

# **Constructing “One-Click” Virtual Appliances**

**Olaf Manczak**

Copyright © 2008 Sun Microsystems, Inc. All Rights Reserved.  
Use is subject to license terms.

## 1 Introduction

The root file system of a Live\* Virtual Appliance is constructed dynamically during an early boot stage. It is composed out of one or more Live\* disk images containing various software components and an optional copy-up disk image (device). All software images are immutable and the hypervisor virtualizes them as separate read-only disk devices. The writable copy-up image (device) provides persistent storage for mutations. It appears as a writable disk inside of the virtual machine. If no persistent copy-up image (device) is given, then the Live\* runtime will use a dynamically created memory based file system (tmpfs) instead. However, all mutations will be lost when the virtual machine reboots. At the early boot stage, the Live\* runtime mounts all the disk images individually, and uses file system namespace unification (union file system), to combine them into a single root file system. Also, the Live\* runtime generates the configuration files using so-called *generators* embedded in the Live\* disk images.

Each of the Live\* images contains two regular disk partitions with file systems (e.g. ext3) and a metadata section appended at the end of the image. The first partition contains all the software that the image adds to the root file system. The second partition contains generators. The *metadata* section, which is easily accessible both from the hypervisor host side and from within the virtual machine, contains a set of key-value pairs that describe the image name, version, etc. A separate document [1] contains the Live\* disk image specification and comprehensive instructions how to create Live\* disk images.

In this document we discuss in detail how the unified Live\* root file system is composed, and we give a comprehensive overview of dynamic file instantiation using the Live\* generators.

### 1.1 User input

To create a virtual machine the user has to provision host memory and processor resources dedicated to the VM, supply the Live\* software and copy-up disk images used for the construction root file system, and, optionally, identify all additional disk and swap images. Provisioning of the host resources for a Live\* VM does not differ from a traditional (non-Live\*) case.

We assume that the user will either supply a mutually compatible set of Live\* images with desired software components, or the user will select images from a repository. Newly initialized (empty) copy-up and swap images in standard sizes can be generated on demand, for example, by uncompressing pre-built images and stamping them with new UUIDs. Also, the user will have to provide a file with so-called *properties* – key-value pairs used by the Live\* generators embedded in the software images to instantiate configuration files at the early boot stage.

In addition, the user might supply a set of files (typically configuration files) that supplement or supersede files contained in the Live\* disk images (effectively a top read-only layer for the namespace unification), as well as, a set of generators to be used in addition or instead of the generators embedded in the images.

#### 1.1.1 Live\* launchpad

To simplify the resource provisioning, selection of the software disk images and definition of the configuration properties, we provide a *launchpad* – a simple graphical interface (GUI). Similarly to the Windows *Control Panel* or Mac OS X *System Properties*, the *launchpad*'s main panel allows the user select a specific sub-category. The *Virtual Hardware* panel allows the user to provision virtual hardware resources (such as cpu, memory, swap space, virtual machine name, etc.), and either create or assign an existing copy-up disk image.

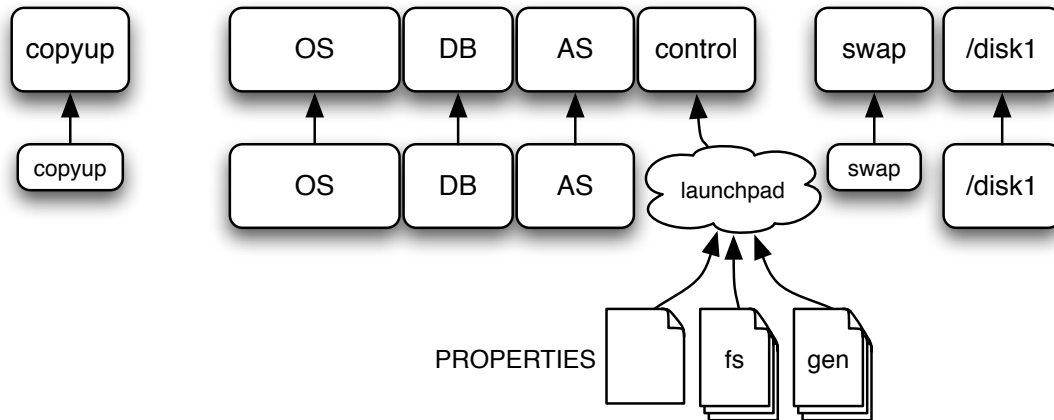
## Constructing "One-Click" Virtual Appliances

