**5.16. Building a Tiny System[¶](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html" \l "building-a-tiny-system" \o "Permalink)**

Very small distributions have some significant advantages such as requiring less on-die or in-package memory (cheaper), better performance through efficient cache usage, lower power requirements due to less memory, faster boot times, and reduced development overhead. Some real-world examples where a very small distribution gives you distinct advantages are digital cameras, medical devices, and small headless systems.

This section presents information that shows you how you can trim your distribution to even smaller sizes than the poky-tinydistribution, which is around 5 Mbytes, that can be built out-of-the-box using the Yocto Project.

**5.16.1. Overview**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#tiny-system-overview)

The following list presents the overall steps you need to consider and perform to create distributions with smaller root filesystems, achieve faster boot times, maintain your critical functionality, and avoid initial RAM disks:

* [Determine your goals and guiding principles.](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#goals-and-guiding-principles)
* [Understand what contributes to your image size.](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#understand-what-gives-your-image-size)
* [Reduce the size of the root filesystem.](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#trim-the-root-filesystem)
* [Reduce the size of the kernel.](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#trim-the-kernel)
* [Eliminate packaging requirements.](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#remove-package-management-requirements)
* [Look for other ways to minimize size.](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#look-for-other-ways-to-minimize-size)
* [Iterate on the process.](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#iterate-on-the-process)

**5.16.2. Goals and Guiding Principles**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#goals-and-guiding-principles)

Before you can reach your destination, you need to know where you are going. Here is an example list that you can use as a guide when creating very small distributions:

* Determine how much space you need (e.g. a kernel that is 1 Mbyte or less and a root filesystem that is 3 Mbytes or less).
* Find the areas that are currently taking 90% of the space and concentrate on reducing those areas.
* Do not create any difficult "hacks" to achieve your goals.
* Leverage the device-specific options.
* Work in a separate layer so that you keep changes isolated. For information on how to create layers, see the "[Understanding and Creating Layers](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#understanding-and-creating-layers)" section.

**5.16.3. Understand What Contributes to Your Image Size**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#understand-what-gives-your-image-size)

It is easiest to have something to start with when creating your own distribution. You can use the Yocto Project out-of-the-box to create the poky-tiny distribution. Ultimately, you will want to make changes in your own distribution that are likely modeled afterpoky-tiny.

**Note**

To use poky-tiny in your build, set the [DISTRO](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#var-DISTRO) variable in your local.conf file to "poky-tiny" as described in the "[Creating Your Own Distribution](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#creating-your-own-distribution)" section.

Understanding some memory concepts will help you reduce the system size. Memory consists of static, dynamic, and temporary memory. Static memory is the TEXT (code), DATA (initialized data in the code), and BSS (uninitialized data) sections. Dynamic memory represents memory that is allocated at runtime: stacks, hash tables, and so forth. Temporary memory is recovered after the boot process. This memory consists of memory used for decompressing the kernel and for the \_\_init\_\_ functions.

To help you see where you currently are with kernel and root filesystem sizes, you can use two tools found in the [Source Directory](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#source-directory)in the scripts/tiny/ directory:

* ksize.py: Reports component sizes for the kernel build objects.
* dirsize.py: Reports component sizes for the root filesystem.

This next tool and command help you organize configuration fragments and view file dependencies in a human-readable form:

* merge\_config.sh: Helps you manage configuration files and fragments within the kernel. With this tool, you can merge individual configuration fragments together. The tool allows you to make overrides and warns you of any missing configuration options. The tool is ideal for allowing you to iterate on configurations, create minimal configurations, and create configuration files for different machines without having to duplicate your process.

The merge\_config.sh script is part of the Linux Yocto kernel Git repositories (i.e. linux-yocto-3.14, linux-yocto-3.10,linux-yocto-3.8, and so forth) in the scripts/kconfig directory.

For more information on configuration fragments, see the "[Generating Configuration Files](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#generating-configuration-files)" section of the Yocto Project Linux Kernel Development Manual and the "[Creating Configuration Fragments](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#creating-config-fragments)" section, which is in this manual.

* bitbake -u taskexp -g *bitbake\_target*: Using the BitBake command with these options brings up a Dependency Explorer from which you can view file dependencies. Understanding these dependencies allows you to make informed decisions when cutting out various pieces of the kernel and root filesystem.

**5.16.4. Trim the Root Filesystem**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#trim-the-root-filesystem)

The root filesystem is made up of packages for booting, libraries, and applications. To change things, you can configure how the packaging happens, which changes the way you build them. You can also modify the filesystem itself or select a different filesystem.

First, find out what is hogging your root filesystem by running the dirsize.py script from your root directory:

$ cd *root-directory-of-image*

$ dirsize.py 100000 > dirsize-100k.log

$ cat dirsize-100k.log

You can apply a filter to the script to ignore files under a certain size. The previous example filters out any files below 100 Kbytes. The sizes reported by the tool are uncompressed, and thus will be smaller by a relatively constant factor in a compressed root filesystem. When you examine your log file, you can focus on areas of the root filesystem that take up large amounts of memory.

