited to 5. So you can use custom1 to custom5.

6.1.3. Match Types for rules

you can have five different match_types:

- exact - which is the default

- greater

- lower

- range

- regex (available since 10.1 and SLES10)

"greater" and "lower" can be used for memsize or totaldisk for example. They can match only on rules which return an integer value. A range is only possible for integer values too and has the form of "value1-value2", for example "512-1024". regex can be used to match substrings like "ntel" will match "Intel","intel" and "intelligent".

6.1.4. Combine Attributes

It's possible to combine multiple attributes via a logical operator. So it's possible to let a rule match if disksize is greater than 1GB or memsize is exact 512MB (well, not the best example maybe).

You can do that with the "operator" element in the rules.xml file. Here is an example:

```
<rule>
    <disksize>
        <match>/dev/hda 1000</match>
        <match_type>greater</match_type>
    </disksize>
    <memsize>
        <match>256</match>
        <match_type>greater</match_type>
    </memsize>
    <result>
        <profile>machine2.xml</profile>
```

```
        <continue config:type="boolean">false</continue>
    </result>
    <operator>or</operator>
</rule>
```

Just "and" and "or" are possible operators and the default operator is "and".

6.1.5. Rules file structure

The `rules.xml` file must have:

- At least one rule

- It must have the name `rules.xml`

- It must be located in the directory *rules* in the profile repository

- At least one attribute to match in the rule

6.1.6. Predefined System Attributes

The following table lists the predefined system attributes you can match in the rules file.

If you are unsure about a value on your system, start an autoinstallation. If the proposal shows up, switch to the console via CTRL+ALT+F2 and run

```
y2base ayast_probe ncurses
```

. It might help to to turn the confirmation on for this, so that the installation does not start in the background while you are watching the values. The textbox with the values is scrollable.

Table 6.1. System Attributes

Attribute	Values	Description
hostaddress	IP address of the host	This attribute must always match exactly
hostname	The name of the host	This attribute must always match exactly
domain	Domain name of host	This attribute must always match exactly
installed_product	The name of the product that is getting installed. For example "SUSE LINUX"	This attribute must always match exactly
installed_product_version	The version of the product that is getting installed. For example "9.3"	This attribute must always match exactly
network	network address of host	This attribute must always match exactly
mac	MAC address of host	This attribute must always match exactly. (MAC addresses to be matched should be in the form *0080C8F6484C*
linux	Number of installed Linux partitions on the system	This attribute can be 0 or more
others	Number of installed non-Linux partitions on the system	This attribute can be 0 or more

Attribute	Values	Description
xserver	X Server needed for graphic adapter	This attribute must always match exactly
memsize	Memory available on host in MByes	All match types are available
totaldisk	Total disk space available on host in MBytes	All match types are available
haspcmica	System has PCMCIA (i.e Laptops)	Exact match required, 1 for available PCMCIA or 0 for none
hostid	Hex representation of IP address	Exact match required
arch	Architecture of host	Exact match required
karch	Kernel Architecture of host (i.e. SMP kernel, Athlon Kernel)	Exact match required
disksize	Drive device and size	All match types are available
product	The hardware product name as specified in SMBIOS	Exact match required
product_vendor	The hardware vendor as specified in SMBIOS	Exact match required
board	The system board name as specified in SMBIOS	Exact match required
board_vendor	The system board vendor as specified in SMBIOS	Exact match required
custom1-5	Custom rules using shell scripts	All match types are available

6.2. Classes

You can assign a system to different classes which can be defined in the control file. Unlike rules, classes have to be configured in the control file and represent a configuration which is typical for a group of systems.

Here is an example of a class definition:

```
<classes config:type="list">
    <class>
        <class_name>TrainingRoom</class_name>
        <configuration>Software.xml</configuration>
    </class>
</classes>
```

The file Software.xml must be in the directory "classes/TrainingRoom/" then and it will get fetched from the same place like the autoyast profile and the rules were fetched from.

If you have multiple kind of profiles and those profiles share parts, it's recommended to use classes for that. You just have to change the class and all profiles using that class are fixed too. By the way, you can reach something similar by using XIncludes.

