BitRock InstallBuilder
User Guide 4.0

Release 4.0.0 2006.01.03

http://www.bitrock.com

Overview

Welcome to BitRock InstallBuilder Multiplatform, the smart way to distribute your applications.

BitRock InstallBuilder allows you to create easy to use multiplatform installers that can be run in GUI, text and unattended modes. Having the right installer for your product will help you:

- **Sell your software**. A complicated, error-prone installation process can hinder your sales. BitRock installers "just work" providing a great end-user experience that helps sell your programs.
- **Reduce development time**. BitRock InstallBuilder simplifies the process of creating installation packages so you can focus on developing your product - not building installers.
- **Lower support costs**. BitRock installers simplify and automate many aspects of the installation process, reducing support costs and end-user frustration.

Features

- **Multiplatform Support** : BitRock installers are native binaries that can run on Windows 98, ME, 2000, XP, 2003, Mac OS X, FreeBSD, Solaris (Intel & Sparc), AIX, HP-UX, IRIX, and Linux (Intel x86/x64, s390& PPC). Contact us for legacy Windows 95 support and additional platform support such as Tru64, and operating systems running Itanium IA64
- **Desktop Integration** : BitRock installers provide native look and feel and desktop integration for Windows, KDE and Gnome.
- **RPM Integration** : BitRock installers can register your software with the RPM package database, combining ease of use with the powerful RPM package management system.
- **RPM generation (beta)** : In addition to creating native executables that can register with the RPM subsystem, BitRock InstallBuilder can generate RPM packages that can be installed using the native rpm package management tool.
- **Optimized** : BitRock installers are optimized in size and speed and do not require a self-extraction step, reducing download, startup and installation time.
- **No External Dependencies** : BitRock installers are single-file, self-contained native executables with no external dependencies and minimal overhead. Unlike competing products, all BitRock installers are truly native code and do not require bundling a Java Runtime Environment.
- **Ease of Use** : BitRock installers provide an intuitive and easy to use interface on all platforms, even for end users without previous Linux experience.
- **Ease of Development** : BitRock InstallBuilder includes an easy to learn, easy to use GUI environment. Design, build and test installers with the click of a button.
- **Time Saving Functionality** : For advanced users, a friendly XML project format supports source control integration, collaborative development and customizing projects both by hand and using external scripts. A command line interface allows you to automate and integrate the building process. QuickBuild functionality allows you to update installers in a few seconds, without having to repack the entire application.
- **Built-in actions** : BitRock InstallBuilder provides convenient built-in actions for commonly required installation functionality such as autodetecting a Java(tm) Runtime, changing file permissions and ownership, substituting text in a file, adding environment variables, adding directories to the path, creating symbolic links, changing the Windows registry, launching external scripts and so on.
- **Crossplatform Build Support** : The installer builder tool can run on Windows, Mac OS X, Solaris, HP-UX, AIX, FreeBSD, IRIX, and Linux (Intel x86/x64, s390, PPC) and generate installers for all target platforms from a single project file. Create all your installers from a single build environment!
- **Customization** : BitRock installers can be customized in a variety of ways, both graphically and in functionality.
- **Multiple Installation modes** : BitRock installers provide: several GUI modes with native look-and-feel, for installation in a variety of desktop environments, a text-based installation mode, for console-

based and remote installations, and a silent/unattended install mode which can be used for integration in shell scripts for automated deployment.

- **Uninstall Functionality** : An uninstall program is created as part of every installation, allowing users to easily uninstall the software. As the installer, it can be run in a variety of modes. On Windows, uninstall functionality can also be accessed from the Add/Remove Program entry in the Control Panel.
- **Startup Failure Detection** : BitRock installers will automatically detect the best installation mode available. Users also have the option to manually select a mode.
- **Language and Platform Independent** : BitRock installers can install applications written in any language, including: Java, PHP, Perl, Python, C/C++ and .NET/Mono.
- **Multiple Language Support** : BitRock installers support a variety of installation languages, including English, German, Spanish, Italian, French, Portuguese, Traditional Chinese, Dutch, Polish, Valencian, Catalan, Estonian, Slovenian, Romanian, Hungarian and Welsh. You can specify a default language or let the user decide. Please contact us if you require additional language support.

Coming soon: Support for Debian package management system.

What's new on InstallBuilder 4.0

- **Support for Qt® GUI Frontend** : The InstallBuilder for Qt family of products provides a new GUI installation mode using the Qt crossplatform toolkit, enhancing the end-user experience on Linux systems. Initially available for Windows, Linux and Mac OS X, we will be expanding support for additional platforms such as Solaris and HP-UX on upcoming releases.
- **Support for multiple parameters per page** : It is now possible to ask for multiple parameters, like username and passwords, in the same installer screen. These helps simplify the installation process for end-users.
- **Support for RPM generation (beta)** : InstallBuilder 4.0 allows the generation of RPM packages on Linux platforms using the 'rpm' target platform. Those RPMs are self-contained and have no dependencies.
- **Support for Linux 64 bits** : This release adds support for Linux 64-bit platforms natively. Current 32-bit installers also work on 64-bit Linux platforms with compatibility libraries installed, which is the default in most distributions.