You need to be sure that what you eliminate does not cripple the functionality you need. One way to see how packages relate to each other is by using the Dependency Explorer UI with the BitBake command:

$ cd *image-directory*

$ bitbake -u taskexp -g *image*

Use the interface to select potential packages you wish to eliminate and see their dependency relationships.

When deciding how to reduce the size, get rid of packages that result in minimal impact on the feature set. For example, you might not need a VGA display. Or, you might be able to get by with devtmpfs and mdev instead of udev.

Use your local.conf file to make changes. For example, to eliminate udev and glib, set the following in the local configuration file:

VIRTUAL-RUNTIME\_dev\_manager = ""

Finally, you should consider exactly the type of root filesystem you need to meet your needs while also reducing its size. For example, consider cramfs, squashfs, ubifs, ext2, or an initramfs using initramfs. Be aware that ext3 requires a 1 Mbyte journal. If you are okay with running read-only, you do not need this journal.

**Note**

After each round of elimination, you need to rebuild your system and then use the tools to see the effects of your reductions.

**5.16.5. Trim the Kernel**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#trim-the-kernel)

The kernel is built by including policies for hardware-independent aspects. What subsystems do you enable? For what architecture are you building? Which drivers do you build by default?

**Note**

You can modify the kernel source if you want to help with boot time.

Run the ksize.py script from the top-level Linux build directory to get an idea of what is making up the kernel:

$ cd *top-level-linux-build-directory*

$ ksize.py > ksize.log

$ cat ksize.log

When you examine the log, you will see how much space is taken up with the built-in .o files for drivers, networking, core kernel files, filesystem, sound, and so forth. The sizes reported by the tool are uncompressed, and thus will be smaller by a relatively constant factor in a compressed kernel image. Look to reduce the areas that are large and taking up around the "90% rule."

To examine, or drill down, into any particular area, use the -d option with the script:

$ ksize.py -d > ksize.log

Using this option breaks out the individual file information for each area of the kernel (e.g. drivers, networking, and so forth).

Use your log file to see what you can eliminate from the kernel based on features you can let go. For example, if you are not going to need sound, you do not need any drivers that support sound.

After figuring out what to eliminate, you need to reconfigure the kernel to reflect those changes during the next build. You could runmenuconfig and make all your changes at once. However, that makes it difficult to see the effects of your individual eliminations and also makes it difficult to replicate the changes for perhaps another target device. A better method is to start with no configurations using allnoconfig, create configuration fragments for individual changes, and then manage the fragments into a single configuration file using merge\_config.sh. The tool makes it easy for you to iterate using the configuration change and build cycle.

Each time you make configuration changes, you need to rebuild the kernel and check to see what impact your changes had on the overall size.

**5.16.6. Remove Package Management Requirements**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#remove-package-management-requirements)

Packaging requirements add size to the image. One way to reduce the size of the image is to remove all the packaging requirements from the image. This reduction includes both removing the package manager and its unique dependencies as well as removing the package management data itself.

To eliminate all the packaging requirements for an image, be sure that "package-management" is not part of your[IMAGE\_FEATURES](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#var-IMAGE_FEATURES) statement for the image. When you remove this feature, you are removing the package manager as well as its dependencies from the root filesystem.

**5.16.7. Look for Other Ways to Minimize Size**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#look-for-other-ways-to-minimize-size)

Depending on your particular circumstances, other areas that you can trim likely exist. The key to finding these areas is through tools and methods described here combined with experimentation and iteration. Here are a couple of areas to experiment with:

* glibc: In general, follow this process:
  1. Remove glibc features from [DISTRO\_FEATURES](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#var-DISTRO_FEATURES) that you think you do not need.
  2. Build your distribution.
  3. If the build fails due to missing symbols in a package, determine if you can reconfigure the package to not need those features. For example, change the configuration to not support wide character support as is done for ncurses. Or, if support for those characters is needed, determine what glibc features provide the support and restore the configuration.
  4. Rebuild and repeat the process.
* busybox: For BusyBox, use a process similar as described for glibc. A difference is you will need to boot the resulting system to see if you are able to do everything you expect from the running system. You need to be sure to integrate configuration fragments into Busybox because BusyBox handles its own core features and then allows you to add configuration fragments on top.

**5.16.8. Iterate on the Process**[**¶**](http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html#iterate-on-the-process)

If you have not reached your goals on system size, you need to iterate on the process. The process is the same. Use the tools and see just what is taking up 90% of the root filesystem and the kernel. Decide what you can eliminate without limiting your device beyond what you need.

Depending on your system, a good place to look might be Busybox, which provides a stripped down version of Unix tools in a single, executable file. You might be able to drop virtual terminal services or perhaps ipv6.