Using the configuration management system, you can define a set of classes. The class definition consists of the following variable for each class:

• Name: Class name

• Descriptions: Class description

• Order: Order (or priority) of the class in the stack of migration

Figure 6.2. Defining Classes

You can create as many classes as you need, however it is recommended to keep the set of classes as small as possible to keep the configuration system concise. As an example, the following set of classes can be used:

• site: Classes describing a physical location or site.

• machine: Classes describing a type of machine or make

• role: Classes describing the function of the machine to be installed

• group: Classes describing a department or a group within a site or a location.

A file saved in a class directory can have the same syntax and format as a regular control file but represents a subset of the configuration. For example, to create a new control file for a special computer with a specific network interface, only the resource in the control file, which controls the configuration of the network is needed. Having multiple network types, you can merge the one needed for a special type of hardware with other class files and create a new control file which suits the system being installed.

6.3. Mixing Rules and Classes

It is possible to mix rules and classes during an auto-installation session. For example you can identify a system using rules which contain class definitions in them. The process is described in the figures "Rules Retrieval Process".

After retrieving the rules and merging them, the generated control file is parsed and the presence of class definitions is checked. If classes are defined, then the class files are retrieved from the original repository and a new merge process is initiated.

6.4. The merging process of Rules and Classes

With classes and with rules, multiple XML files get merged to one resulting XML file. This process of merging is often confusing for people, because it behaves different than one would expect.

Let's take a look at two XML parts that we want to merge:

```xml
<partitioning config:type="list">
    <drive>
        <partitions config:type="list">
            <partition>
                <filesystem config:type="symbol">swap</filesystem>
                <format config:type="boolean">true</format>
                <mount>swap</mount>
                <partition_id config:type="integer">130</partition_id>
                <size>2000mb</size>
            </partition>
            <partition>
                <filesystem config:type="symbol">xfs</filesystem>
                <partition_type>primary</partition_type>
                <size>4Gb</size>
                <mount>/data</mount>
            </partition>
        </partitions>
    </drive>
</partitioning>
```

```
<partitioning config:type="list">
  <drive>
    <initialize config:type="boolean">false</initialize>
    <partitions config:type="list">
        <partition>
          <format config:type="boolean">true</format>
          <filesystem config:type="symbol">xfs</filesystem>
          <mount>/</mount>
          <partition_id config:type="integer">131</partition_id>
          <partition_type>primary</partition_type>
          <size>max</size>
        </partition>
    </partitions>
    <use>all</use>
  </drive>
</partitioning>
```

What you would expect is, that you'll end up in a profile with 3 partitions. That is not the case. You'll end up with two partitions and the first partition is a mixup of the swap and the root partition. Stuff that is configured in both partitions, like *mount* or *size*, will be used from the second file. Stuff that only exists in the first or second partition, will be copied to the merged partition too.

Sometimes this is what you want, but sometimes this is not what you want. Actually, in my example above, this is both. You don't want a second *drive* right? You want the two drives to be merged into one but for partitions you want three seperate ones. So how can you achieve what you want? In this example, how do you get three partitions but still one drive?

> **Note**
>
> For SLES9/SUSE Linux 10.0 and earlier versions

you can only use a trick. It's not official supported by autoyast and more a workaround than a nice solution. For each partition in one file, add an attribute to the partition like this:

```
<partition dontmerge="1">
...
</partitions>
```

The trick is, that the merge script will not detect the partitions as the same element type because of the new attribute. If you have more files, it might be needed to to add more attributes like *dontmerge="2"*.

> **Note**
>
> For SLES10/SUSE Linux 10.1 and later

you can use the *dont_merge* element in the rules or classes file like this:

```
<classes config:type="list">
    <class>
        <class_name>swap</class_name>
        <configuration>largeswap.xml</configuration>
        <dont_merge config:type="list">
            <element>partition</element>
        </dont_merge>
    </class>
</classes>
```

```
<rule>
    <board_vendor>
        <match>ntel</match>
        <match_type>regex</match_type>
    </board_vendor>
    <result>
        <profile>classes/largeswap.xml</profile>
        <continue config:type="boolean">true</continue>
        <dont_merge config:type="list">
          <element>partition</element>
        </dont_merge>
```

```
    </result>
</rule>
```

Chapter 7. The Auto-Installation Process

7.1. Introduction

After the system has booted and the control file has been retrieved, YaST2 performs configuration of the system according to the information provided in the control file. All the configuration is summarized in a window that is shown by default and should be deactivated if a full automatic installation is needed.