Specifications

Supported Platforms

BitRock installers run in most Linux platforms and have been extensively tested in:

- Windows 98, ME, 2000, XP, 2003
- Mac OS X 10.2, 10.3, 10.4
- Solaris Sparc 2.6, 7, 8, 9, 10
- Solaris Intel 8, 9, 10
- AIX 4.3, 5.x and later
- HP-UX 11 and later
- IRIX 6.5
- FreeBSD 5.x. 4.x
- Ubuntu Linux PPC
- Red Hat 8.x, 9.x series
- Red Hat Enteprise Linux 3, 4
- Fedora Core 1, 2, 3, 4
- Suse 9.x series
- Suse Enterprise Server 8, 9, 10

as well as in different versions of Gentoo, Mandrake and Debian.

Our goal is to build installers that work seamlessly in the greatest number of operating system versions. Please let us know what your experience is with your particular operating system.

Authoring Environment Requirements (Windows)

- Windows 98, ME, 2000, XP, 2003 (contact us for Windows 95 support)
- 32Mb of free RAM
- Minimum of 640 x 480 screen resolution

Authoring Environment Requirements (Mac OS X)

- Mac OS X 10.2, 10.3, 10.4 (PPC or Intel)
- 64Mb of free RAM
- Minimum of 640 x 480 screen resolution

Authoring Environment Requirements (Unix)

- One of the following
- Solaris Sparc 2.6, 7, 8, 9, 10
- Solaris Intel 8, 9, 10
- HP-UX 11 or later
- IBM AIX 4.3 or later (including 5.x)
- IRIX SGI 6.5 or later

- FreeBSD 5.x, 4.x
- 32Mb of free RAM
- Minimum of 640 x 480 screen resolution

BitRock InstallBuilder can run in both GUI and command line modes. For GUI mode an X-Window installation must be present.

Authoring Environment Requirements (Linux)

- One of the following
- x86 Linux system.
- x64 Linux system.
- PPC Linux system.
- s390 Linux system.
- For x86, a glibc 2.2 distribution such as Red Hat 8.0 or later. For PPC, a glibc 2.3 distribution or later. These configurations cover virtually all Linux distribution currently in use. InstallBuilder may run in older systems but those configurations are not supported. The generated installers do not have those requirements (see below).
- Minimum of 640 x 480 screen resolution.

BitRock InstallBuilder can run in both GUI and command line modes. For GUI mode an X-Window installation must be present (X-Window is installed by default in most Linux distributions).

Generated Installer Requirements (Windows)

- Windows 98, ME, 2000, XP, 2003 (contact us for Windows 95 support)
- 16Mb of free RAM
- Minimum of 640 x 480 screen resolution

Generated Installer Requirements (Mac OS X)

- Mac OS X 10.2, 10.3, 10.4 (PPC or Intel)
- 32Mb of free RAM
- Minimum of 640 x 480 screen resolution.

Generated Installer Requirements (Unix)

- One of the following
- Solaris Sparc 2.6, 7, 8, 9, 10
- Solaris Intel 8, 9, 10
- HP-UX 11 or later
- IBM AIX 4.3 or later (including 5.x)
- IRIX SGI 6.5 or later
- FreeBSD 5.x, 4.x
- 16Mb of free RAM
- Minimum of 640 x 480 screen resolution.

Generated Installer Requirements (Linux)

- One of the following
- x86 Linux system.
- x64 Linux system.
- PPC Linux system.
- s390 Linux system.
- On x86, the minimum requirement is a glibc2-based Linux distribution. That includes most distributions released on or after the year 2000. Please contact us if you require support for older Linux versions.
- Minimum of 640 x 480 screen resolution.

X-Window installation mode

This mode requires an X-Window installation present in the system (otherwise an alternate text mode will be used).

GTK installation mode

This mode, supported on FreeBSD and Linux, requires GTK2 libraries installed in the system (otherwise an alternate X-Window GUI mode will be used). These libraries are installed by default in most FreeBSD and Linux installations.

How to Register

You can download a fully functional evaluation version of BitRock InstallBuilder from **www.bitrock.com**. It will add a reminder message to each installer ("Created with an evaluation version of BitRock InstallBuilder") and can only be used for evaluation purposes.

You can purchase a license for BitRock InstallBuilder online following the instructions you will find at **www.bitrock.com**. Once you do so, you will receive a license file. To register the product, just copy the file as `license.xml` to the directory where BitRock InstallBuilder was installed.

Alternatively you can install the license through the GUI interface. From the main application menu select "License", then "Register License", and a window will appear where you can enter the license file location.

Visit **www.bitrock.com** for further information on the License Agreement.

