oneSIS v2.0.5: Administrator's Manual

Josh England <jjengla@gmail.com>

Last update: oneSIS-2.0.5 (February, 2017)

This document is continually being updated.
Refer to `http://onesis.org/docs.php` for the latest documentation.

Contents

1 Introduction

oneSIS is a system for enabling a single root filesystem (ie: '/' on your Linux machine) to be shared by many (even functionally different) machines. It is comprised of a collection of tools to install, configure, and maintain a cluster of diskless machines. It allows read-only NFS (Network File System) root clusters to be deployed and maintained with ease, and enables variations between nodes while still using a single root filesystem.

The root image (or any subset of it) can be deployed to disk, if desired, so that any cluster node can boot from the local disk. Using oneSIS, it is possible to have nodes that use almost any combination of NFS, RAM-based, and local disk for files/directories in the root image.

This document describes some of the techniques oneSIS employs and details all of the configuration directives and helper applications that make up the system. It is written to serve as reference material for the configuration directives and utilities, not as a detailed usage guide. A useful resource detailing specific configuration examples can be found online at oneSIS HOWTO at `http://onesis.org/oneSIS-HOWTO.php`.

1.1 A primer on diskless booting using NFS root

Although it can handle 'diskful' machines, oneSIS was designed primarily for managing diskless (NFSroot) nodes. Configuring machines to run diskless (to boot and run off the network) is not a new concept. The technique has been available for several years and does not require oneSIS or other similar software. oneSIS enables an administrator to cope easily with the main peculiarity of running diskless: managing a read-only shared root filesystem.

Linux and DHCP enable diskless machines. oneSIS enables desired elements of the root filesystem to 'become' writable and allows functional groups of nodes or individual nodes to use different variations of the root filesystem. For a primer on configuring machines to run diskless, read the NFSroot HOWTO at `http://onesis.org/NFSroot-HOWTO.php`.

1.2 One root filesystem

oneSIS was developed as a system that could essentially be a cluster 'building block' of sorts. It does this by adhering to one simple constraint: the root filesystem of every node should always be **exactly** identical. Every node is an exact mirror of the root image. This property has many desirable benefits in terms of system manageability and scalability.

The root filesystem of every node is bit-for-bit identical whether it resides on a local disk or is mounted via NFS. Functional groups of nodes or individual nodes can be configured to behave differently by using 'linkbacks' to configure themselves at boot-time with symbolic links going to and from a small RAM disk.

For large clusters, diskful mirrors of the image can each NFS export their root filesystem to a reasonable number of 'diskless' nodes so that services such as DHCP, TFTP, and NFS of the

root image can be distributed efficiently to any scale necessary using commodity hardware.

1.3 Centrally configured, centrally maintained

Even with the complex-sounding architecture mentioned above, configuring and maintaining a cluster of any size is remarkably simple. When running NFSroot with a read-only root filesystem, oneSIS is used to configure certain paths to be read-write. These paths can be configured to be either persistent across a reboot or not by directing them to reside on the local disk, on remote mount points, or in RAM.

Managing system services and configuration files to be class-specific or node-specific is the other central role of oneSIS. In addition, oneSIS provides some power wrapper scripts to consolidate different remote power management and console utilities into a single command. These aspects of the cluster's behavior are centrally configured from a single configuration file within each master image.

Typical configurations are usually minimal, but even complex setups are still easy. The behavior of the entire cluster is easily viewable and modifiable with the control being as fine-grained as necessary.

2 Installation

Building a cluster with oneSIS is a straight-forward process. The first step is to get your nodes to boot NFSroot. Once NFS is setup and the infrastructure is in place to send a kernel to every node, any cluster nodes can be booting quickly into a diskless NFSroot environment.

More information on installing and configuring oneSIS can be found in the oneSIS HOWTO at `http://onesis.org/oneSIS-HOWTO.php`.

Any installed linux distribution can be copied and used as the master image for your cluster. Several distributions are currently well-supported. Look in the 'distro-patches' directory in the source tree to see well-supported distributions. See section 4.6 for information on how to port to a new distribution.

Procedure 1 Download and install oneSIS

- **Download**
  The latest source can be found at http://onesis.org

- **Install**
  This will install the oneSIS scripts, perl module, initramfs templates, and documentation into the current distribution.
  # `cd oneSIS-2.0.5`
  # `make install`

2.1 Dependencies and requirements

Several of the utility programs included with oneSIS are dependent on certain (mostly standard) system utilities being present. Also, certain functionality does need to be configured into the kernel for oneSIS to operate.

The following features should be enabled in the kernel for all components of oneSIS to operate effectively

- `tmpfs` (Virtual memory file system support)
- loopback device support (for initrd support – not needed for initramfs)
- `devfs` (`/dev` file system support) **Note**: only needed with `2.4` kernels

oneSIS uses a `tmpfs` ramdisk for many operations. Also, oneSIS requires either `udev` (with 2.6 kernels) or `devfs` (with 2.4 kernels) to handle the `/dev` directory.

A static `/dev` can be made to work (ie: LINKDIR /dev -d) with limited functionality, but it is not recommended. Depending on what applications you are running you may not even need to use `udev` or `devfs` to handle `/dev`, but that is also not recommended.

Note: It may be necessary to download and install `udev` or the `devfsd` program depending on your linux distribution.

oneSIS makes use of the following linux utilities, most of which are present in most linux distributions.

- `patch`
- `cpio`
- `sfdisk`
- `mke2fs/tune2fs/e2label`
- `gzip/gunzip`
- `rsync`
- `grub`
- `lilo`

2.2 Preparing the master image

Before any nodes can be booted, a 'master image' must be created that will be shared across all nodes of the cluster. A filesystem image, normally an installation of some Linux distribution, must be copied into an NFS-exportable directory to serve as the image of the cluster.

Creating the master image from an installed linux machine is a simple process, although it can take some time to copy such a large volume of data. The `copy-rootfs` script (section 6.3) automates the details of copying a root filesystem from the local machine, or from a remote machine with an ssh daemon running.

After it has been prepared, this master image will serve as the root filesystem for all cluster nodes. Copies are used on diskful nodes, and the image itself is NFS mounted read-only on diskless nodes.

Procedure 2 Create a root image

- **Method 1**: Creating an image from the local machine
  This will copy the root filesystem of the local machine into `/var/lib/oneSIS/image`
  `# copy-rootfs -l /var/lib/oneSIS/image`

- **Method 2**: Creating an image from a remote machine
  For a remote machine named **rook**, this will copy the root filesystem of the remote machine into `/var/lib/oneSIS/image`, excluding the remote `/home` directory.
  `# copy-rootfs -r rook -e /home /var/lib/oneSIS/image`

  Note: If it wasn't installed on the remote machine, oneSIS will need to be installed in the image. In the oneSIS source directory, run
  `# cd oneSIS-2.0.5`
  `# prefix=/var/lib/oneSIS/image make install`

  Also note: If the local machine and the image use different versions of Perl, it may be necessary to chroot into the image before installing oneSIS. This will ensure that the oneSIS perl module is installed under the right directory in `/usr/lib/perl`.
  `# cp -a oneSIS-2.0.5 /var/lib/oneSIS/image/usr/local/src`
  `# chroot /var/lib/oneSIS/image`
  `# cd /usr/local/src/oneSIS-2.0.5`
  `# make install`
  `# exit`

Although diskful machines can mount their local root filesystems in read-write mode, it is often desirable to treat the nodes as if they were diskless and mount the root filesystem read-only. This is especially important if the diskful nodes are intended to distribute the image to more diskless nodes. Any necessary filesystem alterations can be made in the master image, and synchronizing diskful nodes with the master image is accomplished easily with the `sync-node` script (section 6.6).

2.2.1 Run mk-sysimage

Once the master image has been created, the `mk-sysimage` script (section 6.1) must be run to prepare the image for use. This script alters the filesystem of the master image so that each node effectively sees a different 'view' of the image.

As described in the 'Implementation' section, `mk-sysimage` alters any files listed as a `LINK*` directive to enable the image to serve as the root filesystem for as many nodes with potentially many different functional roles. It will convert the distribution to be used as a read-only root filesystem. As a convenience, it will also remove (backup) any configuration

files that try to mount local disk devices or configure network interfaces, or any other configuration files that would create problems for client nodes.

On some distributions at may be desirable to manually alter the network configuration after `mk-sysimage` has run. For example, on an Ubuntu system, if network-manager is not disabled it may attempt to reconfigure your primary boot interface and the root filesystem could become unaccessible. To prevent this one could either disable network-manager (preferred) or try to tell network-manager to not touch your boot interface by adding the following to the `/etc/network/interfaces` file:

```
auto eth0
iface eth0 inet manual
```

Procedure 3 Run `mk-sysimage` on your image

- **Define your distribution**
  Edit `/etc/sysimage.conf` in the image to reflect the linux distribution being used.
  Add the appropriate `DISTRO` directive.

- **Run** `mk-sysimage`
  This will prepare the image to be used as a shared root filesystem for many nodes. It may be desirable to run with the `--dryrun` option the first time to see if the distribution patch will apply cleanly. The effects of `mk-sysimage` can always be reverted with the `--revert` option.
  # `mk-sysimage /var/lib/oneSIS/image`

2.2.2 Using git for revision control

System images can go through many revisions throughout their life cycle. It is extremely useful to be able to view and track these changes, have a complete history of changes, and be able to revert back to any previous state if necessary. This can all be accomplished by using the git (http://git-scm.com) version control system. See Appendix C for a brief HOWTO on using oneSIS with git.

3 Booting nodes

When booting, your machine should come up, boot a kernel, and mount a root filesystem. The kernel can come from the network via a mechanism such as PXE or EtherBoot, from a local disk, or even from onboard flash with LinuxBIOS. Methods described here are traditional diskless NFSroot, and booting diskless/diskful/mixed systems with initramfs.

3.1 Traditional diskless NFSroot

To boot using the traditional in-kernel NFSroot mechanism, the network interface must be compiled directly into the kernel rather than as a module. The same is also true for NFS client capabilities. Kernel-level autoconfiguration via DHCP is needed, and the ability to

have the root file system on NFS (CONFIG_ROOT_NFS) must be enabled. Once DHCP is configured, nodes can boot diskless by specifying the following on the kernel command line:

- `root=/dev/nfs ro ip=dhcp`

Booting in this way has the benefit that an initramfs doesn't need to be sent across the network at boot time, and the boot is a little faster.

3.2 Booting diskless/diskful/mixed systems with an initramfs

Although not required, using an initramfs to bootstrap oneSIS nodes can provide more flexibility than traditional NFSroot methods. The `mk-initramfs-oneSIS` script (described in section 6.4) is capable of automatically building an initramfs that can be customized for any kind of node, or generalized for an entire cluster of nodes.

Once the initramfs is created, nodes can boot into diskless/diskful/mixed environments by specifying the initramfs on the kernel command line:

- `initrd=initramfs-3.10.0`

The primary job of the initramfs is to bring up a network interface, configure the interface via DHCP, set the node's hostname, and mount its root filesystem. Specific kernel modules can be loaded by supplying options to `mk-initramfs-oneSIS`.

The root filesystem itself can be mounted via NFS or from a local disk. When a portion of the root filesystem has been deployed on a local disk, the initramfs can be configured to automatically mount those partitions before pivoting into the root filesystem.

An entire cluster (or any individual node or group within it) can be configured any way you want. The default initramfs template can also be extended to support almost any conceivable creative boot method.

The standard `mkinitramfs` or `mkinitrd` utility supplied with the distribution can still be used to bootstrap diskful nodes in many scenarios.

Note: Some distributions may not provide a kernel with CONFIG_PACKET support built directly into the kernel. If this is the case you will need to make sure to include the `af_packet` module when building an initramfs in order for the built-in DHCP client to work.