When YaST2 has reached the point where the summary of the configuration is shown, YaST2 has only probed hardware and prepared the system for auto-installation, thus, nothing has been changed in the system yet, so that in case of any error, the process still can be aborted.

A system should be automatically installable without the need to have any graphic adaptor or monitor. Having a monitor attached to the client machine is nevertheless recommended to follow the process and to get feedback in case of any errors. Choosing between the Qt and the Ncurses interfaces is possible. For headless clients, system messages can be monitored using the serial console.

7.1.1. X11 Interface

This is the default interface while auto-installing. No special variables are required to activate it.

7.1.2. Serial console

You can start installing a system using the serial console by adding the keyword console (i.e. console=ttyS0) to the command line of the kernel. This will start linuxrc in console mode and later in the process, YaST2 also is started in serial console mode.

7.1.3. Text based YaST2-Installation

This option can also be activated on the command line. This will start YaST2 in *Ncurses* mode. To start YaST2 in text mode, add *textmode=1* on the command line.

Starting YaST2 in text mode is recommended when installing a client with less than 64 MB or when X11 is not being configured at all, especially on headless machines.

7.2. Choosing the right Boot Medium

There are different methods for booting the client. The computer can boot from its network interface card (NIC) to receive the boot images via DHCP /TFTP or a suitable kernel as well as an initrd image are loaded from a floppy or a boot-able CD-ROM.

7.2.1. Booting from a floppy

For testing/rescue purposes or because the NIC does not have a PROM or PXE you can build a boot floppy to use with AutoYaST. Using a floppy to initiate an auto-install process is limited due to the size of the data a floppy can hold. However, it is still possible to use floppies when auto-installing a single, disconnected machine.

Floppies can be used to store the control file, especially when using the original *SuSE* CD-ROMs for a single, disconnected machine. Using the kernel command line, you can specify the location of the control file on the floppy.

Even without specifying any command line options, it is still possible to initiate the auto-install process by placing a control file on a floppy with a special, pre-defined file name. (`autoinst.xml`)

YaST2 will check for `autoinst.xml` upon startup and if it was found it will switch from interactive to automated installation.

7.2.2. Booting from CD-ROM

You can use the original *SuSE* CD-ROMs in combination with other media, i.e. with a floppy to hold the control file or in combination with network where the control file can be located.

It is also possible to create customized CD-ROMs to hold only the package you need in addition to the control file which also can be saved on the CD-ROM. This method requires creation of CD-ROMs every time you wish to change the configuration though.

7.2.3. Booting via PXE over the network

Booting via PXE requires a DHCP and a TFTP server in your network. The computer will boot then without a physical media like a boot floppy or CDROM.

Here is a small example of a "/srv/tftp/pxelinux.cfg/default" file:

```
default SLES9

# install SLES9
label SLES9
  kernel linux_sles9
    append initrd=initrd_sles9 vga=0x0314 install=.... autoyast=... language=de_DE

# boot harddisc
label hd·
  LOCALBOOT 0
```

It's recommended to add the vga=... parameter with a valid value for graphical installations, to trigger an installation with the frame buffer device instead of the vesa driver or ncurses mode.

Here is as a small example my "/etc/dhcp.conf" file:

```
option domain-name-servers 192.168.66.1;
default-lease-time 600;
max-lease-time 7200;
ddns-update-style none; ddns-updates off;
log-facility local7;
option grub-menufile code 150 = text;
option grub-menufile "(nd)/menu.lst";·
subnet 192.168.66.0 netmask 255.255.255.0 {
        range 192.168.66.100 192.168.66.200;
        # PXE related stuff ...
        #
        # "next-server" defines the tftp server which will·
        # serve the pxelinux image to the PXE clients.
        next-server 192.168.66.1;
        allow booting;
        allow bootp;
        option routers 192.168.66.1; # default gateway

        #
        # "filename" specifies the pxelinux image on the tftp server·
        # which will be served to the PXE clients.
        # The configured tftp server on 192.168.100.1 runs in a·
        # "change-root jail" to /srv/tftpboot
        filename "pxelinux.0";
}
```

A problem you might run into if you do installation via PXE is, that the installation will run into an endless loop, because after the first reboot, the machine is doing PXE boot again and will restart the installation instead of booting from harddisc for the second stage of the installation.