Using the Software panel the user can select software disk images that will be included in the composition of the virtual root file system: The user can also assign additional disk images that will be mounted within the virtual execution environment in a traditional way. The *Network*, *Automount* and *Nameservice Switch* panels convert configuration parameters supplied by the user into the corresponding configuration properties. Optionally, the user can supply a text file with additional configuration properties, as well as, a directory tree with additional configuration files and a directory tree with supplemental generators.



## 1.2 Control image

We need make the *properties*, the user supplied files and generators available to the Live\* runtime within the *guest* operating system. Also, we need to instruct the runtime about the namespace layering order. To do so we combine the *properties* file and the user supplied files and generators into an auxiliary read-only disk image – the *control image*.



The control image is built using the *host* tools, but it is interpreted in the *guest* environment. Therefore, it has to use one of the file system (or file archive) formats shared between the host and the guests. We decided to use a compressed *tar* image with the following canonical internal structure:

```
# tar -tjvf /tmp/launchpad/vml-control-mfZJnz.tbz2
drwxr-xr-x root/root      0 2009-01-15 12:30 gen/
-rw-r--r-- root/root    3663 2009-01-15 12:30 gen/PROPERTIES
drwxr-xr-x root/root      0 2008-11-14 03:56 fs/
-rw-r--r-- root/root     394 2008-11-14 03:56 fs/etc/auto.users
drwxr-xr-x demo/wheel    0 2008-11-14 03:56 fs/disk1/
drwxr-xr-x demo/wheel    0 2008-11-11 20:49 fs/disk2/
```

Obviously, one could use several other formats, for example, ISO9660 or VFAT file systems, or *cpio* archives.

The control image metadata section, appended to the archive contains the following four parameters: *name*, *rootset*, *searchorder* and *copyup*. For example:

```
# ./imagetool -M /tmp/launchpad/vml-control-mfZJnz.tbz2
name='control'
rootset='control:fedora8:java6u10'
copyup='copyup-256M'
searchorder='all'
```

The *name* parameter, which has to be unique for each of the images, defines the logical name of the image. By convention, *name* is also used by the Live\* runtime as a mount-point name. The *rootset* parameter lists logical (metadata) names of the images in order in which their namespaces are unionized. This list includes the name of the copy-up image and the name of the control image itself. The *copyup* parameter specifies the logical (metadata) name of the copy-up image. Finally, the *searchorder* parameter instructs the Live\* runtime which of the block devices to probe and in what.

### 1.3 Runtime discovery

In a traditional case, the kernel uses content of the *initrd* archive (a compressed `cpio` archive) to initialize the initial root file system. The initial root file system does not have any backing storage and resides purely in memory (pinned in the kernel file system cache). Once the file system is ready, the kernel transfers control to the `init` program (typically a shell script). The role of the `init` program (script) is to discover and mount the real (persistent) root device, then mount `/dev`, `/sys` and `/proc`, and then trampoline to the actual `init` program from the real root file system.

The real root device is defined by the root kernel command line parameter passed by the boot loader. For example, if `grub.conf` (or `menu.lst`) contains the following line:

```
kernel /boot/vmlinuz root=UUID=e5b78358-2af1-491c-ae8b-79380cb7264e ro
```

then, the *initrd* will search among all the block devices (i.e. all disks and disk partitions) discovered by kernel and visible under `/sys/block/...` for one which has a superblock with a matching UUID.

The only change introduced by the Live\* runtime to this procedure is how the root file system is discovered and mounted. The Live\* runtime ignores the root kernel command line parameter.

Instead, the Live\* runtime probes all disk devices discovered by kernel and listed under `/sys/block/...` trying to identify the devices corresponding to the Live\* control image, the copy-up image, and the Live\* software images, as well as, devices which contain Live\* images in its top directory.