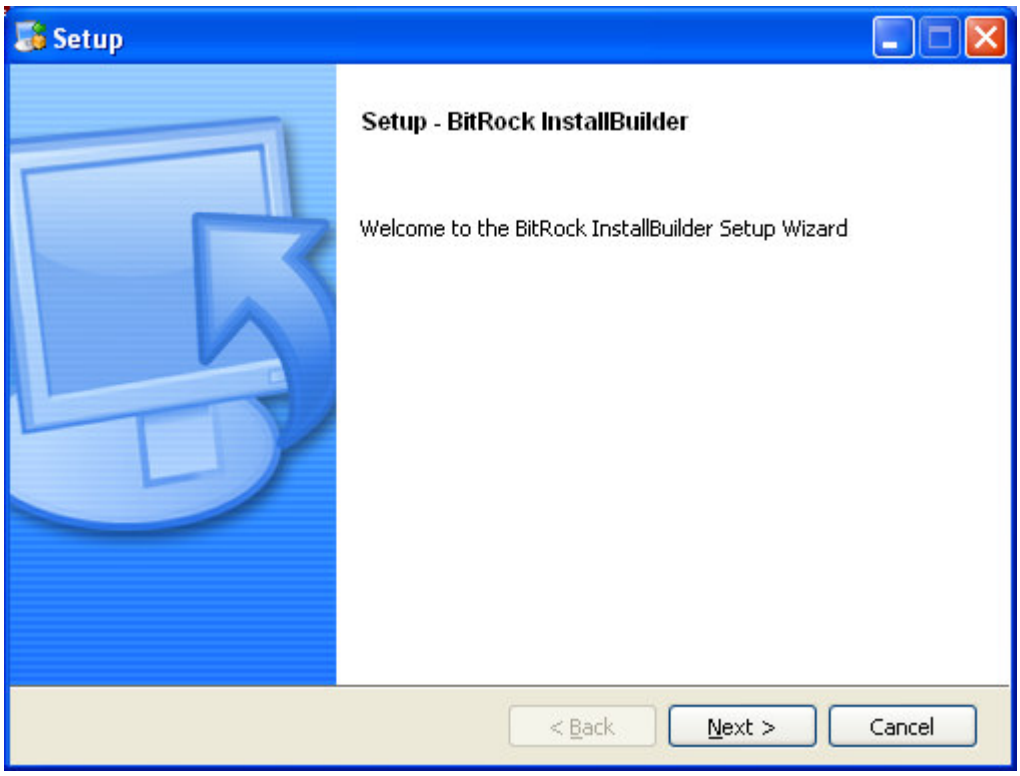
Installing BitRock InstallBuilder

Installing on Windows

You can download the BitRock InstallBuilder binary from the BitRock website. It should have a name similar to `installbuilder-professional-3.8-windows-installer.exe` . You can start the application by double-clicking on the downloaded file.

You will be greeted by the Welcome screen shown in Figure 1 .

Figure 1 : Windows Welcome Screen



Pressing Next will take you to the License Agreement page, shown in Figure 2. You need to accept the agreement to continue the installation. The next step is to select the installation directory (Figure 3). The default value is `C:\Program Files\BitRock InstallBuilder\`

Figure 2 : Windows License Agreement

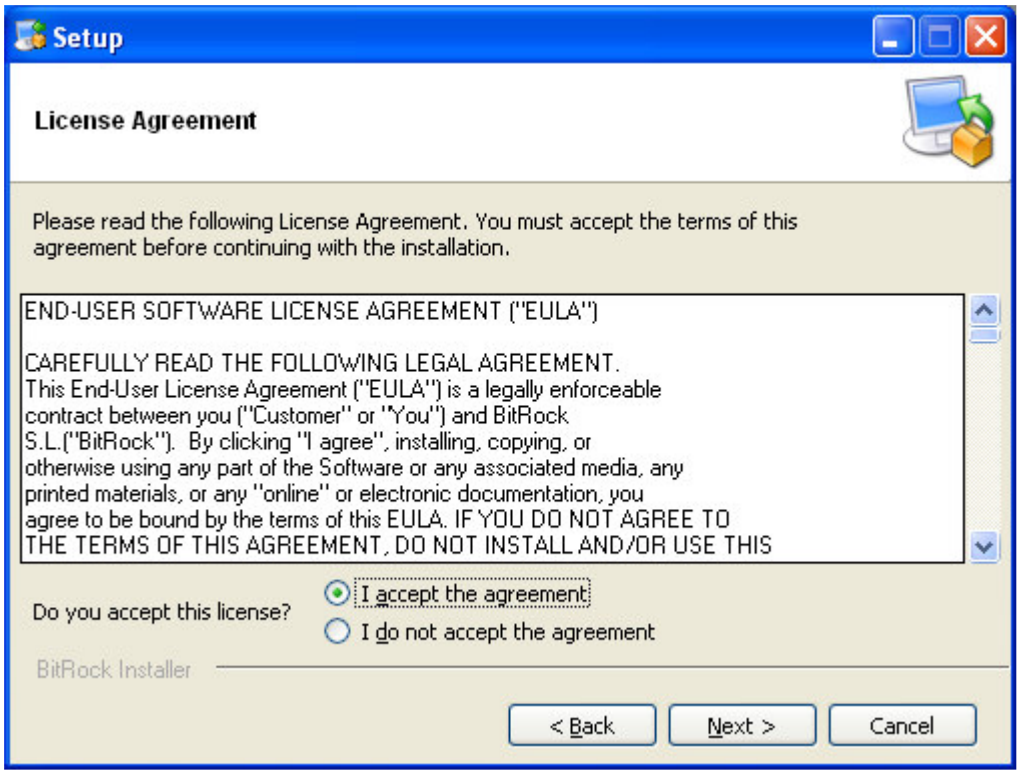
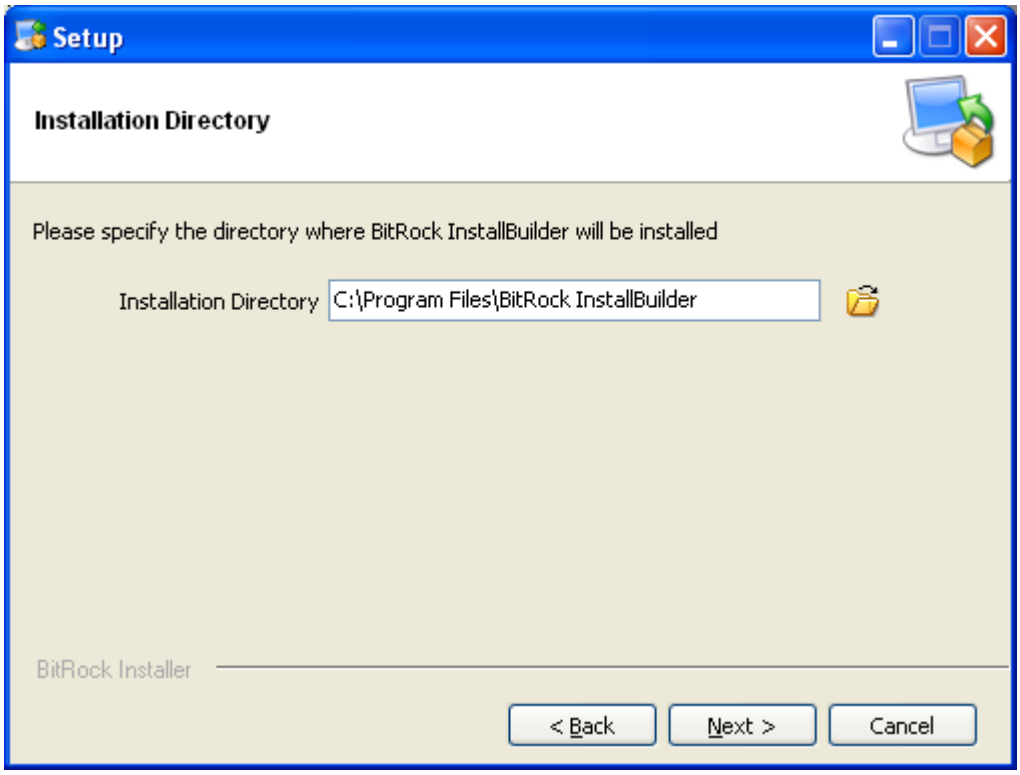