This problem can be solved in different ways. One way is to use a http server to provide the autoyast profile and instead of a static profile, a CGI script on the webserver is run that provides the profile and then changes the TFTP server configuration then for this special host, so that the next PXE boot from that machine will be from harddisc by default.

Another way is to use autoyast to upload a new PXE boot configuration for that host. That is done via autoyast profile like this:

```
    <pxe>
      <pxe_localboot config:type="boolean">true</pxe_localboot>
      <pxelinux-config>
DEFAULT linux
  LABEL linux
  localboot 0


      </pxelinux-config>
      <tftp-server>192.168.66.1</tftp-server>
      <pxelinux-dir>/pxelinux.cfg</pxelinux-dir>
    </pxe>
```

This will upload a new configuration for the actual machine to the tftp server short before the first reboot happens. In most installations the TFTP daemon runs as user "nobody". You have to make sure that that user has write permissions to the "pxelinux.cfg" directory if you use that mechanism. So if your machine got the IP address "192.168.66.195" a file "C0A842C3" will be uploaded and if the machine reboots and will get the same IP address via DHCP again, the new configuration will be used that has the harddisc as a default boot media.

Of course this requires that the machine will get the same IP address again after the reboot and if you want to do another autoinstallation for that machine, you have to remove the file from the TFTP server.

7.3. Invoking the Auto-Installation process

7.3.1. Command line Options

Adding the command line variable *autoyast* will make *linuxrc* start in automated mode. **Linuxrc** searches for a configuration file, which should be distinguished from the main control file in the following places:

- In the root directory of the initial ram-disk used for booting the system up

- In the root directory of the floppy

The configuration file used by **linuxrc** can have the following keywords (for a detailed description of how linuxrc works and other keywords, see "Advanced Linuxrc Options"):

Table 7.1. Keywords for linuxrc

Keyword	Value
netdevice	Which network device to use for network setup (Device used for BOOTP / DHCP requests)
server	Which server to contact for source directory (NFS Server)
serverdir	Directory on NFS Server
hostip	When empty, client sends BOOTP request, otherwise client is configured with entered IP configuration.
netmask	Netmask
gateway	Gateway
nameserver	Nameserver
insmod	Kernel modules to load.
autoyast	Location of the the control file to be used for the automatic installation, i.e *autoyast=http://192.168.2.1/profiles/*

Keyword	Value
install	Location of the installation directory, i.e. *install=nfs://192.168.2.1/CDs/*
instmode	Installation mode, i.e. nfs, http etc. (Not needed if *install* is set)
y2confirm	even with <confirm>no</confirm> in the profile, the confirm proposal comes up. This is available since SUSE Linux 10.1 / SLES10

These variables and keywords will bring the system up to the point where YaST2 can take over with the main control file. Currently, the source medium is automatically discovered, which in some cases makes it possible to initiate the auto-install process without giving any instructions to linuxrc.

The traditional **linuxrc** configuration file (info) has the function of giving the client enough information about the installation server and the location of the sources. In most cases this file is not needed; it is however needed in special network environments where DHCP / BOOTP are not used or when special kernel modules have to be loaded.

All linuxrc keywords can be passed to **linuxrc** using the kernel command line. The command line can for example also be set when creating network boot-able images or it can be passed to the kernel using a specially configured DHCP server in combination with Etherboot or PXE.