3.2.1 Specifying the root filesystem

The logic in the initramfs allows the root filesystem to be specified a number of different ways. Two methods for setting the root filesystem are:

- Using the `root=` kernel command-line parameter

- Using a `root-path` option in DHCP
  Note: Setting the `root-path` option for a node in DHCP will always override any `root=` parameter on the kernel command line.

In either case, the actual root can be specified in three ways:

- NFS root, ie: `192.168.1.1:/var/lib/oneSIS/image,v3,tcp,rsize=8192,wsize=8192`
- Local root specified by disklabel, ie: `LABEL=/`
- Local root specified by device, ie: `/dev/hda1`

3.2.2 Creating a new initramfs template

The initramfs template supplied by oneSIS can be used to bootstrap any `x86` compatible machine. Similar templates for other architectures may be included in future releases of oneSIS. It is also possible to create a new template and still have the functionality and convenience offered by `mk-initramfs-oneSIS` for creating new initramfs images.

`mk-initramfs-oneSIS` can use any derived templates to create initramfs images with added functionality. There is a specific place in the `init` script specifically designated for additional logic. Many other kinds of creative bootstrapping logic can be added there, without losing existing functionality.

3.3 Advanced bootstrapping methods

The initramfs mechanism that oneSIS uses to boot allows for a significant amount of flexibility with respect to bootstrapping a cluster node. Section 3.2 describes the basics of how to use an initramfs. Now we'll go into some more detail about how the initramfs works and detail some more advanced methods for bootstrapping cluster nodes.

3.3.1 Booting over bonded interfaces

In order to use channel bonding on an NFS-root node, the NFS root needs to be mounted over the bonded interface. This is all done in the initramfs. Be aware that many bonding modes require that the switch is properly configured to have the appropriate ports grouped together. Refer to `Documentation/networking/bonding.txt` in your linux source for more information. IP information can still be retrieved normally via DHCP using either of the network interfaces.

Procedure 4 Enable the initramfs to use ethernet channel bonding

- **Add bonding options to modprobe.conf in your image**

```
alias eth0 e1000
options bonding mode=balance-rr miimon=100
```

- **Tell mk-initramfs-oneSIS to bond the desired interfaces**

```
mk-initramfs-oneSIS  --basedir /var/lib/oneSIS/images/<image>
                     --bond-ifs eth0,eth1 /tftpboot/<initramfs-img>
                     <kernel-version> [OTHER_OPTIONS]
```

3.3.2 Tricks with PXE

Normally, the initramfs will load up, load a network driver, and attempt to contact a DHCP server to configure itself. However, on many modern systems, the PXE boot mechanism is already contacting DHCP and receiving IP information. This makes the DHCP step that runs from the oneSIS initramfs somewhat redundant. If you are interested in shaving some time off of your boot, or if for some reason you cannot run DHCP over your boot interface (as with boot over Infiniband (section 3.3.3)), oneSIS can use the IP information supplied via DHCP in the PXE phase of the boot. Note that it is necessary to use the '`root=`' kernel parameter to specify an NFS root as this data is not sent via DHCP in PXE. Also the host name must be set by one of the methods in section 5.11, such as an entry in `/etc/hosts`.

Procedure 5 Enable the initramfs to use IP information from PXE

- **Add '`ipappend 3`' and '`root=`' to your pxelinux configuration file:**

```
label initramfs
kernel vmlinuz-3.10.0
append  initrd=initramfs-3.10.0console=ttyS0,115200
        root=10.20.10.20:/var/lib/oneSIS/images/RedHat-EL-5.1,tcp,hard
ipappend 3
```

- **Tell mk-initramfs-oneSIS not to run DHCP**

```
mk-initramfs-oneSIS  --nodhcp /tftpboot/<initramfs-img>
                     <kernel-version> [OTHER_OPTIONS]
```

3.3.3 Boot over Infiniband

For clusters that are capable of it, booting nodes over an Infiniband interface via IP over InfiniBand (IPoIB) is a desirable way to boot. As of this writing, it is necessary to install a DHCP server that is capable of supporting IPoIB. More information on booting over IPoIB can be found at `http://mellanox.com/products/boot_over_ib.php`.

To network boot a oneSIS node over IPoIB requires using the '`ipappend 3`' pxelinux option detailed in section 3.3.2. The initramfs then needs to include kernel modules for the InfiniBand driver and IPoIB. Using OFED and a ConnectX card, this would look like:

```
mk-initramfs-oneSIS  -w mlx4_ib -w ib_ipoib /tftpboot/<initramfs-img>
                     <kernel-version> --nodhcp [OTHER_OPTIONS]
```

3.3.4 Using a ramdisk root

Simplicity is the ultimate sophistication. While oneSIS is capable of managing NFS-root based clusters, there are mechanisms for deploying simpler ramdisk-based root filesystems. Such minimal systems are ideal, for instance, for running cluster software such as XCPU (`http://xcpu.org`).

To use a ramdisk-root, simply use the '`mk-initramfs-oneSIS --ramdiskroot`' option.

3.3.5 Populating a ramdisk root

To add functionality to your minimal ramdisk root nodes, it is necessary to add certain files and directories. There are two mechanisms for doing this. One is to use the `COPY` directive in your initramfs.conf file (or respective command-line parameter). This will copy the given file or directory into your ramdisk. Unless otherwise specified, it will also copy all necessary run-time library dependencies to the ramdisk.

Another method for populating your ramdisk is to use overlays. Overlays are simply directory trees that are added to an initramfs image that perform a certain function. As an example, consider an overlay that is purpose-built to do one and only one thing: run HP linpack. The overlay could look like this:

```
overlay-xhpl/HPL.dat
overlay-xhpl/etc/init.d
overlay-xhpl/etc/init.d/rcS
overlay-xhpl/bin/xhpl
overlay-xhpl/lib64/libc.so.6
overlay-xhpl/lib64/ld-linux-x86-64.so.2
overlay-xhpl/lib64/libpthread.so.0
```

The overlay contains the linpack binary (`xhpl`), some shared objects needed by the binary, an input file (`HPL.dat`) and a start script (`/etc/init.d/rcS`).

The `rcS` file is always run when entering a ramdisk root, so this is where anything that needs to run gets started from. Depending on whether your distribution already has one or not, you can create an etc/init.d/rcS in your image and use a `COPY` directive to put it in your ramdisk image. Otherwise you'll need to create an overlay to include at least that file.

Any `root-path` directory specified in DHCP (or on the kernel command line) is automatically mounted via NFS on the `/mnt` directory unless `--nonfs` is given to `mk-initramfs-oneSIS`. Any other filesystems need to be mounted via the `rcS` script or an etc/fstab file.
For our example linpack initramfs above to be able to start linpack and send output to a useful place, we need an `rcS` start script like such:

```
#!/bin/sh
hostname=$(hostname)
export OMP_NUM_THREADS=8
/bin/xhpl |tee /mnt/xhpl-$hostname.out
```

3.3.6 Booting over alternate network filesystems

As it turns out, NFS is not the only game in town. There are several other network filesystems that could just as easily serve a shared root filesystem to a number of cluster nodes. As of yet there is no official support in oneSIS for alternate network (or parallel) filesystems, but it has been shown to work. To use one, it is necessary to edit an initramfs template (see section

3.2.2) to add the bits necessary for mounting root over the network filesystem of your choice. The new template can then be used via '`mk-initramfs-oneSIS --template=<template>`' to create an initramfs image.

3.3.7 Initramfs configuration

The `mk-initramfs-oneSIS` script has many options that are available for use. These options can be specified on the command line, but it might also be desirable to make one or more options permanent by including them in the `etc/oneSIS/initramfs.conf` file. If you run `mk-initramfs-oneSIS` with the '`--basedir`' option, be aware that the configuration used is in `<basedir>/etc/oneSIS/initramfs.conf`. Alternate config files can be specified using the '`--config`' option. A description of `initramfs.conf` configuration directives can be found in Appendix B.

3.3.8 Custom initramfs logic

If there is a need to add any custom logic to bootstrap some nodes, it can be added by providing a custom script in `<basedir>/etc/oneSIS/initramfs-postnetcheck.sh`. This script will be copied to any generated initramfs image and will execute after the initramfs network check has completed. For instance, since there is no RAID support built into the ramfs, you can add a script that runs `mdadm` to configure your RAID boot device.

4 Implementation

The root filesystem of every cluster node is bit-for-bit identical, whether it resides on a local disk or is mounted via NFS. To achieve different behavior for each functional role within a cluster, oneSIS uses a technique called a 'linkback' that involves creating symbolic links to and from the oneSIS RAM disk at run-time.

4.1 Role abstractions

Clusters commonly consist of groups of many machines that behave the same way. The large majority of nodes are usually used to compute scientific algorithms, run database applications, or provide high availability or failover services for web servers and the like. However, a smaller set of nodes in the cluster usually have auxiliary functions necessary to the operation of the cluster.

A typical cluster may have administration nodes, front-end nodes, login nodes, and nodes dedicated to providing filesystem I/O to the rest of the cluster. For these nodes, it is desirable to have a root filesystem image that is identical to the main 'compute' nodes. Although using the same filesystem image for 'admin' or 'IO' nodes is not required, for example as for 'compute' nodes, consistency helps ease administration and reduce the overall complexity of the system.

4.2 Node deviations

The independent behavior of nodes and functional groups of nodes can be configured in several ways. In a cluster, compute nodes may run different services (daemons), have different configuration files, mount different filesystems, and otherwise behave differently from other class of nodes in many ways.

These differences usually require deviations from the master image that configure unique behavior for each class of nodes. The types of deviations are grouped into three main categories: deviation of system services, deviation of files and directories, and deviation in usage of local hard disks (if they exist).

4.3 Utilization of the RAM disk

Most deviations in node behavior result from differences in configuration files in the root filesystem. For example, some nodes may need to mount different filesystems than those listed in the default configuration. Those nodes would require a different configuration in the `/etc/fstab` file.

For any file requiring variations between nodes, oneSIS replaces the file with a symbolic link pointing to its corresponding path in `/ram`. The original file is moved to a file with the same name, but with a '`.default`' extension.

At boot time, each node determines its role and uses the version of the file that corresponds to that node's class (see figure 1). Any node not configured to use an alternate file or directory uses the original '`.default`' file. This technique can be used to achieve different behavior for any node or class of nodes.

4.4 Inherited behavior (node sub-classing)

To further ease the management of node deviations, oneSIS allows subclasses to be defined that inherit all properties of their parent class. For example, 'compute.infiniband' and 'compute.myrinet' subclasses can be derived from a 'compute' class to account for differences in the network hardware used on respective nodes. All 'compute' classes behave the same, but 'compute.myrinet' classes can be configured to additionally run myrinet specific services.

Subclasses inherit all the behaviors of a parent class, can override those behaviors if desired, and can define any additional behaviors considered necessary. There is no explicit limit on the depth of sub-classing.

4.5 Boot-time configuration

At boot time, a node comes up and determines its hostname, usually in the initramfs via DHCP (with `use-host-decl-names` set to on). After `/sbin/init` is run, the system runs the primary oneSIS boot-time configuration script, `/sbin/rc.preinit`. The `rc.preinit`

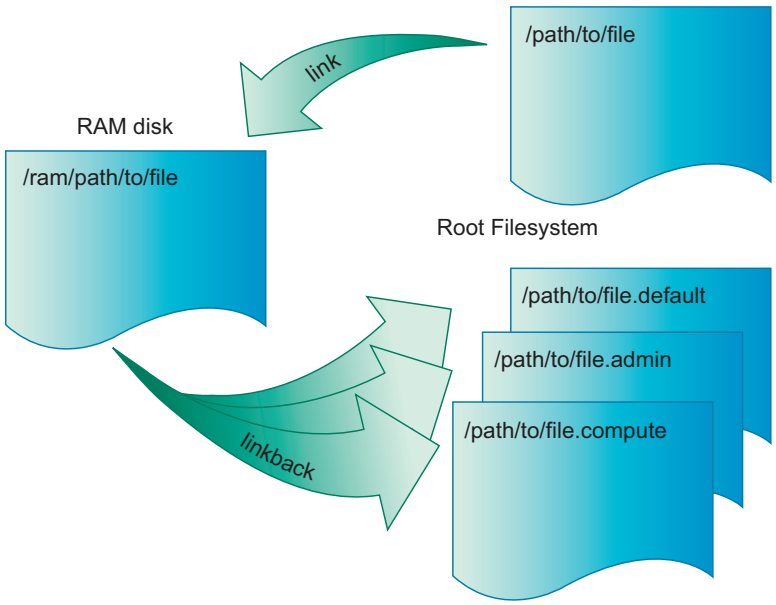

Figure 1: **RAM disk Usage. Configuration files for each class of node are linked back to at run-time.**

script determines which class the node belongs to and builds a RAM disk in `/ram` appropriate for the node.