Figure 3 : Windows Select Installation Directory



The rest of this guide assumes you installed BitRock InstallBuilder in `C:\Program Files\BitRock InstallBuilder\`

You are now ready to start the installation (Figure 4), which will take place once you press Next (Figure 5). When the installation completes, you will see the installation completed page shown in Figure 6. You may choose to view the README file at this point.

If you found a problem and could not complete the installation, please refer to the Troubleshooting section or contact us at support@bitrock.com. Please refer to the Support section for details on what information you should include with your request.

Figure 4 : Windows Ready To Install

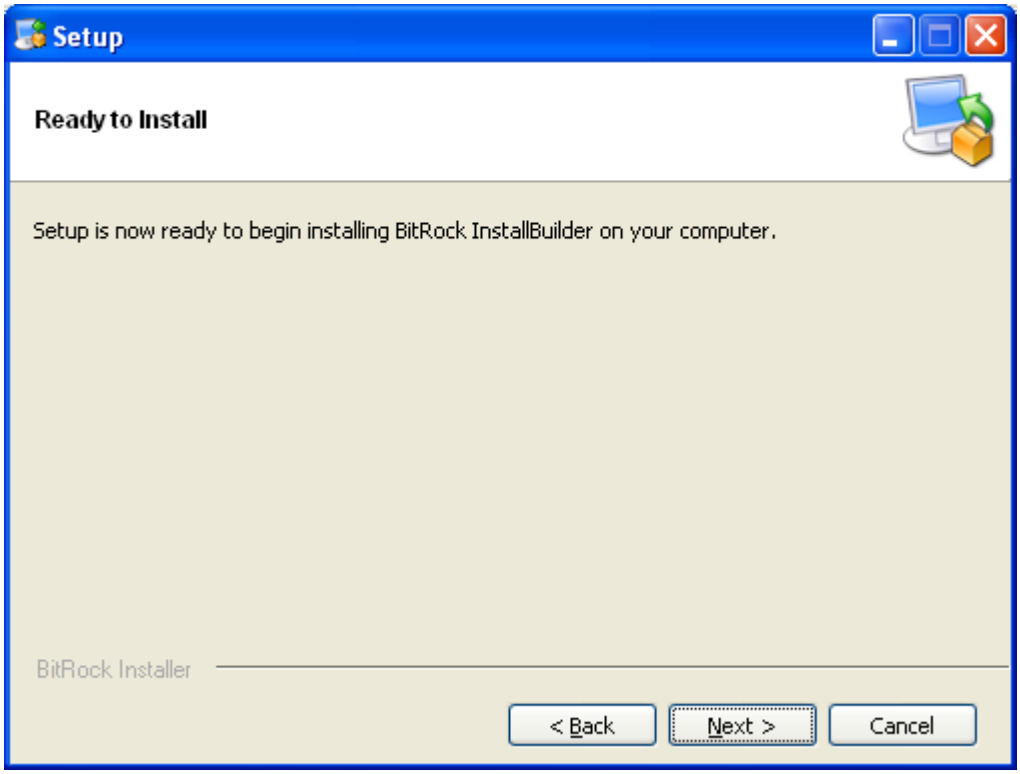


Figure 5 : Windows Installation Under Way

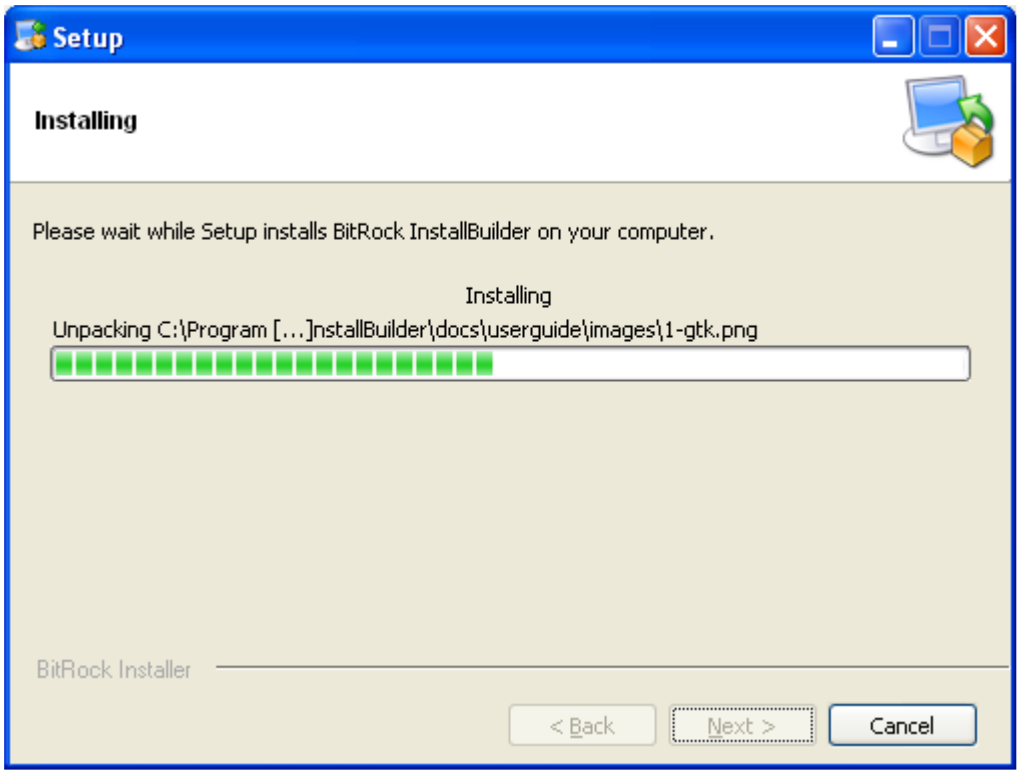
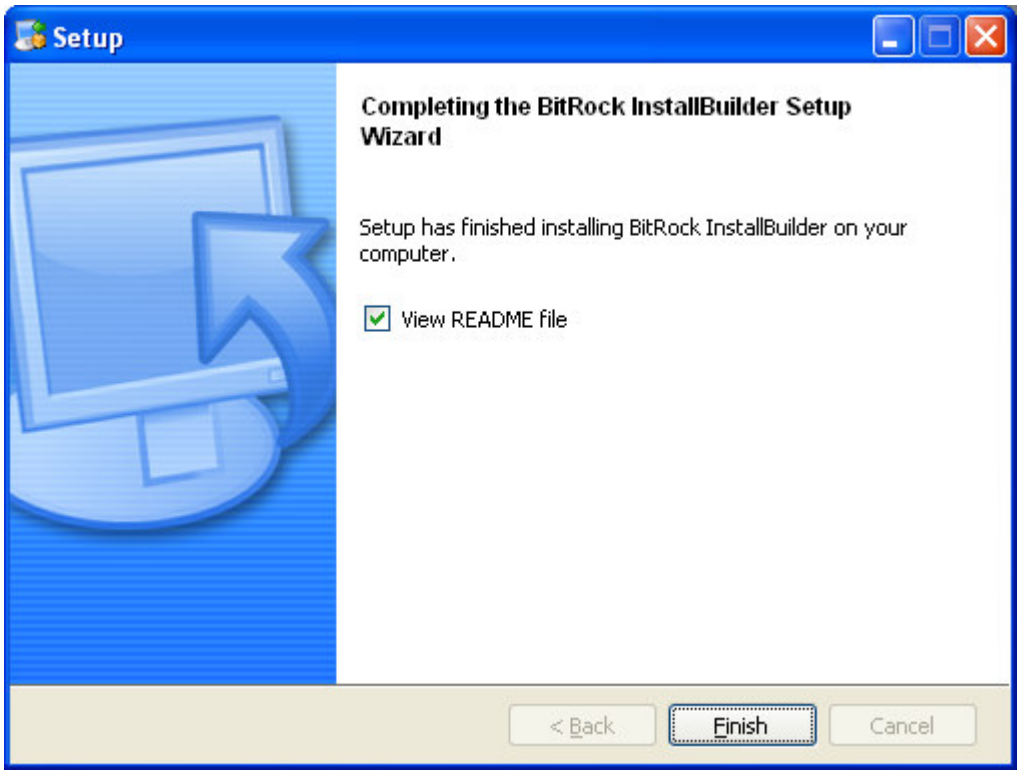


Figure 6 : Windows Installation Completed



Installing on Unix

The process for installing on Linux, FreeBSD, AIX, HP-UX, IRIX, and Solaris is identical. The rest of this section assumes you are running Linux.