The format of the special command line variable *autoyast* can be used as described in table "Command line variables for AutoYaST"

Table 7.2. Command line variables for AutoYaST

Command line variable	Description
autoyast=default	Default auto-installation option
autoyast=file://<path>	Looks for control file in specified path (relative to source root directory, i.e. *file:///autoinst.xml* if in the top directory of a CD-ROM)
autoyast=device://<device>/<file>	Looks for control file on a storage device. (only device name needed without full path, i.e. /dev/sda1 is wrong, instead use sda1)
autoyast=floppy://<path>	Looks for control file in the floppy (Useful when booting from CD). Since SLES10 SP1 and later the fallback is looking on USB devices too
autoyast=nfs://<server>/<path>	Looks for control file on <server>
autoyast=http://<server>/<path>	Retrieves the control file from a web server using the HTTP protocol.
autoyast=https://<server>/<path>	Retrieves the control file from a web server using HTTPS (encrypted connection) protocol (possible since SL 10.1 and SLES10
autoyast=tftp://<server>/<path>	Retrieve the control file with TFTP
autoyast=ftp://<server>/<path>	Retrieve the control file with FTP
autoyast=usb://<path> (since SLES10 SP1)	Retrieve the control file from USB devices (autoyast will search on all USB devices it can find)

Several scenarios for auto-installation are possible using different types of infrastructure and source media. The simplest way is by using the source media from the *SuSE* Box. In that case you have either a DVD with all *SuSE* packages or a set of CD-ROMs. To initiate the auto-installation process however, the auto-installation command line variable should be entered at system boot-up and the control file should be accessible to YaST2. The following list of scenarios explains how the control

file can be supplied and the setup needed for the auto-installation process to be successful.

- Using *SuSE* original CD-ROMs from *SuSE* Linux box:

  To use the original CD-ROMs, you need a media with the control file, the control file can reside on the following locations:

  1. *Floppy*: Control file accessible via the *autoyast=floppy* option. YaST2 also searches upon startup for a file named `autoinst.xml`. If such a file is found, YaST2 will switch into auto-installation mode even if no special command line variables were supplied. (See " Auto-installing a Single System ")

  2. *Network*: Control file accessible via the *autoyast=nfs://..*, *autoyast=ftp://.. autoyast=http://..* or *autoyast=tftp://..* options.

- Using 'self-made' CD-ROMs:

  In this case, you can include the control file on the CD-ROM for easy access (using the *autoyast=file://* option) or use one of the above mentioned methods used with the original *SuSE* CD-ROMs.

  Using CD-ROMs for autoinstallation, it is required to instruct the installer to use the CD-ROM for installation and not try to find the installation files on the network. This can be accomplished by adding the *instmode=cd* option to the kernel command line (this can be done by adding the option to the isolinux.cfg file on the CD).

- Using NFS and Floppy, Network or CD-ROM for system boot-up.

  This option is the most important one due to the fact that installations of PC farms are normally done using NFS servers and other network services like BOOTP / DHCP . The control file can reside in the following places:

  1. *Floppy/CD-ROM*: Control file accessible via the *autoyast=file://..* option.

  2. *Network*: Control file accessible via the *autoyast=http://..*, *autoyast=ftp://..*, *autoyast=nfs://..* or *autoyast=tftp://..* options.

Disabling netowrk and DHCP

To disable network during installations where network is not needed or not available, for example when auto-installing from CD-ROMs use the linuxrc option *netsetup* to set network configuration behavior. To disable network setup use *netsetup=0*

If *autoyast=default* is defined, YaST2 will look for a file named `autoinst.xml` in the following three places:

1. The root directory of the floppy disk.

2. The root directory of the installation medium.

3. The root directory of the initial ram disk used to boot the system.

With all autoyast invocation options, excluding *default*, it is possible to specify the location of the control file in the following ways:

1. Specify the exact location of the control file:

```
autoyast=http://192.168.1.1/control-files/client01.xml
```

2. Specify a directory where several control files are located

```
autoyast=http://192.168.1.1/control-files/
```

In this case the relevant control file is retrieved using the hex digit representation of the IP as described below.

If only the path prefix variable is defined, YaST2 will fetch the control file from the specified location in the following way:

1. First, it will search for the control file using its own IP address in upper case hexadecimal, e.g. *192.0.2.91 -> C000025B*.

2. If that file is not found, it will remove one hex digit and try again. This action is repeated till the file with the correct name is found. Ultimately, it will try looking for a file with the MAC address of the clients as the file name (mac should have the following syntax: *0080C8F6484C*) and if not found a file named `default` (in lower case).

As an example, for 192.0.2.91, the HTTP client will try:

```
C000025B
C000025
C00002
C0000
C000
C00
C0
C
0080C8F6484C
default
```

in that order.