The oneSIS RAM disk contains all necessary files and directories configured to help the node function as normal without a writable root filesystem. It has all class-specific and node-specific deviations. If a 'login' node needs a different `/etc/fstab`, `rc.preinit` creates a `/ram/etc/fstab` symlink that points to `/etc/fstab.login` in the master image, if it exists.

If there is no class-specific version of `/ram/etc/fstab`, that symbolic link will link back to the `/etc/fstab.default` file. This 'linkback' process provides flexibility. Any file in the master image can be changed or deleted on a per-class and per-node basis, allowing for fine-grained control of any file in the master image. All deviations in the filesystem are handled similarly.

4.6 Porting to a new distribution

When oneSIS finishes configuring the system, control is passed back to the normal boot scripts provided by the distribution. However, these boot scripts often attempt to do things that do not make sense in a read-only environment. Most of these quirks from the distribution's boot scripts are harmless, merely cluttering the normally aesthetic boot sequence with garbage, but some can be detrimental.

Commenting out the detrimental lines from the `rc` scripts usually eliminates the errors originating from the distribution's `rc` scripts. oneSIS does this automatically for several

distributions by applying a 'distribution patch' against the filesystem.

Currently, oneSIS includes patches for several distributions. Minor changes often are made to a distribution's `rc` scripts between sub-versions of a distribution release. This requires development of a patch to 'port' oneSIS to each version of a distribution.

Creating a patch for a newer version of an already supported distribution is simple. An older patch for the same distribution can be used as a model for the new patch. The primary goal of the patch is to ensure that the root filesystem is not mounted read-write at boot time. The patch also comments some actions in the `rc` scripts that try to write to the root filesystem. This results in a more aesthetic bootup.

At bootup many errors complaining about the 'Read-only file system' may appear on the console. Tracking down the source of these errors and commenting out the offending lines of code is not difficult. However, leaving the script intact and creating configuration directives so that the data is written to the RAM disk instead may be preferable in some cases.

In general, developing a patch for a new version of a distribution can be accomplished with only a few iterations of booting a machine.

5 Configuration

Special directives in the configuration file, `/etc/sysimage.conf` are used to manage cluster node behavior and node classes.

`/etc/sysimage.conf` centrally manages the behavior of every node booting into that image. The directives are few and simple with clear functions. The directives are used to define role abstractions for all the nodes and express the desired behavior of each role.

With the exception of the `NODECLASS*` directives, any directive in the configuration can be overridden by directives further down in the configuration. Additionally, any directive (except for `NODECLASS*` directives) can be limited to apply to only one or more classes of nodes or to individual nodes.

5.1 Specifying the distribution

oneSIS performs some distribution-specific operations. The most notable is the distribution patch, which is usually required to keep a distro's `rc` scripts from causing errors when booting.

Enabling and disabling system services in the default runlevel is also distro-specific, and some distro's may require some minor boot-time initialization tasks.

A directive is included in `/etc/sysimage.conf` to specify the distribution of the master image to enable oneSIS to handle these tasks for different distributions. If the specified

distribution is not currently supported, it may be necessary to create a distribution patch as mentioned in section 4.6.

oneSIS will operate even on an unsupported distribution, but some of the distro-specific features described above will not be available. Remember that any `SERVICE` directives in the `/etc/sysimage.conf` file must always be enabled in the default runlevel. The rest of the system will still work as expected.

5.1.1 DISTRO syntax

DISTRO <***name***> [***version***] [**-sp**] [**-kn**]
Specifies the *name* of the distribution of the master image, and optionally the *version*. Look in the `distro-patches` directory to see distributions name/version pairs that are currently supported.

-sp Skips the distribution patch normally applied by `mk-sysimage`.

-kn Skips the modification of network configuration files normally applied by `mk-sysimage`.

5.2 Defining node classes (functional roles)

Nodes must be configured to belong to a single class, although that class can be a derived subclass. Class names are completely arbitrary. oneSIS defines the class a node belongs to based solely on the node's hostname.

Node classes can be defined by perl-style regular expressions or by using a syntax to describe range expressions. A combination of multiple `NODECLASS*` directives can be used to describe a single class. For nodes with more than one matching `NODECLASS*` directive, but different class names, later directives will override earlier ones.

A classname may also be passed in on the kernel command line vie the `classname=` parameter. This can be used to set or override the oneSIS class for the given node.
Once a class is defined, the class name can be used in other directives to define behavior specific to that class.

5.2.1 NODECLASS syntax

NODECLASS_MAP <***node***> <***class***[***.subclass***]...>
Adds *node* to the specified *class*.

NODECLASS_REGEXP <***regexp***> <***class***[***.subclass***]...>
Adds all nodes matching the regular expression to the specified *class*. Refer to related perl documentation for the syntax of perl-style regular expressions.

NODECLASS_RANGE <***prefix[range]...suffix***> <***class***[***.subclass***]...>
Adds all nodes matching the range expression to the specified *class*.
Any occurrence of a numerical range within brackets matches hostnames having digits within

that range. For instance the expression `rack[1-4]node[1-32]` would match a host with the name `rack4node12`.
— **Note:** A *range* is always enclosed in [ ] brackets. and can be of the form a[-b][,x[-y]]... , where a<b and x<y

5.3 Creating node sub-classes

Any time a '**.**' occurs in the name of a class, a subclass is implicitly created (see section 4.4). If no `NODECLASS*` directive is present for the parent class, the subclass can still operate. For a cluster of 64 nodes, named `cn1` through `cn64`, if half use gigabit ethernet (gige) and the other half use myrinet, classes for each type of interconnect could be subclassed from a common 'compute' class as follows:

```
NODECLASS_REGEXP  cn\d+       compute
NODECLASS_RANGE   cn[1-32]    compute.gige
NODECLASS_RANGE   cn[33-64]   compute.myri
```

5.4 Defining node properties

Any node or any class that has been defined can also be assigned an arbitrary property. Properties can be used to define behavioral aspects that span over several classes and/or nodes. A node may only belong to one class, but it can have as many properties as desired.

PROPERTY <***property_name***> [**-c** ***class***[**,*class***]**...**] [**-n** ***node***[**,*node*]**...**]
[**-r** ***range***] [**-re** ***regexp***]

Defines a property that applies to the given nodes/classes.

> ***property_name*** is any alphanumeric string to be used to reference the property.
> ***-c*** grants the property to all nodes in the given classes.
> ***-n*** grants the property to the given nodes.
> ***-r*** grants the property to all nodes matching the given node ***range***.
> ***-re*** grants the property to all nodes matching the given regular expression.

5.5 Specifying the RAM disk size

The oneSIS RAM disk often only needs to contain links and several small files. Server logs and similar output can be configured to be sent to log servers or NFS-mounted directories to eliminate the need for persistent local storage.

The oneSIS RAM disk defaults to use half of physical memory. The RAM disk size can be bounded or made larger if necessary.

The RAM disk is first created by `rc.preinit` at boot time, but the `update-node` script can be used to re-size the RAM disk according to the current configuration after a node is booted.

RAMSIZE <***max_size*** [k|m|g]> [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]
Directs oneSIS to create a RAM disk that can grow to at most *max_size* size. The max size defaults to half of physical RAM. Units can be specified in kilobytes(k), megabytes(m), or gigabytes(g).

max_size is the upper bound on the size of the ramdisk created by oneSIS.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

5.6 Configuring RAM disk elements

Any files or directories that should exist in the ramdisk are configured with the `RAM*` and `LINK*` directives. These directives can be used to create writable files and directories in the RAM disk, and the `LINK*` directives can be used to essentially make any file or directory in the root filesystem writable by converting it into a link into `/ram`, the oneSIS RAM disk.

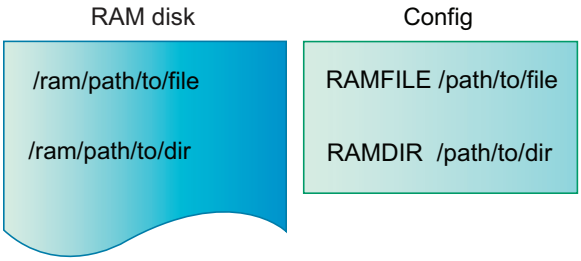

Figure 2: **Defining ram-based files with the RAM* directives.**

The `RAMFILE` directive creates an empty file in the RAM disk, and the `RAMDIR` directive creates an empty directory.

The `LINKFILE` directive also creates an empty file in the RAM disk, and the `LINKDIR` directive creates an empty directory. In addition the `LINK*` directives direct the `mk-sysimage` script to alter the corresponding files in the root filesystem to become symlinks to `/ram`.

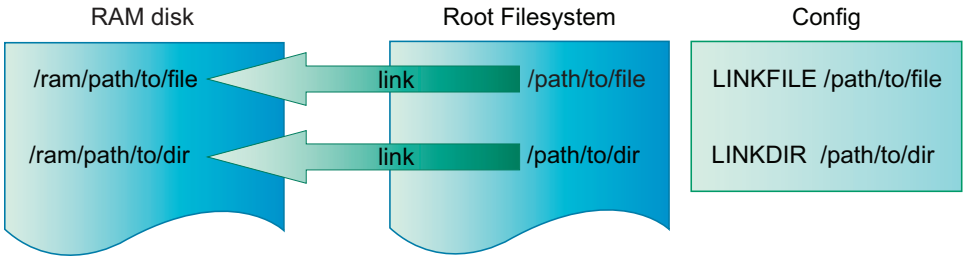

Figure 3: **Defining links to ram-based files with the LINK* directives.**

Both sets of directives have their role. The `mk-sysimage` script creates the necessary links in the master image. `rc.preinit` creates the corresponding files and directories in the `/ram` at boot-time or after `update-node` boots a node.

File permissions and ownership are mirrored from the root image, but can be set explicitly for `RAM*` directives.

5.6.1 Duplicating files into /ram

At times having more than just an empty file or directory in `/ram` may be desirable. Larger clusters or performance-sensitive systems may want to duplicate certain system libraries into `/ram` to alleviate the overhead of going to NFS.

Other reasons to have certain files/directories copied into `/ram` are that any `RAM*` or `LINK*` directives can specify the `-d` flag to duplicate the specified files or directories into `/ram`. The maximum size of the ramdisk may need to be adjusted accordingly.

5.6.2 RAM* and LINK* syntax

RAMDIR <*dir*> [**-d**] [**-cl**] [**-c** *class*[,*class*]...] [**-n** *node*[,*node*]...] [**-p** *property*[,*property*]...] [**-m** *mode*] [**-u** *user*] [**-g** *group*]

Creates a directory in the RAM disk.

-d causes the directory and all of its contents to be duplicated into `/ram` (default behavior is to create an empty directory).

-cl flags the given *dir* in `/ram` as cleanable for any ′`update-node --clean`′ operations.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

-m manually sets permissions of the file in `/ram`.

-u manually sets the owner of the file in `/ram`.