You can download the BitRock InstallBuilder binary from the BitRock website. It should have a name similar to `installbuilder-professional-3.8-linux-installer.bin` . Make sure it has read and executable permissions by right clicking in the file, selecting "Properties" and then setting the appropriate permissions. Alternatively you can issue the following shell command.

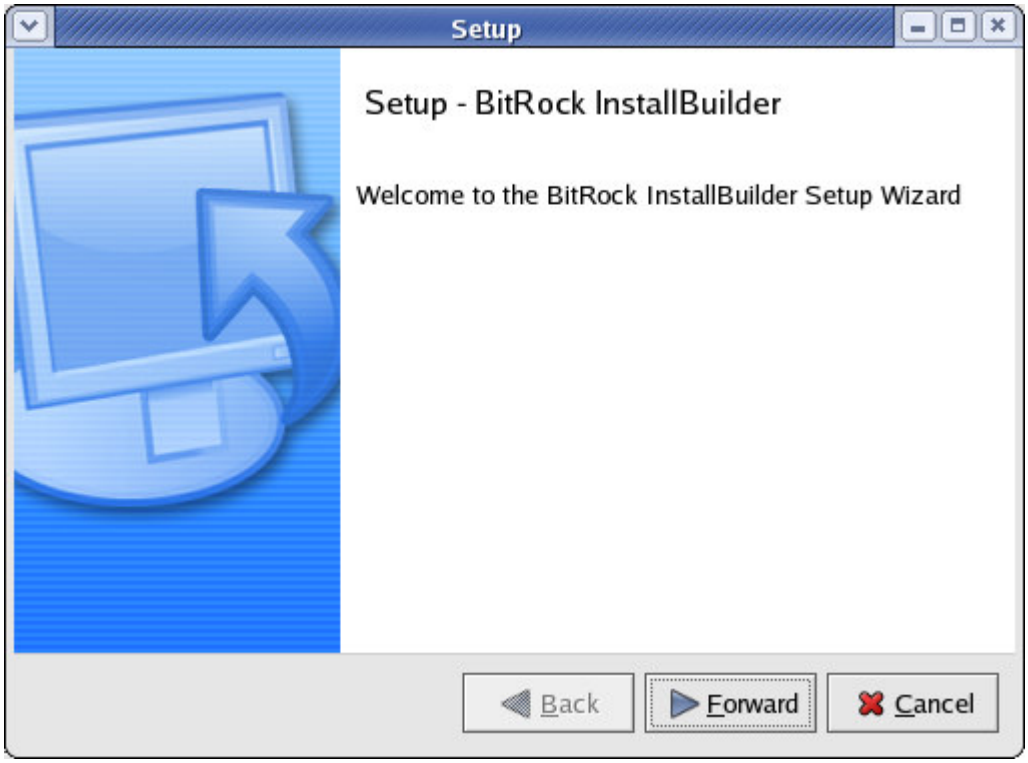
```
$ chmod 755 installbuilder-professional-3.8-linux-installer.bin
```

You can now start the installation by double-clicking on the file from your Desktop environment or by invoking it directly from the command line with:

```
$ ./installbuilder-professional-3.8-linux-installer.bin
```

You will be greeted by the Welcome screen shown in Figure 7 .