#### 1.3.1 Control image discovery

Discovery of the control image is fairly simple. The control image is expected to be a compressed `tar` archive with a Live\* metadata section appended at the end. The metadata section of the control image should always contain the `name`, `rootset`, `searchorder` and `copyup` metadata parameters. To make the archive content available, the Live\* runtime creates a memory-based file system (*tmpfs*) mounted as `/Volumes/<name>` (e.g. `/Volumes/control`) and unpacks the archive in this directory.

#### 1.3.2 Discovery of software images

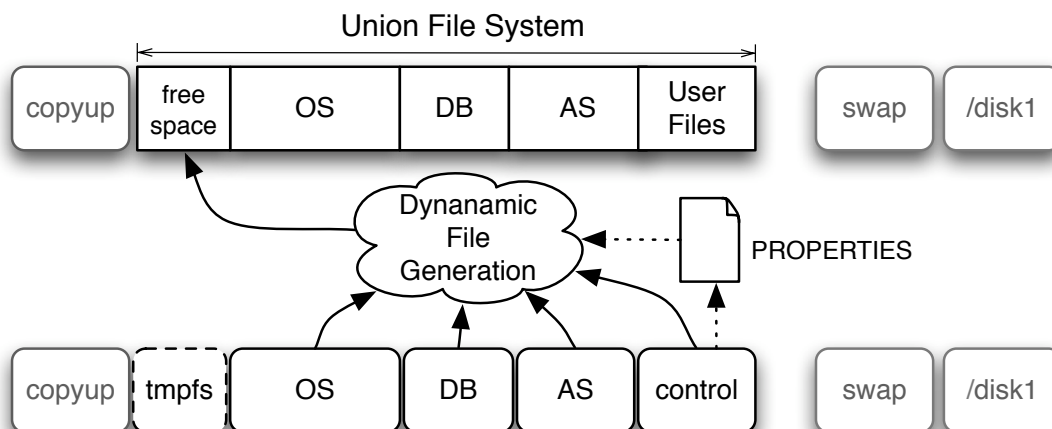
Similarly, when Live\* runtime probes software images it anticipates two `ext3` partitions and a valid Live\* metadata section with the `name` parameter at the end. For each of the images that pass this test, the runtime retrieves the logical name of the image from the metadata section, and attempts to mount (in read-only mode) both the software partition and the partition with generators as, respectively, `/Volumes/<name>/fs` and `/Volumes/<name>/gen`.

## 1.4 Dynamic instantiation of structured data

After all the images have been mounted the file system tree below /Volumes may, for example look like this:

```
# ls -ld /Volumes/*
drwxr-xr-x  2 root root  4096 Nov 15  2008 /Volumes/control
drwxr-xr-x  2 root root  4096 Nov 15  2008 /Volumes/copyup-vm1
drwxr-xr-x  2 root root  4096 Nov 11  2008 /Volumes/fedora8
drwxr-xr-x  2 root root  4096 Oct 21  2008 /Volumes/java6u10
# ls -ld /Volumes/*/*
drwxr-xr-x  2 root root  4096 Nov 15  2008 /Volumes/control/fs
drwxr-xr-x  2 root root  4096 Nov 15  2008 /Volumes/control/gen
drwxr-xr-x  2 root root  4096 Nov 11  2008 /Volumes/fedora8/fs
drwxr-xr-x  2 root root  4096 Nov 11  2008 /Volumes/fedora8/gen
drwxr-xr-x  2 root root  4096 Oct 21  2008 /Volumes/java6u10/fs
drwxr-xr-x  2 root root  4096 Oct 21  2008 /Volumes/java6u10/gen
# ls -l /Volumes/*/gen/MANIFEST
-rw-r--r--  2 root root   12 Nov 15  2008 /Volumes/control/gen/MANIFEST
-rw-r--r--  2 root root 1246 Nov 11  2008 /Volumes/fedora8/gen/MANIFEST
-rw-r--r--  2 root root   69 Oct 21  2008 /Volumes/java6u10/gen/MANIFEST
# ls -l /Volumes/*/gen/PROPERTIES
-rw-r--r--  2 root root 3663 Nov 15  2008 /Volumes/control/gen/PROPERTIES
```