-g manually sets the group of the file in `/ram`.

Note: All permissions and ownership are mirrored from the image by default.

RAMFILE <*file*> [**-d**] [**-c** *class*[,*class*]...] [**-n** *node*[,*node*]...] [**-p** *property*[,*property*]...] [**-m** *mode*] [**-u** *user*] [**-g** *group*]

Creates a file in the RAM disk.

-d causes the file to be duplicated from the image into `/ram` (default behavior is to touch the file).

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

-m manually sets permissions of the directory in `/ram`.

-u manually sets the owner of the directory in `/ram`.

-g manually sets the group of the directory in `/ram`.

Note: All permissions and ownership are mirrored from the image by default.

LINKDIR <*dir*> [**-d**] [**-cl**] [**-c** ***class***[**,*class***]**...**] [**-n** ***node***[**,*node***]**...**] [**-p** ***property***[**,*property***]**...**]

Creates the directory in `/ram`, and changes the corresponding directory in the image into a link pointing to the directory in `/ram`.

-d causes the directory and all of its contents to be duplicated into `/ram` (default behavior is to create an empty directory).

-cl flags the given *dir* in `/ram` as cleanable for any '`update-node --clean`' operations.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

LINKFILE <*file*> [**-d**] [**-c** ***class***[**,*class***]**...**] [**-n** ***node***[**,*node***]**...**] [**-p** ***property***[**,*property***]**...**]

Creates a file in `/ram`, and changes the corresponding file in the image into a link pointing to the file in `/ram`.

-d causes the directory and all of its contents to be duplicated into `/ram` (default behavior is to touch an empty file).

-cl flags the given *dir* in `/ram` as cleanable for any '`update-node --clean`' operations.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

Note: With all `RAMDIR`, `RAMFILE`, `LINKDIR`, and `LINKFILE` directives, any wildcard syntax consisting of *, ?, [ ], or { } characters can be used to specify multiple files/directories in accordance with the POSIX.2 `glob()` function.

5.7 Using a read-only root filesystem

Using a read-only root filesystem can be tricky at first. Many startup scripts and even some daemons expect to be able to write to the root filesystem, and fail if they cannot.

To solve these kinds of problems, carefully watch the console of a booting node for errors related to a 'Read-only file system'. When these kinds of errors occur, determine which file or directory was trying to be written to and include LINKDIR or LINKFILE directives in the configuration as appropriate.

As an example, several distributions like to write `.pid` files into `/var/run` to keep track of the process IDs of running daemons. At boot time, when these daemons try to start, there will be complaints about a 'Read-only file system' when `/var/run` is not writable. One solution for this problem is to add the following directive to the `sysimage.conf` file of the master image:

LINKDIR /var/run

Don't forget to run `mk-sysimage` on the image after creating any `LINK*` directives. Now when the node boots, it will be able to write to `/var/run`, since the directory now effectively lives in `/ram`, the oneSIS RAM disk.
Many distributions have directories nested under `/var/run`, and while the above directive may work for most scenarios, it may be desirable to duplicate all of `/var/run` by adding the `-d` option:

LINKDIR /var/run -d

This will duplicate your image exactly as-is, so you may want to clean out any existing `.pid` files from that directory in your image.
Configuring the ability to write to a single file, such as `/var/lib/random-seed`, can be handled similarly. Rather than link all of `/var/lib` into the RAM disk with a LINKDIR directive, we can link just the one file the needs to be writable:

LINKFILE /var/lib/random-seed -d

Again, don't forget to run `mk-sysimage` on the image. This directive also has the `-d` flag so that the `random-seed` file is not created empty.

Closely watch the bootup and add any needed directives to the configuration to handle the idiosyncrasies of a read-only root filesystem.

Important tip: It usually helps to disable most of the unnecessary daemons enabled by default in many distributions. You can do this on RedHat compatible distributions by chroot'ing into the image and running '`chkconfig` <*service*> `off`'.

5.8 Setting up variant system services

oneSIS only needs to handle services that vary between nodes. If a service runs on every node, it should be handled through the distribution's normal mechanisms, for instance `chkconfig` on RedHat systems.

When a service should only run on some subset of nodes using the image, `SERVICE` directives can be used to specify which services should run on which nodes. The `mk-sysimage` script must be run after adding or removing a SERVICE directive.

5.8.1 SERVICE syntax

SERVICE <*service*> [**-c** *class*[,*class*]...] [**-n** *node*[,*node*]...] [**-p** *property*[,*property*]...]

Directs oneSIS to start the named *service*. The name of the service must match the name of its start script in the `/etc/init.d` directory.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

5.9 Using linkbacks

Linkbacks are the primary technique for defining variant configurations between nodes. A linkback causes a file or directory to become a link into the oneSIS RAM disk, which can then point back to a target in the master image based on a node's class, hostname, or property.

A linkback can have several potential targets. The literal '`CLASS`' target causes the linkback to point back to the original filename appended with an extension that is the name of the node class as determined by the `NODECLASS` directives. Similarly, the literal '`NODE`' target causes the final target to point back to the original filename appended with an extension that is the hostname of the node. Since a node can potentially have many properties, the literal '`PROPERTY`' target causes the linkback to point to the first existing alphanumeric file or directory having an extension that matches one of the node's defined properties.

When not using `CLASS`, `NODE`, or `PROPERTY`, a linkback target can be given as any arbitrary pathname. This path will be interpolated to replace any instance of '`$CLASS`' and '`$NODE`' with the class name and hostname of the node, respectively.

Note: Some files cannot have linkbacks created for them for various reasons. The most notable of these are `/etc/inittab`, `/boot/grub/menu.lst`, and `/etc/sysimage.conf`.

5.9.1 Forcing a linkback

When oneSIS creates the linkback symlinks at boot time, it normally checks to make sure that the symlink target actually exists in the master image first. If the target does not exist, the linkback ends up pointing back to the '`.default`' for the specified file.

However, the linkback can be forced even if the target doesn't exist. Using the `-f` flag of the `LINKBACK` directive will cause the link back to the target to be created even if the target doesn't exist at boot time.

Since oneSIS creates these links before any other system initialization is done, remote filesystems specified in `/etc/fstab` are typically not mounted yet. The `-f` flag can be especially useful to point to locations on a network filesystem that have not been mounted yet.

5.9.2 Linkbacks with sub-classes

Nodes defined as a subclass of another class will attempt to create a linkback for each parent class if the target exists. If no appropriate targets are found, a linkback to the '`.default`' file is created.

For example, if a node in the '`compute.gige`' class has a '`CLASS`' linkback defined for `/etc/fstab`, oneSIS will attempt to linkback to `/etc/fstab.compute.gige`. If that target doesn't exist, it will attempt to linkback to `/etc/fstab.compute`, then finally `/etc/fstab.default`.

5.9.3 The case for hidden linkbacks

In several distributions, it is common for system scripts to try to operate on all files in a directory to accomplish a task. For example, RedHat tries to bring up every interface matching `/etc/sysconfig/network-scripts/ifcfg-*` at boot time.

If a LINKBACK is defined for /etc/sysconfig/network-scripts/ifcfg-eth0, the glob above will match all of the linkback targets for that file, including `ifcfg-eth0.default` and all node-specific versions of the file. The system script then tries to bring up the interface several times with potentially different configurations. This creates many problems, which could result in losing all static configurations.

For such cases it is desirable to 'hide' (with the LINKBACK `-h` flag) all linkback targets so that the system scripts still function correctly.

When a linkback is hidden, all linkback targets will have a '.' pre-pended to the name, so `ifcfg-eth0.default`, when hidden, will become `.ifcfg-eth0.default`. Remember that all variants of the file also will need to be hidden. If you want a NODE-specific version of `ifcfg-eth0` for admin1, the file `.ifcfg-eth0.admin1` needs to be created to hold the configuration.

Note: Hidden LINKBACK directives only apply to CLASS, NODE, and PROPERTY specific linkback targets.

The `mk-sysimage` script transitions the `.default` file between hidden and un-hidden according to the configuration, but it does not alter any other CLASS, NODE, or PROPERTY specific linkback targets. The administrator must ensure that all of these targets are hidden or un-hidden according to the configuration.

5.9.4 LINKBACK syntax

LINKBACK <***file***|***dir***> <**CLASS**|**NODE**|**PROPERTY**|***target***>
[**-h**] [**-f**] [**-c** ***class***[**,***class***]**...**] [**-n** ***node***[**,***node***]**...**] [**-p** ***property***[**,***property***]**...**]

A LINKBACK converts a file or directory in the image to point to its corresponding location in `/ram`. At boot time, the location in `/ram` then points back to a location in the

master image based on the node's class, hostname, or a defined property.

– The `CLASS` target causes the linkback to point to the file or directory having an extension that matches the node's class name.

– The `NODE` target causes the linkback to point to the file or directory having an extension that matches the node's hostname.

– The `PROPERTY` target causes the linkback to point to the first existing alphanumeric file or directory having an extension that matches one of the node's defined properties.

– Any arbitrary file or directory can be specified as the direct *target* of a linkback. This *target* is interpolated to replace any instance of '`$CLASS`' with the node's class name, and any instance of '`$NODE`' with the node's hostname.

-f forces the linkback to point to the specified target even if the target doesn't exist. **Note**: The *-f* option cannot be used with PROPERTY linkbacks.

-h specifies that the linkback target should be 'hidden'. **Note**: Only CLASS, NODE, and PROPERTY linkbacks can use the *-h* option.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

5.10 Management of local disks

On clusters with local disk(s), oneSIS can partition and initialize local hard disks to be swap partitions or local filesystems at boot time.

When detecting disks, oneSIS assigns a number to each disk it finds as determined by the kernel order seen in `/proc/partitions`. This allows the configuration to specify that it wants to use the first disk, for example, rather than requiring a specific device name. oneSIS can detect any disk device that shows up as a normal block device if the appropriate drivers are loaded.

Note: For `rc.preinit` to operate on a disk, the driver must already be loaded. This means the disk driver needs to either be compiled directly into the kernel, or the module needs to be loaded from an initramfs.

5.10.1 Dynamic or static partitions

There are two forms of disk directives, those that dynamically create disk partitions each time at boot time, and those that create permanent locally deployed portions of the master image.

The `DISKMOUNT` and `DISKSWAP` directives cause the specified disk(s) to be re-partitioned (if necessary) and filesystems to be created every time at boot time. These directives are best used for creating filesystems for temporary storage, such as `/tmp`.

Note: Local disk partitions that need to retain data across a reboot should be handled normally with `/etc/fstab`.

5.10.2 DISK* syntax

DISKMOUNT <***disk***[,***disk***]...]> <***size***[%]> <***mointpoint***> [**-t** ***fstype***] [**-l** ***label***] [**-r** ***raidlevel***] [**-f**] [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]

Creates and mounts a local filesystem of size *size* on the specified *disk*.

– The *disk* parameter is either a number specifying disk order as seen in `/proc/partitions`, or the name of a disk device (ie: `/dev/sda`). If more than one *disk* is given, a RAID level must be specified with the *-r* parameter.

– The *size* parameter can either be a percentage of the disk to use, or the exact size in megabytes. If *size* is larger than the remaining capacity of *disk*, the remainder of the disk is used.

– The *mountpoint* parameter specifies where the filesystem should be mounted. *mountpoint* must be a directory that already exists in the image.

-t specifies the filesystem type to create (the default filesytem type is `ext3`).

-l specifies a filesystem label to use for this partition (the default is to create a label that matches the *mountpoint*).

-r When more than one disk is specified, specifies the RAID level to create across the devices.

-f enables fastboot. With this option enabled, the filesystem will only be re-created at boot time if and only if the on-disk label does not match the configured label.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

DISKSWAP <***disk***[,***disk***]...]> <***size***[%]> [**-r** ***raidlevel***] [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]

Creates and enables a swap partition of a specified *size* on the specified *disk*.