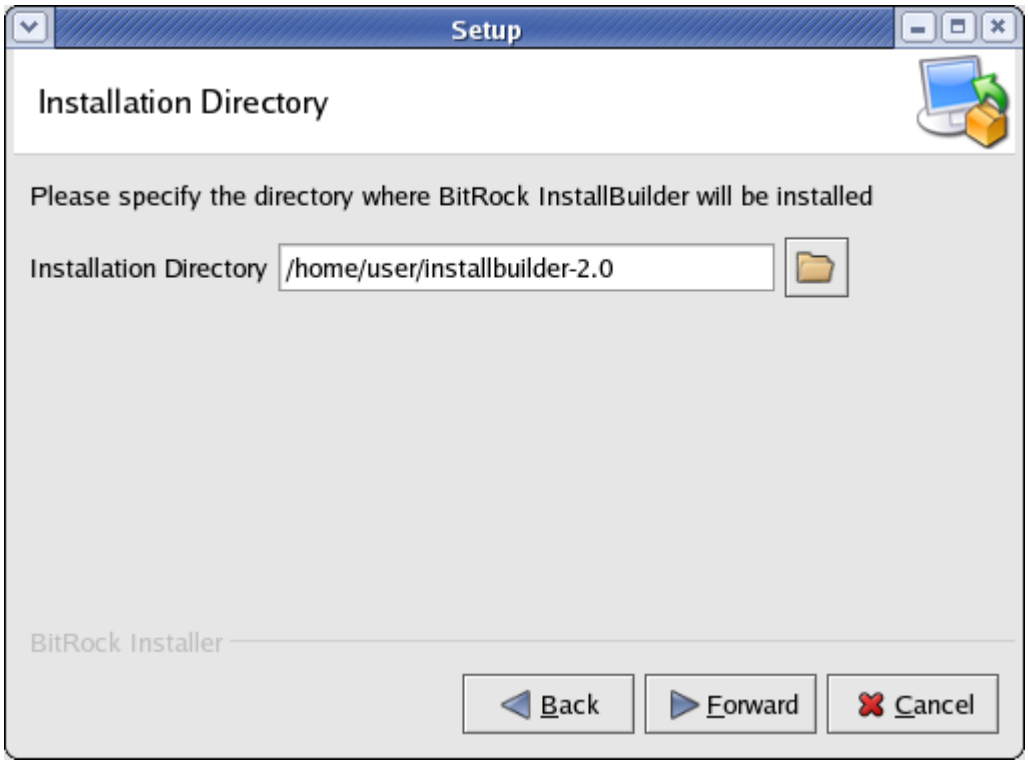
Figure 7 : Linux Welcome Screen



Pressing Next will take you to the License Agreement page, shown in Figure 8. You need to accept the agreement to continue the installation. The next step is to select the installation directory (Figure 9). The default value will be a folder on your home directory if you are running the installer as a regular user (recommended) or `/opt/installbuilder-3.0beta2` if you are running the installation as superuser (root). If the destination directory does not exist, it will be created.

Figure 8 : Linux License Agreement

Figure 9 : Linux Select Installation Directory



The rest of this guide assumes you installed bitrock in /home/user/installbuilder-3.8/

You are ready to start the installation now (Figure 10), which will take place once you press Next (Figure 11). When the installation finishes, you will see the Installation Finished page shown in Figure 12. You may choose to view the README file at this point.

If you found a problem and could not complete the installation, please refer to the Troubleshooting section or contact us at support@bitrock.com. Please refer to the Support section for details on what information you should include with your request.