To determine the hex representation of the IP address of the client, use the utility called **/usr/sbin/gethostip** available with the *syslinux* package.

Example 7.1. Determine HEX code for an IP address

```
# /usr/sbin/gethostip 10.10.0.1
10.10.0.1 10.10.0.1 0A0A0001
```

7.3.2. Auto-installing a Single System

The easiest way to auto-install a system without any network connection is by using the standard CD-ROMs that come in the *SuSE* Linux box. Using the CD-ROMs in combination with a floppy disk lets you getting started with AutoYaST very fast and without spending much time configuring installation server and network environments.

Create the control file and name it `autoinst.xml`. Copy the file `autoinst.xml` to a floppy by either mounting the floppy or by using the *mtools*.

```
mcopy autoinst.xml a:
```

7.3.3. Combining linuxrc *info* file with YaST2 control file

If you choose to pass information to *linuxrc* using the *info* file, it is possible to integrate the keywords in the XML control file. In the case the file has to be accessible to linuxrc and has to be named *info*.

Linuxrc will look for a string (*start_linuxrc_conf* in the control file which represents the beginning of the file. If it is found, it will parse the content starting from that string and will finish when the string *end_linuxrc_conf* is found. The options are stored in the control file in the following way:

Example 7.2. Linxurc options in the control file

```
....
  <install>
....
    <init>
      <info_file>
<![CDATA[
#
# Don't remove the following line:
# start_linuxrc_conf
#
install: nfs://192.168.1.1/CDs/full-i386
textmode: 1
autoyast: file:///info

# end_linuxrc_conf
# Do not remove the above comment
#
]]>

      </info_file>
    </init>
......
  </install>
....
```

Note that the autoyast keyword must point to the same file, i.e. if it is on a floppy, then the protocol *floppy* has to be used. In other cases where the *info* file is stored in the initial ram-disk, the *file* option has to be used.

7.4. System Configuration

The system configuration during auto-installation can be seen as the most important part of the whole process. Customizing a system to your environment needs is what makes an auto-installation system attractive, not the installation part.

As you have seen in the previous chapters, almost anything can be configured automatically on the target system. In addition to the pre-defined directives, you can always use post-scripts to change other things in the system. Additionally you can change any system variables and if required, copy complete configuration files into the target system.

7.4.1. Post-Install and System Configuration

The Post-Installation and the System Configuration are initiated directly after the last package is installed in the target system and is continued after the system has booted for the first time.

Before the system is booted for the first time, YaST2 writes all data collected during installation into the system and finally it writes the boot loader in the specified location. In addition to these regular tasks, which are also done when performing a regular installation, YaST2 executes the *chroot-scripts* as specified in the control file. Note that these scripts are executed while the system is still not mounted.

If a different kernel than the default is installed, a hard reboot will be required. A hard reboot can also be forced during auto-installation, independent of the installed kernel. This can be accomplished using the *reboot* property of the *general* resource. (See General Options)

7.4.2. System Customization

Most of the system customization is done in the second stage of the installation. YaST2 provides most of the important resources needed to bring up a system to a usable , general state. However, you may have other requirements for the installed system. If the required customizations can't be done using YaST2 resources, then the post-install scripts can be used to accomplish this task.

You can define an unlimited number of custom scripts in the control file either by editing the control file or by using the configuration system.

Chapter 8. Legacy and foreign Configuration formats

8.1. Migration from YaST1 and ALICE

ALICE, SuSEs former auto-installation system was a system built around the auto-installation features that were available with YaST1. In order to be able to use existing ALICE configuration files and resources, a special option is provided in the configuration system will let you convert ALICE configuration files into a control file readable by AutoYaST.

ALICE uses YaST1 for the installation of a Linux System. This is done by creating a boot medium with the needed control file (info) from a set of configuration files maintained in a CVS repository. The *info* file has the traditional format used with YaST1.

The system configuration is done almost entirely after the system is installed and is initiated using the *%post* section of the ALICE RPM package.

8.1.1. ALICE modules

Each of ALICE modules perform certain tasks and requires one or multiple configuration files. The modules are shell scripts that are invoked after installation of the machine to setup different services on the client.