– The *disk* parameter is either a number specifying disk order as seen in `/proc/partitions`, or the name of a disk device (ie: `/dev/sda`). If more than one *disk* is given, a RAID level must be specified with the *-r* parameter.

– The *size* parameter can either be a percentage of the disk to use, or the exact size in megabytes. If *size* is larger than the remaining capacity of *disk*, the remainder of the disk is used.

-r When more than one disk is specified, specifies the RAID level to create across the devices.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

For example, to have nodes create (at boot time) a 3GB swap partition on the first local disk and use the rest of the disk for a local `/tmp` directory, use the following directives:

```
DISKSWAP   1  3000
DISKMOUNT  1  100%  /tmp
```

5.10.3 A word of caution

In environments with a mix of persistent and non-persistent local disks, it is important to understand and use the `DISK*` directives very carefully.

By default on most Linux systems, device names are not guaranteed to be consistent across a reboot. One must be aware that a failed disk could cause the device ordering to come up inconsistently and potentially cause damage to subsequent local disks. One solution for persistent device names is to use `udev` and configure it to have strong associations between disk devices and their device names.

5.10.4 Deploying parts of the image to local disk

For some machines, having all or part of the root filesystem reside on a local disk may be desirable. This is accomplished by using the `DEPLOYMOUNT` and `DEPLOYSWAP` directives in combination with the `mk-diskful` script. These directives cause oneSIS to create partitions and copy portions of the root filesystem to one or more local disks on a machine.
When deploying the entire image to local disk, keep in mind that oneSIS requires perl to operate. Since perl and its libraries are most often found in `/usr`, this means that `/usr` must be available very early in the boot process. This is easily accomplished simply by not deploying `/usr` on a separate partition, or by making sure that it gets mounted from within the initramfs.

If a `BOOTLOADER` directive is defined for a node, and the `/boot` directory is deployed on a local disk, the specified bootloader will be installed.

5.10.5 DEPLOY* syntax

DEPLOYMOUNT <***disk***[,***disk***]...]> <***size***[%]> <***mointpoint***> [**-t** ***fstype***] [**-r** ***raidlevel***] [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]

Directs the `mk-diskful` script to create and mount a filesystem partition of a specified *size* on the specified *disk*, and copy the corresponding directory tree in the master image to that partition.

– The *disk* parameter is either a number specifying disk order as seen in `/proc/partitions`, or the name of a disk device (ie: `/dev/sda`). If more than one *disk* is given, a RAID level must be specified with the *-r* parameter.

– The *size* parameter can either be a percentage of the disk to use, or the exact size in megabytes. If *size* is larger than the remaining capacity of *disk*, the remainder of the disk is used.

– The *mountpoint* parameter specifies a directory in the image that should be deployed as a partition on the local disk of a machine.

-t specifies the filesystem type to create (the default filesytem type is `ext3`).

-r When more than one disk is specified, specifies the RAID level to create across the devices.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

Label Note: For `ext*` filesystems, a disklabel matching the *mountpoint* will be created on the filesystem. This label can be used to auto-mount the filesystem from an initramfs, or can be used in `/etc/fstab` to mount by label (ie: `LABEL=/`).

RAID Note: If any RAID configurations are deployed it will be necessary to include the *SETUP_RAID* directive in the appropriate `initramfs.conf` file or use the `--raid` option for `mk-initramfs-oneSIS` (section 6.4).

DEPLOYSWAP <***disk***[,***disk***]...> <***size***[%]> [**-r** ***raidlevel***]
[**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]

Directs the `mk-diskful` script to create and enable a swap partition of size *size* on the specified *disk*.

– The *disk* parameter is either a number specifying disk order as seen in `/proc/partitions`, or the name of a disk device (ie: `/dev/sda`). If more than one *disk* is given, a RAID level must be specified with the *-r* parameter.

– The *size* parameter can either be a percentage of the disk to use, or the exact size in megabytes. If *size* is larger than the remaining capacity of *disk*, the remainder of the disk is used.

-r When more than one disk is specified, specifies the RAID level to create across the devices.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

5.10.6 SYNCDIR syntax

SYNCDIR <***path***> [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]

Specifies the location of the master image to the `sync-node` program (see section 6.6).
Note: This directive is only applicable to diskful nodes.

– *path* can either be the a location where the master image is NFS mounted on the node, or a '`host:path`' location of the image.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

5.10.7 EXCLUDESYNC syntax

EXCLUDESYNC <***path***> [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]

Specifies files/directories that should **not** be synchronized with the master image when `sync-node` (section 6.6) is run.
Note: This directive is only applicable to diskful nodes.

– *path* specifies a file or directory that should not be synced.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

Network filesystems mounted on a node are automatically excluded, so those directories do not need EXCLUDESYNC directives. However, any SAN filesystem that presents itself to the OS as block device (eg: fiber-channel and iSCSI) will likely need one.
This is crucial for diskful nodes mounting filesystems from any storage that appears to the system as a local device. Any attempt to synchronize that mountpoint would synchronize it with the (probably empty) directory in the master image, possibly resulting in data loss.

5.10.8 EXCLUDEDEPLOY syntax

EXCLUDEDEPLOY <***path***> [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...]

Specifies files or directories that should **not** be deployed to nodes when the `mk-diskful` command (section 6.5) is run.
Note: This directive is only applicable to diskful nodes.

– *path* specifies a file or directory (glob) that should not be synced.

-c limits the directive to apply only to the given classes.

-n limits the directive to apply only to the given nodes.

-p limits the directive to apply only to nodes having one or more of the given properties.

This is useful for excluding directories on diskful node deployments. As an example, if you are using git for revision control for your image (see section **??**), you could use this directive to keep the *.git* directory from being deployed.

5.10.9 Booting from a local disk

oneSIS supports booting from a local disk using either the `grub` or `lilo` bootloader. Other methods can still be used, but oneSIS does not handle them automatically.

For a bootloader to be installed, a `/boot` directory must be deployed on a local disk. The bootloader is installed onto the disk containing the `/boot` directory. A working configuration for the chosen bootloader is necessary (ie: in `lilo.conf` or `grub.conf`).

It is not necessary to install a bootloader even if the entire root filesystem is on a local disk. Any node capable of network booting can still retrieve its kernel and initramfs from a network resource such as DHCP and PXELINUX.

Alternatively, NFSroot nodes can create a single `/boot` partition on a local disk, install a bootloader, and load the kernel off the local disk, but still mount the root filesystem accessed via NFS. Loading the kernel from a local disk can help reduce network contention at boot-time when many machines power on all at once.

Many options exist to boot any node or functional group of nodes (locally or from the network) into a root filesystem that is either local, NFS mounted, or a combination of the two. The best scenario depends on the function of the node and the situation.

5.10.10 BOOTLOADER syntax

BOOTLOADER <**grub**|**lilo**> [**-c** ***class***[**,*class***]...] [**-n** ***node***[**,*node***]...] [**-p** ***property***[**,*property***]...]

Directs the `mk-diskful` script to install a bootloader onto the master boot record of a disk.

Either **grub** or **lilo** must be specified, and the given bootloader installed on the system.
-c limits the directive to apply only to the given classes.
-n limits the directive to apply only to the given nodes.
-p limits the directive to apply only to nodes having one or more of the given properties.

5.11 Manually setting a hostname

A substantial dependency of oneSIS is that the node's hostname must have already been set when `rc.preinit` runs at boot time. For this reason the hostname for diskless nodes is normally set (via the `use-host-decl-names` option in DHCP) in an initramfs prior to running `/sbin/init`.

There are alternatives that may prove useful for diskful nodes running an initramfs supplied by the distribution (which typically do not set a hostname), some other initramfs not created by oneSIS, or when using a `NODHCP` configuration (section 3.3.2).

If a node reaches `rc.preinit` without having a hostname set, oneSIS can determine it from the first of the following methods (in order) which succeeds:

1. By consulting a list of `MAC_ADDR` directives (subsection 5.11.1).
2. From a `hostname=` parameter on the kernel command line.
3. By looking in `/etc/hosts` for the address specified by a kernel command line `ip=` parameter (this requires that `/etc/hosts` is not the target of a `LINKBACK` and contains entries for all the oneSIS nodes).
4. By reading it from `/etc/oneSIS/hostname`, which can be set after deploying a diskful node.

5.11.1 MAC_ADDR syntax

MAC_ADDR <*hostname*> <*mac_address*>

This directive can be used for nodes that do not retrieve their hostname from DHCP in an initramfs. If a node boots without a hostname set, the MAC address of every network interface is scanned. If the MAC address of any interface matches the *mac_address* of a directive, the node's hostname is set from the directive's corresponding *hostname* parameter.

For this directive to work, any referenced network interface's drivers must have already been loaded. This means the drivers need to be directly compiled into the kernel, loaded from an initramfs, or specified via a `ETH_PRELOAD` directive.

5.11.2 ETH_PRELOAD syntax

ETH_PRELOAD <*driver*[,*driver*]...>

This directive directs the boot-time script (`rc.preinit`) to load each ethernet driver specified in the directive. The driver gets loaded early on so that any `MAC_ADDR` directive in the configuration will be able to look at the mac address for all interfaces created by the given ethernet driver.

5.12 Manually setting a class

It may be desirable sometimes to force a node to temporarily belong to a particular class (without changing the configurition). This can be done by appending *classname=<class>* to your kernel command line while booting. For a diskless boot this usually means creating a custom PXE config and using the `pxe-config` command (section 6.9) to temporarily point your node at it.

5.13 MATCH_DNS_HOSTS syntax

MATCH_DNS_HOSTS <*yes*|*no*>

This directive causes oneSIS to look at DNS host names when operating with host ranges. This can be useful when you do not want to add an entry in `/etc/hosts` for every host you want to use `pxe-config` on.
Note: There must at least one *domain* or *search* configuration option present in `/etc/resolv.conf` and the node must be allowed to do a DNS zone transfer for those domains.

5.14 Configuring the power and console interfaces

There are many existing solutions for power management of cluster nodes. Utilities have been written to interface to many power controllers, and vendors often include their own power management utilities with their products.

Every power utility has a different way of representing the set of nodes on which to operate. Similarly, many different methods can be used to access the serial console of all cluster nodes.

oneSIS provides a generalized wrapper interface that can easily tie into any existing power or console management solution. It serves as a common interface for power and console management across all machines, eliminating the need for an admin to remember the particular command and syntax used for power management and console access on each particular group of machines.

The POWERCMD directive is a way to quickly describe how a particular power management utility works so that the `pwr` (section 6.7) command can then interface to it. Similarly, the CONSOLECMD directive can quickly describe how to access the remote console of any node, and thereafter the `consl` (section 6.8) command can be used to access a node's serial console.

5.14.1 POWERCMD syntax

POWERCMD <***function***> [**-c** ***class***[**,*class***]**...**] [**-n** ***node***[**,*node***]**...**] [**-p** ***property***[**,*property***]**...**]
<***command***>

> *function* can be one of: ON,OFF,CYCLE,STATUS,LEDON,LEDOFF,LEDSTATUS
> *command* is the shell command to use to perform the given *function*. Any valid bash command sequence, including pipes/redirects, is acceptable.
> ***-c*** limits the directive to apply only to the given classes.
> ***-n*** limits the directive to apply only to the given nodes.
> ***-p*** limits the directive to apply only to nodes having one or more of the given properties.

Every *command* must reference the *spec_id* of a SPECFORMAT directive (see section 5.15) by including 'SPEC:*spec_id*' in the appropriate place in the command sequence. The 'SPEC:*spec_id*'

text gets replaced with a hostname, or a range, etc., as defined in the SPECFORMAT corresponding to *spec_id*.

A simple *hostname* format may work in most cases, but may not operate as fast as one of the *range* formats. When using the *hostname* or *ipaddr* specformats, *command* is interpolated to replace any instance of '$NODE' or '$IP' with the hostname or IP address of the node being operated on, respectively.