One can use the *union file system* to combine (overlay) namespaces of all the `fs` file systems into one virtual root file system integrating all software components provided by the individual images. However, this technique applies only to unstructured data. To instantiate structured data (primarily configuration files), the Live\* runtime executes the generators supplied with the software images and by the user through the control image. To do so, the runtime creates a union file system combining the all the `fs` (software) file systems in order defined by the `rootset` metadata parameter, but we use an empty writable memory-based copy-up file system (*tmpfs*) mounted at `/Volumes/config` as the copy-up space. This way the generators will see the complete, though initially un-configured, root file system, and they can take advantage of any tools and commands present in the root file system. In addition, after the generators are executed all the newly created (or mutated) files will be stored in the copy-up space.



To make the execution of generators deterministic, the Live\* runtime iterates through the `gen` file systems in the order defined by the `rootset` metadata parameter, and it executes the generators

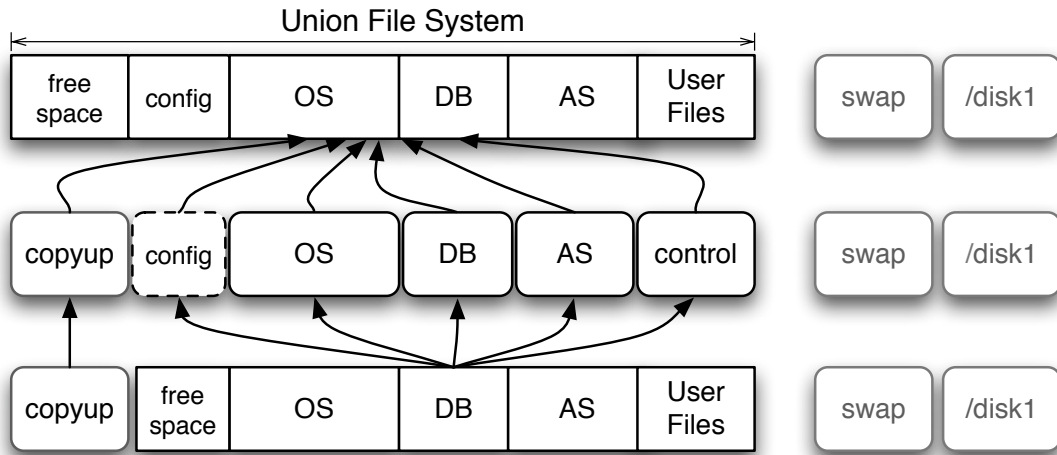
sequentially in the order specified in the file `/Volumes/<name>/gen/MANIFEST`. During execution the generators have access to the entire set of properties given in the control image.

### 1.5 Final composition of the virtual root file system

After all the generators have been executed have been created, the Live\* runtime removes (destroys) the virtual *unionfs* layer. The generated configuration files remain in the copy-up space. For example:

```
# find /Volumes/config
/Volumes/config/etc
/Volumes/config/etc/hosts
/Volumes/config/etc/nsswitch.conf
/Volumes/config/etc/sysconfig
/Volumes/config/sysconfig/network
/Volumes/config/sysconfig/network-scripts
/Volumes/config/sysconfig/network-scripts/ifcfg-eth0
...
```

The Live\* runtime remounts the `/Volumes/config` in read-only mode, and then it creates a *union* file system once again. Though, this time, it combines namespaces of the persistent copy-up image, the generated structured data in `/Volumes/config`, and all the *fs* directory trees. The resulting file system contains not only all the software components but also dynamically generated system configuration, as well as, all the prior mutations. Even though the generated system configuration files are not persistent – they are dynamically instantiated upon every boot – any mutations to these files are kept in the copy-up space, which is applied as the top layer in namespace unification. Hence, any modifications to the generated files will persist across reboots.



This step completes the Live\* instrumentation of *initrd*. We have only substituted the conventional traditional discovery and mounting of the root device with a bit more elaborate process of composition the root file system from multiple images and dynamically generated data. Yet, this substitution is entirely transparent to the operating system and application software included in the images. In particular, no changes are required to the operating system startup (`/etc/init.d` scripts), since the configuration files are generated before control is transferred to the true *init* program.