YaST2 offers an extensive and rich interface to the installed system which replaces most of the modules that were available with ALICE The following is a list of all ALICE modules, their function and how their functionality is provided by YaST2 modules which are already available or which can be integrated easily:

The modules *prepare_alice* and *make_all* provided the base and main scripts of all ALICE modules. The module *prepare_alice* is called right after the initial installation process - which is done after YaST1 has finished installation packages.

With YaST2, the alternative to these modules is part of the system and is not an addition or extension. This means that YaST2 configures the system in one single process and not, as with ALICE in two different, independent steps.

The following table show how the result of some ALICE modules can be accomplished with YaST2

Table 8.1. ALICE vs. YaST2 modules

Module	ALICE	YaST2	Concerned files
User configuration	Adding users is performed without any consistency checks.	Using YaST2 users module	/etc/passwd
Group configuration	Adding groups is performed without any consistency checks.	Using YaST2 users module	/etc/group
Services	X	-	/etc/services
Inetd	X	Using the inetd module	/etc/inetd.conf
Syslog	X		/etc/syslog.conf
Lilo	X	X	/etc/lilo.conf
SNMP	X	-	/etc/ucdsnmpd.conf

Module	ALICE	YaST2	Concerned files
Cron	X	-	`/etc/crontab`
hosts	X	X	`/etc/hosts`
Routing	X	X	`/etc/route.conf`
Printer	X	X	`/etc/printcap`
SSH	X	-	`/etc/ssh`
Kernel modules	X	X	`/etc/modules.conf`

8.1.2. Other configuration options with YaST2 and ALICE

ALICE completed most of the configuration using the system configuration file `/etc/rc.config`. This interface to the system does not exist anymore with new SuSE products. Instead, configuration options are now available in `/etc/sysconfig`.

Entries in `/etc/sysconfig` can be easily set and modified using YaST2 and AutoYaST. For network services it is recommended to use the relevant YaST2 modules dealing with networking , rather than modifying `/etc/sysconfig/network` directly.

8.2. Redhat Kickstart

The Configuration Management System offers an option for importing a foreign auto-installation configuration file using the *Kickstart* system. To import a *Kickstart* file, you only need to enter the path of the configuration file and decide how to handle the data, by either saving the result into a file directly or by loading the resulting AutoYaST compatible control file into the Configuration Management System to add more configuration options available with SuSE. The following sections describe some limitations and issues to consider when importing from a foreign source:

8.2.1. Software selections and packages

It is not possible at the moment to import package selections and single package listing into AutoYaST. The packages will be added to the control file, but manual selection of the desired packages is needed. The reason is that package groups and package names might differ depending on the source distribution

8.2.2. User scripts

Kickstart offers two types of scripts, *pre* and *post* scripts. More scripts (>2) are not supported as in AutoYaST. So it might be needed to split large scripts into smaller components depending on the function to make such scripts manageable

Appendix A. Handling Rules

The following figure illustrates how rules are handled and the processes of retrieval and merge.

Figure A.1. Rules Retrieval Process

Appendix B. Advanced Linuxrc Options

Linuxrc is a program used for setting up the kernel for installation purposes. It allows the user to load modules, start an installed system, a rescue system or an installation via YaST.

Linuxrc is designed to be as small as possible. Therefore, all needed programs are linked directly into one binary. So there is no need for shared libraries in the initdisk.

Note

If you run Linuxrc on an installed system, it will work slightly different as it tries not to destroy your installation. As a consequence you cannot test all features this way.

B.1. Passing parameters to Linuxrc

Unless Linuxrc is in manual mode it will look for an 'info' file in these locations: first `/info` on the floppy disk and if that does not exsist for `/info` in the initrd. After that it parses the kernel command line for parameters. You may change the 'info' file Linuxrc reads using the `info` command line parameter. If you don't want Linuxrc to read the kernel command line (say, because you need to give a kernel parameter that accidentally is recognized by Linuxrc, too), use *linuxrc=nocmdline*.

Independend if the above, Linuxrc will always look for and parse a file `/linuxrc.config`. You can use this file to change default values, if you need to. But in general, use the `info` file instead. Note that `/linuxrc.config` is read before any 'info' file and even in manual mode.