5.14.2 CONSOLECMD syntax

CONSOLECMD [**-c** ***class***[,***class***]...] [**-n** ***node***[,***node***]...] [**-p** ***property***[,***property***]...] <***command***>

> *command* is the shell command to use to connect to a remote console. Any valid bash command sequence, including pipes/redirects, is acceptable.
>
> ***-c*** limits the directive to apply only to the given classes.
>
> ***-n*** limits the directive to apply only to the given nodes.
>
> ***-p*** limits the directive to apply only to nodes having one or more of the given properties.

Just as with POWERCMD, every *command* must reference the *spec_id* of a SPECFORMAT directive (see section 5.15) by including 'SPEC:*spec_id*' in the appropriate place in the command sequence. The 'SPEC:*spec_id*' text gets replaced with a hostname or a range, etc., as defined in the SPECFORMAT corresponding to *spec_id*.

A simple *hostname* format will work in most cases for remote console operations.

When using the *hostname* or *ipaddr* spec formats, *command* is interpolated to replace any instance of '$NODE' or '$IP' with the hostname or IP address of the node being operated on, respectively.

5.15 How to make a SPECFORMAT

oneSIS can interface to almost any conceivable power or console utility by generalizing the single point of commonality that any such utility must have: a host specification. To make use of a particular power or console utility, oneSIS needs to know the format that the utility uses to represent a single host or a group of hosts. Some power utilities operate on a single hostname. Others can operate in parallel on a range of hostnames. Others don't operate on hostnames at all, instead referencing IP addresses or particular ports on a power controller.

The oneSIS interface for power and console management can be used as long as any mapping exists between the hostname (or IP address) of a node and the resulting parameter that gets passed to the power management utility used for that host. The parameter itself could be a hostname, a port, or anything else required by the specific utility.

The hostname to parameter mapping can be defined directly in the configuration with a SPECFORMAT directive, or can be determined via more cumbersome methods involving combinations of shell commands in the POWERCMD or CONSOLECMD directives.

5.15.1 SPECFORMAT syntax

SPECFORMAT <***spec_id***> <***format***> [**NODE:///** | **IP:///**] [**SPEC:///**]

– ***spec_id*** can be any arbitrary single-word identifier.
– ***format*** must be one of the format names described below.
– ***NODE:///*** provides a way to translate a node's hostname using perl `s///` syntax before using it in a command
– ***IP:///*** provides a way to translate a node's IP address using perl `s///` syntax before using it in a command
– ***SPEC:///*** provides a way to translate the entire string generated by a SPECFORMAT substitution using perl `s///` syntax before using it in the final command

Every POWERCMD directive must reference exactly one *spec_id* from a SPECFORMAT description. The resulting formats are sent through a POWERCMD *command* and executed by the `pwr` script. The same is true for all CONSOLECMD directives.

A SPECFORMAT describes the hostname representation a particular command uses to specify which nodes to operate on. Several formats currently exist which can be used to describe a set of nodes.

Format name	Format behavior
hostname	The resulting command uses the hostname of each specified node and runs one command for each hostname.
ipaddr	The resulting command uses the IP address of each specified node and runs one command for each IP address.
basic_range	One or more node ranges are constructed. Each range is of the form: `prefix[a-b]`, where a<b. (ie: node[1-32])
ext_range	One or more node ranges are constructed. Each range is of the form: `prefix[a-b,x-y,...]`, where a<b and x<y. (ie: node[1-4,10-20,25])

Note: Adding a '`+`' to the end of any *format* causes that format to be used multiple times (space seperated) in a single command.

Consider the `ipmipower` utility as an example, which represents a range of nodes as `host[a-b,x-y]`. The ***ext_range*** format translates a set of hostnames into one or more ranges in the form acceptable to `ipmipower`.

The ***hostname*** and ***ipaddr*** formats are both used for commands that operate on a single hostname or IP address. Each hostname or IP address can optionally have a transformation applied to it before applying the given *format*, by using the **NODE:///** or **IP:///** options.

Similar to the NODE and IP translations, a final translation can be done on the formatted string before it gets substituted into a command. The **SPEC:///** option can define this post-translation, if needed

The **NODE:///**, **IP:///**, and **SPEC:///** translations can be any perl-style pattern replacement expression. Refer to the perl documentation (`man perlop` and `man perlre`) for details on using the `s///` operator.

In addition to normal perl syntax, the right-hand-side (replacement) portion of the translation expression can contain minimal inline perl code blocks within {} brackets. These code blocks can be used to replace patterns in the hostname or IP addresses with values computed from evaluating the inline code expression. This is useful for doing inline math on an IP address, when necessary. The {...} code blocks must be kept very simple as they cannot yet contain any spaces.

Note: The `--dryrun` option to the `pwr` and `consl` commands is useful when developing a SPECFORMAT, POWERCMD, and CONSOLECMD directives for an environment. It shows the commands that the current configuration can generate. A working configuration can be arrived upon fairly quickly by iterating through changes in your configuration and using the `--dryrun` option.

5.16 Including extra configuration directives

INCLUDE <***path***>
Includes the given file into the current configuration.

> *path* is the absolute pathname of a valid oneSIS configuration file.

Directives can be bundled together into groups and included (or removed) all at once by including a single extra config file. All directives from the included config file are applied as if they were inserted into the configuration at the exact point as the INCLUDE directive. This can be used to bundle configuration directives for specific scenarios, and then add or remove them all at once.

6 Utility Programs

oneSIS comes with several utility programs to aid in deploying and maintaining a cluster. Among other things, these programs help prepare a master image, make the master image conform to the configuration, update a running node's environment, build an initial ramdisk for bootstrapping cluster nodes, deploy the root filesystem (or portions of it) onto a

node's local disk, and synchronize a node with locally deployed portions of the master image.

Most utility programs have usage information that can be seen by running the command with no arguments.

6.1 mk-sysimage

Usage: `mk-sysimage [OPTION]... <basedir>`
This program prepares a pre-installed linux distribution to be used as a master image for oneSIS cluster nodes.

basedir should be the root of the client's linux image.

Options:

-d,	–dryrun	Preview changes
-r,	–revert	Revert all files and services back to normal
-c,	–config=FILE	Specify alternate configuration file
-p,	–patchfile=FILE	Specify alternate distribution patch
-sp,	–skippatch	Skip distribution patch
-q,	–quiet	Suppress output

The `mk-sysimage` script reads the oneSIS configuration file, `/etc/sysimage.conf`, and alters components of the filesystem for oneSIS to operate correctly. It creates some directories, applies the patch file for the specific distribution (see section 4.6), and performs other helpful tasks.

Several directives in `/etc/sysimage.conf` require altering a file in the image to point to its corresponding location in `/ram`. `mk-sysimage` creates any new symbolic links to `/ram` and the corresponding '`.default`' files or directories.

`mk-sysimage` automatically restores files in the master image to their original state when they are removed from the configuration. It can also revert the entire filesystem back to its original state with the `--revert` option.

To ensure that configuration changes are reflected in the system image, it is recommended that the `mk-sysimage` script be run after changing any `LINK*` directives in the configuration.

For an image located in `/var/lib/oneSIS/image`, `mk-sysimage` would be run with:

```
# mk-sysimage /var/lib/oneSIS/image
```

Directives can be safely added or removed from the `sysimage.conf` file in any order.

Note: `mk-sysimage` will attempt to patch the target distribution every time it is run. A warning will be displayed unless a patch exists for the distribution or the `--skippatch` option

is supplied. This is meant to encourage anyone hacking the `rc` scripts of a new distribution to develop a patch for it and feed that back to the oneSIS community.

Warning: If you manually alter your distribution's `rc` scripts, `mk-sysimage` will fail to apply the distribution patch and display long error messages. If you plan to do this, you can run `mk-sysimage` with the `--skippatch` option so it doesn't try re-patch the distribution.

6.2 update-node

Usage: `update-node [OPTION]... <--run>`

This script updates all files and directories in `/ram` that have changed in the configuration and starts/stops any services if necessary.

Options:

-r,	–run	This argument must be given to run the script
-c,	–clean	Removes files/directories in /ram not in the config
-d,	–dryrun	Shows updates without making them
-cf,	–config=FILE	Specify alternate configuration file
-q,	–quiet	Suppress output

The `update-node` script performs a very similar function as the boot-time script, `rc.preinit`. It updates all the files and directories configured in `/etc/sysimage.conf` that reside in the oneSIS RAM disk mounted on `/ram`. It will also resize the RAM disk if necessary.

If any new `RAM*` or `LINK*` directives are added to the configuration, running `update-node` on all nodes will ensure that their RAM disk is consistent with the new configuration.

```
# ssh node1 update-node --run
```

By default, if any directives are removed from the configuration, the corresponding files and directories are NOT deleted from the RAM disk.

To remove files and directories in `/ram` that are no longer specified in the config, the `--clean` option must be given to `update-node`. However, it is recommended to clean files no longer in the config without destroying useful data that may be stored in a `RAMDIR` or `LINKDIR`. To protect such directories from having useful data destroyed by an '`update-node --clean`' operation, a `-p` flag can be added to the configuration directive:

```
LINKDIR /var/lock/subsys -p
```

After protecting such directories, '`update-node -r --clean`' can safely be run on all nodes as often as necessary to clean up the oneSIS RAM disk and keep the nodes consistent with their configuration.

```
# ssh node1 update-node -r -c
```

6.3 copy-rootfs

Usage: `copy-rootfs [OPTION]... <-l | -r machine> <IMAGE_DIR>`

This program copies an installed linux distribution to a specified location.

IMAGE_DIR is the destination directory for the root image.

Options:

-l,	–local	Copy root filesystem from the local machine
-r,	–remote=MACHINE	Copy root filesystem from a remote machine
-e,	–exclude=DIR	Exclude contents of DIR from being copied
-d,	–dryrun	Show local/remote directories that would be copied
-v,	–verbose	Verbose output (copies much slower)

The `copy-rootfs` script copies an installed linux distribution into a new location to serve as a new master image for a cluster of nodes. The script recognizes which partitions reside on a local disk, and copies each one over in the correct order without recursively copying itself (for a local copy).

Since `copy-rootfs` attempts to copy any partitions mounted from a local disk, it may copy more than you want or need to be a part of the master image. To prevent this, run `copy-rootfs` with the `--dryrun` option to see a list of what the script intends to do. Any directories that shouldn't be copied over can be excluded with the `--exclude` option.

When copying the root filesystem from a remote machine, it is easiest if ssh keys are set up such that no password is required to ssh to the machine. If ssh keys are not set up, the script will prompt for a password several times (once for each remote partition being copied, and once to determine remote partitions).

A typical scenario to create a master image may look as follows:

```
# copy-rootfs -l -e /home /var/lib/oneSIS/image
```

This would copy the root filesystem of the local machine into another directory but exclude the contents of the `/home` directory.

6.4 mk-initramfs-oneSIS

Usage: `mk-initramfs-oneSIS [OPTION]... <`*`initramfs`*`> <`*`kernel-version`*`>`

This program prepares an initramfs for bootstrapping oneSIS nodes.

initramfs is the pathname of the initial ramdisk to create.
Kernel modules are used for the given *kernel-version*.

Options:

-o,	–overlay=DIR	Overlay one or more directories on top of the initramfs template being used
-b,	–basedir=DIR	Look for files relative to DIR (default: `/`)
-cf,	–config=FILE	Configuration file to use for tayloring initramfs. Any commandline parameters override settings specified in the config file (default: basedir/etc/oneSIS/initrams.conf)
-v,	–variant=STRING	If multiple variants of a config file exists, specify the class or node variant to use
-d,	–scsi	Include all scsi_hostadapter modules that are listed in basedir/etc/modprobe.conf
-p,	–preload=STRING	Add the specified module (loads before SCSI modules)
-m,	–module=STRING	Add the specified module (loads after SCSI modules)
-w,	–with=STRING	Add the specified module (deprecated)
-c,	–copy=PATH	Copy given PATH to the ramfs image
-rc,	–recursecopy=PATH	Recursively copy given PATH to the ramfs image
-nl,	–nolibs	Do not copy library dependencies for files copied to ramfs
-t,	–template=FILE	Use the specified initramfs template. (default: `/usr/share/oneSIS/initramfs-templates/initramfs-x86.ta`
-f,	–force	Force overwrite of an existing output file
-td,	–tempdir=DIR	Use alternate staging directory instead of /tmp
-r,	–readme	Show the README of a given initramfs (ie: to see what options were use
-q,	–quiet	Suppress output
-vb,	–verbose	Verbose output

—— Initrd Behavior Flags (deprecated) ——

-i,	–initrd	Output an initrd image instead of an initramfs image
-s,	–size=NUM	Hard code the size of an initrd. By default, the size is determined automatically and a small buffer is added on for usable empty space
-bs,	–buffersize=NUM	Specify initrd empty space buffer size (default: 1024)

—— Initramfs Behavior Flags ——

-rr,	–ramdiskroot	Use the ramdisk as the root filesystem
-ro,	–ram_overlay[=SIZE]	Overlay a RAM disk onto the NFS root filesystem
-am,	–automount	Auto-mount labeled partitions and swapon swap partitions from the initramfs
-rw,	–readwrite=STRING	Auto-mount specified labeled partitions read-write The string '`ALL`' will mount all partitions read-write
-nd,	–nodhcp	Don't run a DHCP client from the initramfs
-di,	–dhcp_if	Run DHCP over the specified interface. (default: eth0)
-dr,	–dhcp_retries=NUM	Attempt to retry DHCP this many times before failing (0 means infinite retries, default: 5)
-nr,	–nfs_retries=NUM	Attempt to retry mounting the NFS root NUM times before failing. (default: 5)
-nc,	–net_check=NUM	Perform NUM network connectivity checks before

An alternate method for booting oneSIS systems is to bootstrap using an initial ramdisk (initramfs). By using `mk-initramfs-oneSIS`, an initramfs can be built that is customized for an entire cluster or for any subset of nodes.

Kernel modules needed for NFS and those specified by any `eth0` aliases in `/etc/modules.conf` are included automatically in the initramfs and loaded at boot time. Likewise, any `scsi_hostadapter` alias in `/etc/modules.conf` will cause the corresponding driver to be loaded when the `--scsi` option is given.

Any other modules can be included with command-line arguments. All modules must exist in `/lib/modules/`*kernel-version* relative to the *basedir*.

For example, to create an initramfs for a node running a 3.10.0 kernel with an `e1000` network card and IDE disk support built into the kernel, assuming kernel modules are installed in `/lib/modules/3.10.0`, you would type:

```
# mk-initramfs-oneSIS -w e1000 /tftpboot/initramfs-3.10.0 3.10.0
```

One initramfs template is included with oneSIS that can be configured to perform several varying tasks (described in section 3.2). Others can be derived from this one to perform specialized pre-boot tasks.

Local disk partitions that have been created with `DEPLOYMOUNT`, or `DEPLOYSWAP` directives and the `mk-diskful` script (see section 6.5) can be mounted automatically (or swapped-on) from the initramfs.

To automount locally deployed partitions on the system described above:

```
# mk-initramfs-oneSIS -w e1000 -am /tftpboot/initramfs-3.10.0 3.10.0
```

Any locally deployed partitions can also be mounted read-write. One must be aware that system utilities may write to the filesystem and erase your carefully crafted symlinks to `/ram`, especially in directories like `/var` and `/etc` during a system boot. To automount the locally deployed `/etc` directory read-write:

```
# mk-initramfs-oneSIS -w e1000 -am -rw /etc /tftpboot/initramfs-3.10.0 3.10.0
```

Other behavior in the initramfs can be controlled by supplying options to `mk-initramfs-oneSIS`. If any other functionality is needed in the initramfs, a new template can be derived from an existing one to provide the extra functionality, as described in section 3.2.2.

6.5 mk-diskful

Usage: `mk-diskful [OPTION]... <--run>`

This script will convert an NFSroot node into a diskful node.

Options:

	–run	This argument must be given to run the script
-i,	–image=DIR	Specify the NFS location of the master image
-e,	–exclude=DIR	Exclude DIR from being copied
-r,	–reboot	Reboot the node when finished
-d,	–dryrun	Show directories that would be copied to each partition
-v,	–verbose	Verbose output (copies much slower)
-q,	–quiet	Suppress output

Although booting oneSIS nodes with NFS root filesystems is preferred, oneSIS fully supports booting from a local disk, mounting the root filesystem from a local disk, or mounting only specific directories of the root filesystem from a local disk.

The `mk-diskful` script can be used to deploy portions of the root filesystem onto partitions on a local disk. The script can be run on a node after it is booted into a normal NFS root with no mounted partitions on the target disk. Alternatively, it can be run by calling it as `init` directly from the kernel command line as follows (note the quotes):

```
# init='/sbin/mk-diskful --run -r -s'
```

Proper `DEPLOY*` directives must be listed in the `sysimage.conf` file of the system image to make the desired portions of the root filesystem diskful, and a `BOOTLOADER` directive must be defined if the node should boot from its local disk.

Any portion of a node's root filesystem can be configured to reside on a local disk, so any combination of NFS and local directories in the root filesystem is possible. Nodes having `/boot` on a local disk can be configured to boot a kernel and initramfs from the disk or may simply continue to boot off the network.

6.6 sync-node

Usage: `sync-node [OPTION]... <-i` *image*`> <`*directory*`>...`

Synchronizes a directory on a diskful or partially diskful oneSIS node with the master image.

At least one local *directory* to synchronize must be given.
The *image* parameter is required and must specify the host and remote path of the NFS-exported master image.

Options:

-i,	–image=HOST:DIR	Specify the location of the master image
-l,	–lilo	Run lilo
-a,	–all	Synchronize all local partitions
-e,	–exclude=PATTERN	Exclude files matching PATTERN
-d,	–dryrun	Preview changes
-q,	–quiet	Suppress output

When portions of the root filesystem exist on local disk partitions, it is necessary to synchronize these partitions with the master image as often as necessary. If a change is made to `/etc/passwd`, for instance, all nodes having a local `/etc` partition could be synchronized with:

```
# sync-node /etc
```

Currently, synchronizing only from an NFS-exported directory is supported. Synchronizing via other methods may be added if desired.

Note: Running '`sync-node /`' will synchronize only the partition that `/` resides on. If `/etc` is on another partition, '`sync-node /etc`' would need to be run to synchronize it. To synchronize all local partitions, use '`sync-node -a`'.

Warning: Running '`sync-node -a`' will attempt to synchronize all locally mounted partitions. However, if a `/data` directory, for example, is mounting a SAN storage device that appears to the system as a SCSI disk, '`sync-node -a`' will detect that it is local and attempt to synchronize `/data` with the (probably-empty) `/data` directory in the master image, resulting in possible data loss. It is advisable to use EXCLUDESYNC directives as appropriate, and use '`sync-node -a`' with caution around nodes with a SAN.

6.7 pwr

Usage: `pwr <FUNCTION> <NODESPEC>... [OPTION]...`

This program is a wrapper script that calls an external power command (specified in /etc/sysimage.conf) to power on/off cluster nodes.

FUNCTION can be one of: *on, off, cycle, status, ledon, ledoff,* or *ledstatus*
Note: Unambiguous short forms of the functions are also accepted.

NODESPEC can be:

[-h] HOSTNAME	
[-r] RANGESPEC	(any text with one or more RANGEs in brackets) a RANGE is of the form <*a*-*b* [,*x*-*y* | ,*z*]...>, where *a*<*b* and *x*<*y* ie: `cn[1-10,15,20-32]` or `su[1,4]cn[1-32]` or `my[1-32]nodes`
-re REGEXP	(perl-style regular expression matching hostnames)
-c CLASS	(oneSIS class name)

Options:

-h,	–host=HOSTNAME	Operate on hostname
-r,	–range=RANGESPEC	Specify a range of nodes to operate on
-re,	–regexp=REGEXP	Specify a regular expression of nodes to operate on
-c,	–class=CLASS	Specify a class of nodes to operate on
-p,	–parallelism=NUM	Specify the maximum number of parallel commands to run (default: no limit)
-d,	–dryrun	Show command(s) that would be executed
-q,	–quiet	Suppress output

The `pwr` script is a convenient wrapper script supplied to provide a unified interface for handling power management for cluster nodes. It enables the same command to be used on every cluster regardless of the underlying mechanisms for handling node power.

Note: At least one valid `SPECFORMAT` directive and a `POWERCMD` for each FUNCTION must be supplied for the `pwr` command to be able to perform that function.

The `pwr` script builds commands that it runs (in parallel) to power on, power off, cycle, or query the power status of a given set of nodes. It can also turn on, turn off, or query the status of a chassis LED (or similar mechanism) if that functionality is available.

To power on nodes named `cn1` through `cn100`:

```
# pwr on cn[1-100]
```

To power off all nodes with hostnames starting with `cn` and ending with `sn`:

```
# pwr off -re cn.*sn
```

To power cycle all nodes belonging to the ‘compute’ class:

```
# pwr c -c compute
```

Several different commands may be being issued under the covers. The actual command that is run is specified in /etc/sysimage.conf with a POWERCMD directive.

For example consider an environment using IPMI for remote power operations and assume hostname of the form `node1` use `node1-ipmi` for their ipmi interface. Yo have `pwr` use the

ipmipower utility in this scenario, you could add the following lines to /etc/sysimage.conf:

```
SPECFORMAT freeipmi_spec -h ext_range HOST:/$/-ipmi/
POWERCMD ON ipmipower --on -h SPEC:freeipmi_spec -u root -p calvin
POWERCMD OFF ipmipower --off -h SPEC:freeipmi_spec -u root -p calvin
POWERCMD CYCLE ipmipower --cycle -h SPEC:freeipmi_spec -u root -p calvin
POWERCMD STATUS ipmipower --stat -h SPEC:freeipmi_spec -u root -p calvin
```

6.8 consl

Usage: `consl <NODESPEC>... [OPTION]...`

This program is a wrapper script that calls an external console command (specified in /etc/sysimage.conf) to get on a remote node's console.

NODESPEC can be:

[-h] HOSTNAME	
[-r] RANGESPEC	(any text with one or more RANGEs in brackets) a RANGE is of the form <*a*-*b* [,*x*-*y* | ,*z*]...>, where *a*<*b* and *x*<*y* ie: `cn[1-10,15,20-32]` or `su[1,4]cn[1-32]` or `my[1-32]nodes`
-re REGEXP	(perl-style regular expression matching hostnames)
-c CLASS	(oneSIS class name)

Options:

-h,	–host=HOSTNAME	Operate on hostname
-r,	–range=RANGESPEC	Specify a range of nodes to operate on
-re,	–regexp=REGEXP	Specify a regular expression of nodes to operate on
-c,	–class=CLASS	Specify a class of nodes to operate on
-p,	–parallelism=NUM	Specify the maximum number of parallel commands to run (default: no limit)
-d,	–dryrun	Show command(s) that would be executed
-q,	–quiet	Suppress output

Like `pwr`, `consl` command is a convenient wrapper supplied to provide a unified interface for accessing the serial console of cluster nodes. Typically, only one serial console is accessed at a time, but if the underlying application supports it (for instance `conman -b`), multiple consoles can be accessed at the same time.

Note: `consl` requires at least one valid `SPECFORMAT` and `CONSOLECMD` directive in `/etc/sysimage.conf` to operate.