B.2. 'info' file format

Lines starting with '#' are comments, valid entries are of the form

```
key: value
```

Note that *value* extends to the end of the line and so may contain spaces. *key* is matched case insensitive.

You can use the same key - value pairs on the kernel command line using the syntax key=value Lines that don't have the form described above are ignored.

Valid keys are (values given are just examples)

Table B.1. Advanced linuxrc keywords

Keyword/Value	Description
Language: de_DE	set the language
Keytable: de-lat1-nd	load this keytable
Display: Color|Mono|Alt	set the menu color scheme
Install: nfs://server/install/8.0-i386	install via NFS from *server* (note: you can give username, password etc. in the URL, too)
InstMode: cd|hd|nfs|smb|ftp|http|tftp	set installation mode
HostIP: 10.10.0.2	the client ip address
Netmask: 255.255.0.0	network mask
Gateway: 10.10.0.1	gateway

Keyword/Value	Description
Server: 10.10.0.1	installation server address
Nameserver: 10.10.0.1	nameserver
Proxy: 10.10.0.1	proxy (either ftp or http)
ProxyPort: 10.10.0.1	proxy port
Partition: hda1	partition with install sources for hard disk install
Serverdir: /install/8.0-i386	base directory of the installation sources
Netdevice: eth0	network interface to use
BOOTPWait: 5	sleep 5 seconds between network activation and starting bootp
BOOTPTimeout: 10	10 seconds timeout for BOOTP requests
DHCPTimeout: 60	60 seconds timeout for DHCP requests
TFTPTimeout: 10	10 seconds timeout for TFTP connection
ForceRootimage: 0|1	load the installation system into ramdisk
Textmode: 0|1	start YaST in text mode
Username: name	set user name (e.g. for FTP install)
Password: password	set password (e.g. for FTP install)
WorkDomain: domain	set work domain for SMB install
ForceInsmod: 0|1	use '-*f*' option when running *insmod*
DHCP: 0|1	start DHCP daemon *now*, but see *UseDHCP*
UseDHCP: 0|1	use DHCP instead of BOOTP (DHCP is default)
MemLimit: 10000	ask for swap if free memory drops below 10000 kB
MemYaST: 20000	run YaST in text mode if free memory is below 20000 kB
MemYaSTText: 10000	ask for swap before starting YaST if free memory is below 10000 kB
MemModules: 20000	delete all modules before starting YaST if free memory is below 20000 kB
MemLoadImage: 50000	load installation system into ramdisk if free memory is above 50000 kB
Manual: 0|1	start Linuxrc in manual mode
NoPCMCIA: 0|1	don't start card manager
Domain: zap.de	set domain name (used for name server lookups)
RootImage: /suse/images/root	installation system image
RescueImage: /suse/images/rescue	rescue system image
InstallDir: /suse/inst-sys	installation system
Rescue: 1|nfs://server/dir	load rescue system; the URL variant specifies the location of the rescue image explicitly
AutoYaST: ftp://autoyastfile	location of autoinstall file; activates autoinstall mode
VNC: 0|1	setup VNC server
VNCPassword: password	sets VNC server password
UseSSH: 0|1	setup SSH server
SSHPassword: password	sets SSH server password (this will not be the final root password!)
AddSwap: 0|3|/dev/hda5	if 0, do never ask for swap; if the argument is a positive number *n*, activate the *n*'th swap partition; is the argument is a partition name, activate

Keyword/Value	Description
	this swap partition
Exec: command	run *command*
USBWait: 4	wait 4 seconds after loading USB modules
Insmod: module params	load this module
Loghost: 10.10.0.22	Enable remove logging via syslog
y2confirm	overrides the confirm parameter in a profile and pops up the confirm proposal (available since SUSE Linux 10.1 / SLES10)

B.3. Advanced Network Setup

The *netsetup* keyword allows advanced network configurations and enables dialogs to setup the network where required.

- netsetup=1

  the normal network setup questions

- netsetup=xxx,yyy

  just xxx and yyy

- netsetup=+xxx,-yyy

  default, additionally xxx, but not yyy

xxx could have the following values: dhcp, hostip, gateway, netmask, nameserver. nameserverN asks for N nameservers (max. 4).

For example, the following can be entered on the command line:

```
netsetup=-dhcp,+nameserver3
```