For example, if the cluster is set up such that the serial console of a node named `node1` is accessible by telnetting to `node1-term`, the following configuration could be used in (/etc/sysimage.conf):

```
SPECFORMAT spec1 hostname NODE:/$/-term/
CONSOLECMD telnet SPEC:spec1
```

To clear the screen and print a helpful message before opening each console:

```
CONSOLECMD clear; echo Connecting to $NODE; telnet SPEC:spec1
```

To open each console in a separate window:

```
CONSOLECMD xterm -T $NODE console -e telnet SPEC:spec1
```

Then to connect to the console of node1, run:

```
# consl node1
```

Or, to connect to several consoles:

```
# consl node[1-8]
```

6.9 pxe-config

Usage: `pxe-config <NODESPEC>... <--show|--list|PXE_CONFIG> [OPTION]...`

This will create any necessary symlinks to use the PXE config specified by PXE_CONFIG on the specified nodes.

NODESPEC can be:
[-h] HOSTNAME
[-r] RANGESPEC (any text with one or more RANGEs in brackets)
a RANGE is of the form <*a*-*b* [,*x*-*y* | ,*z*]...>, where $a<b$ and $x<y$
ie: `cn[1-10,15,20-32]` or `su[1,4]cn[1-32]` or `my[1-32]nodes`

Options:

-l,	–list	List available PXE configuration files
-s,	–show	Show which nodes are using which config file
-r,	–revert	Revert specified nodes back to the 'default' config
-d,	–dryrun	Show command(s) that would be executed
-q,	–quiet	Suppress output

This script is supplied as a convenience for operators using the PXELINUX package for network booting. PXELINUX allows individual nodes to use different configuration files by

looking for a file with the hex equivalent of the node's IP address.

`pxe-config` provides a helpful interface to specify individual configuration files for given nodes, list which configuration files are available, and show which nodes are using which configuration.

PXE configuration files are normally kept in a directory like `/tftpboot/pxelinux.cfg`. If you create a PXE configuration file called `/tftpboot/pxelinux.cfg/x86_64/3.10.0` containing your desired PXE configuration, you could direct nodes `node1` through `node100` to use that config with:

```
# pxe-config node[1-100] x86_64/3.10.0
```

6.10 myclass

Usage: `myclass [OPTION]...`

This is a very small program that will print the class name of the running node. It can be used to do additional scripting based on the node's class name.

Options:

-n,	–node=HOSTNAME	Specify the node to print the class name of
-c,	–class=CLASS	Show all nodes having the given class
-s,	–show	Show all classes of all nodes
-cf,	–config=FILE	Specify alternate configuration file

6.11 myprops

Usage: `myprops [OPTION]...`

This script prints the property name(s) of a oneSIS node. With no arguments, it prints the property names of the machine that executes it. The ordering of the property names is the order used for LINKBACK purposes.

Options:

-n,	–node=HOSTNAME	Specify the node to print the property name of
-p,	–property=PROPERTY	Show all nodes having the given property
-s,	–show	Show all properties of all nodes
-cf,	–config=FILE	Specify alternate configuration file

A sysimage.conf Configuration Directives

DISTRO <*name*> [*version*] [-sp] [-kn]
Section 5.1.1

MATCH_DNS_HOSTS <*yes*|*no*>
Section 5.13

INCLUDE <*path*>
Section 5.16

NODECLASS_MAP <*node*> <*class*[.*subclass*]...>
Section 5.2.1

NODECLASS_REGEXP <*regexp*> <*class*[.*subclass*]...>
Section 5.2.1

NODECLASS_RANGE <*prefix[range]...suffix*> <*class*[.*subclass*]...>
Section 5.2.1

PROPERTY <*property_name*> [-c *class*[,*class*]...] [-n *node*[,*node*]...]
[-r *range*] [-re *regexp*]
Section 5.4

SERVICE <*service*> [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.8.1

RAMSIZE <*max_size* [k|m|g]> [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.5

RAMDIR <*dir*> [-d] [-cl] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
[-m *mode*] [-u *user*] [-g *group*]
Section 5.6.2

RAMFILE <*file*> [-d] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
[-m *mode*] [-u *user*] [-g *group*]
Section 5.6.2

LINKDIR <*dir*> [-d] [-cl] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.6.2

LINKFILE <*file*> [-d] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.6.2

LINKBACK <*file*|*dir*> <CLASS|NODE|PROPERTY|*target*>
[-h] [-f] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]

Section 5.9.4

DISKMOUNT <*disk*[,*disk*]...> <*size*[%]> <*mointpoint*> [-t *fstype*] [-l *label*] [-r *raidlevel*] [-f] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.2

DISKSWAP <*disk*[,*disk*]...> <*size*[%]> [-r *raidlevel*] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.2

DEPLOYMOUNT <*disk*[,*disk*]...> <*size*[%]> <*mointpoint*> [-t *fstype*] [-r *raidlevel*] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.5

DEPLOYSWAP <*disk*[,*disk*]...> <*size*[%]> [-r *raidlevel*] [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.5

BOOTLOADER <grub|lilo> [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.10

SYNCDIR <*path*> [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.6

EXCLUDESYNC <*path*> [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.7

EXCLUDEDEPLOY <*path*> [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...]
Section 5.10.8

MAC_ADDR <*hostname*> <*mac_address*>
Section 5.11.1

ETH_PRELOAD <*driver*[,*driver*]...>
Section 5.11.2

CONSOLECMD [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...] <*command*>
Section 5.14.2

POWERCMD <*function*> [-c *class*[,*class*]...] [-n *node*[,*node*]...] [-p *property*[,*property*]...] <*command*>
Section 5.14.2

SPECFORMAT <*spec_id*> <*format*> [NODE:/// | IP:///] [SPEC:///]
Section 5.15.1

B initramfs.conf Configuration Directives

TEMPLATE <*path*>
Use the specified initramfs template

MODULE <*module*>
Adds the module (and its dependencies) to the ramfs

NOMODPROBE
Does not automaticaly add any ethernet/disk drivers from modprobe.conf

SCSIMODS
Add disk (scsi_hostadapter) modules from modprobe.conf

PRELOAD <*module*>
Adds the module (and its dependencies) to the ramfs **before** any SCSI modules are loaded

RAMDISK_ROOT
Directs the initrd/ramfs to use the ram disk as the root filesystem

COPY <*path*> **[-r] [-n]**
Copies the given path to the initramfs
-r recursive copy
-n Do not copy library dependencies

OVERLAY <*directory*>
Overlays the given directory onto the ramfs

RAM_OVERLAY [*max_size*[**k**|**m**|**g**]]
Overlays a RAM disk that grow up to *max_size* onto the NFS root filesystem. Deprecates the need for LINKDIR/LINKFILE directives. Defaults to half of available RAM. **Note**: A patch is currently needed to get ideal behavior out of overlayfs.

NOMOUNT
Do not automount fomr /etc/fstab when using ramdisk-root mode

NODHCP
Directs the initrd/ramfs not to run a DHCP client

DHCP_INTERFACE <*interface*>
Run dhcp client over the specified interface. Multiple directives can be supplied to try each interface in turn. (default: all available interfaces)

DHCP_RETRIES *<number>*
Attempt to send *number* discovers before trying the next interface. A value of 0 will continue to retry indefinitely. (default: 5)

NFS_RETRIES *<number>*
Attempt to try to mount the NFS root filesystem ¡number¿ times before failing. (default: 5)

RETRY_REBOOT
Reboot the system if any of NFS, DHCP, or network check fails

BOND_INTERFACES *<interface>,<interface>...*
Bond the requested interfaces together. To use this option, you must have a modprobe.conf with the desired options set for the bonding driver. eg:
options bonding mode=balance-alb miimon=100
Be aware that some bonding modes require the switch to have the appropriate ports configured for 'etherchannel' or 'trunking'.

NETWORK_CHECK *<number>*
Before attempting to mount an NFS root, this option checks for network connectivity by attempting to ping the gateway address up to *number* times. This is useful for interfaces such as bonded NICs that may take some time before they are fully ready. A value of 0 disables the network check. (default: 5)

RESUME_FROM *<device>*
Attempt to resume from swsusp image on DEVICE

AUTOMOUNT
Directs the initrd/ramfs to mount any labeled filesystems. eg: A disk labeled '/var' will be mounted on /var

SWAPON
Directs the initrd/ramfs to enable any swap partitions.

SETUP_RAID
Directs the initrd/ramfs to attempt to assemble any existing RAID arrays

MOUNT_RW *<dir>*
Directs the initrd/ramfs to mount the given directory read-write

MOUNT_ALL_RW
Directs the initrd/ramfs to mount all directories read-write

INITRD
Creates an initrd image instead of initramfs

INITRD_SIZE *<kbytes>*
Specifies absolute size to use for the uncompressed initrd image. (**Note:** not necessary for initramfs images)

INITRD_BUFFER *<kbytes>*
Specifies the amount of free space to leave available after all files have been added to an initrd (**Note:** not necessary for initramfs images)

C System image revision control with git, HOWTO

First, a quick note about speed. Doing git commands on a repository the size of a oneSIS system image over NFS can be somewhat slow. If your NFS server is a regular linux box, it is highly recommended that you run all git commands directly on that box so that all access to the image is direct to local disk rather than over NFS.

Git is an extremely powerful revision control system. It is advisable to create and tinker with smaller repositories before jumping to a full system image. Once you have a system image and are ready to put it into git, you can follow this procedure.

Procedure 6 Prepare an image for use with git.
Git is included in most modern distributions, but there is still an extra script that is packaged with the git source that we need to use for managing a system image.

- **Download and install git**
  Directions can be found at http://git-scm.com/download

- **Download setgitperms.perl**
  From git source tree: https://raw.github.com/git/git/master/contrib/hooks/setgitperms.perl

- **Create a git repository in your system image**
  ```
  # cd /var/lib/oneSIS/images/oneSIS_image_dir
  # git init
  ```

- **Create some hooks to enable tracking permissions/ownership**
  ```
  # rm -f .git/hooks/*
  # cp setgitperms.perl .git/hooks
  ```
  Create `.git/hooks/pre-commit`
  ```
  #!/bin/sh
  SUBDIRECTORY_OK=1 .  git-sh-setup
  $GIT_DIR/hooks/setgitperms.perl -r
  ```
  Create both `.git/hooks/post-checkout` **and** `.git/hooks/post-merge`
  ```
  #!/bin/sh
  SUBDIRECTORY_OK=1 .  git-sh-setup
  $GIT_DIR/hooks/setgitperms.perl -w
  ```

- **Git does not track empty directories so make sure that none are empty**
  Put a .gitignore file in every empty directory
  # `find .  -name .git -prune -o -type d -empty -exec touch {}/.gitignore \;`

- **Git does not track device files so create a tar archive for them**
  # `find . \( -type b -o -type c -o -type p \) -exec tar rf dev/devs.tar {} \;`

- **Add all files into the git repository**
  This step may take some time. Be sure to add a useful comment to all commits.
  # `git add .`
  # `git commit`

Thats it! The system image is now under revision-control. It is now possible to detect any changes to the system image with a simple `git status`, commit changes with `git add` and `git commit`, and view the history with `git log`. Refer to git documentation for how to use branching, merging, and other git capabilities.

Note: When cloning a git repository, the .git/hooks directory is not cloned, so you will need to copy all the hooks over by hand and manually untar the device file tarball on any new clone.

Procedure 7 Cloning a system image repository.

- **Clone the repository**
  # `git clone oneSIS_image_dir cloned_image_dir`

- **Set up the hooks**
  # `rm -rf cloned_image_dir/.git/hooks`
  # `cp -a oneSIS_image_dir/.git/hooks cloned_image_dir/.git/`

- **Manually adjust permissions/ownership to match the repository**
  # `cd cloned_image_dir`
  # `.git/hooks/setgitperms.perl -w`

- **Unpack all device files**
  # `tar xvf dev/devs.tar`