Figure 10 : Linux Ready To Install

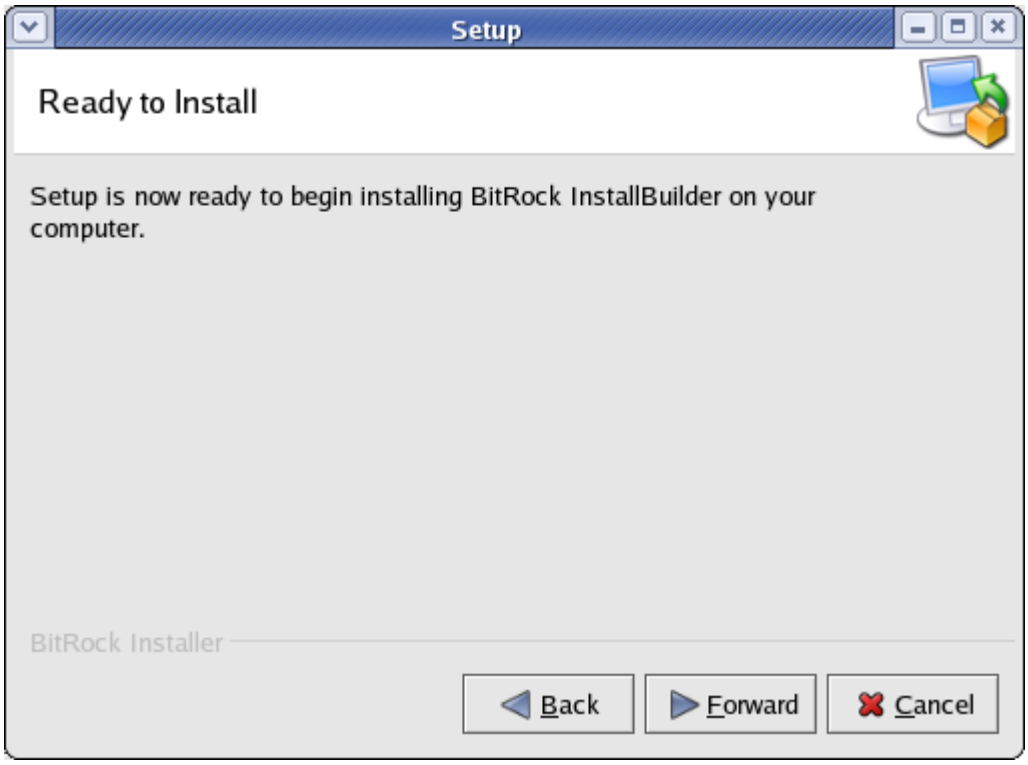


Figure 11 : Linux Installation Under Way

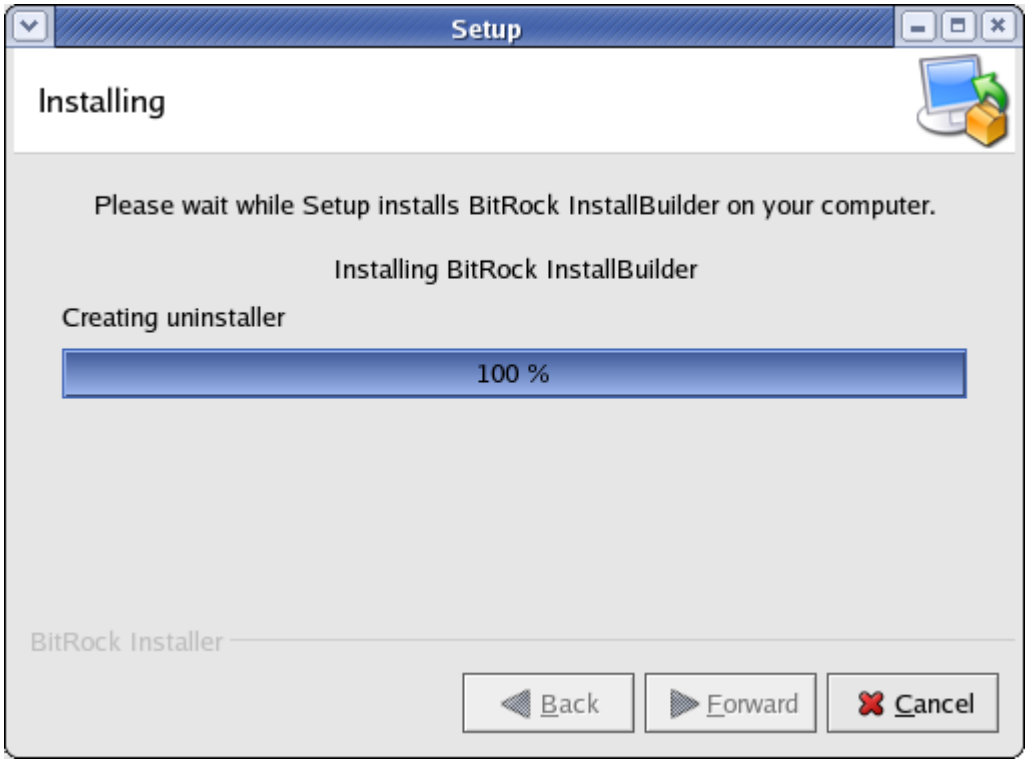
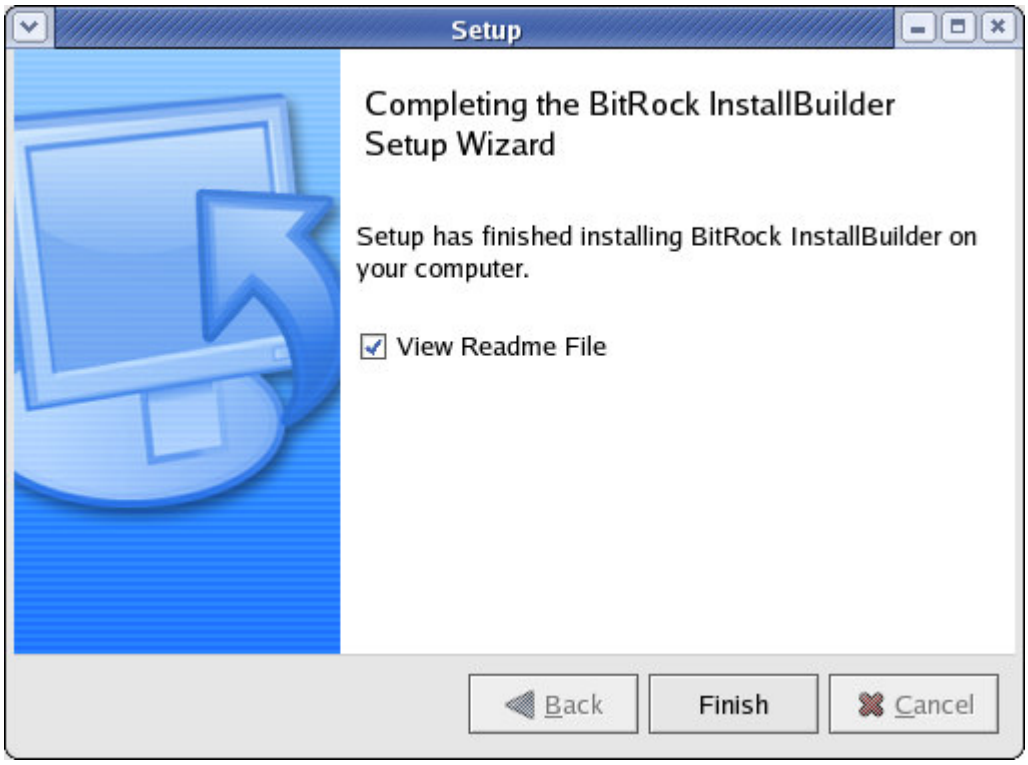


Figure 12 : Linux Installation Completed



Directory Structure

The installation process will create several directories:

- `bin`: BitRock InstallBuilder application binaries.
- `paks`: Support files necessary for creating installers.
- `projects`: Project files for your installers.
- `docs`: Product documentation.
- `demo`: Files for the sample demo project.
- `output`: Finished installers.

You are ready now to start the application and create your first installer, as described in the next section "Building your First Installer"

Building Your First Installer

This section explains how to create your first installer in a few simple steps.

Startup and Basic Information

If you are running Gnome or KDE and performed the installation as a regular user, a shortcut was created on your Desktop. You can either start BitRock InstallBuilder by double-clicking on it or by invoking the binary from the command line:

```
$ /home/user/installbuilder-3.8/bin/builder
```

If you are running Windows, the installer created the appropriate Start Menu entries. Additionally, a shortcut was placed on your Desktop.

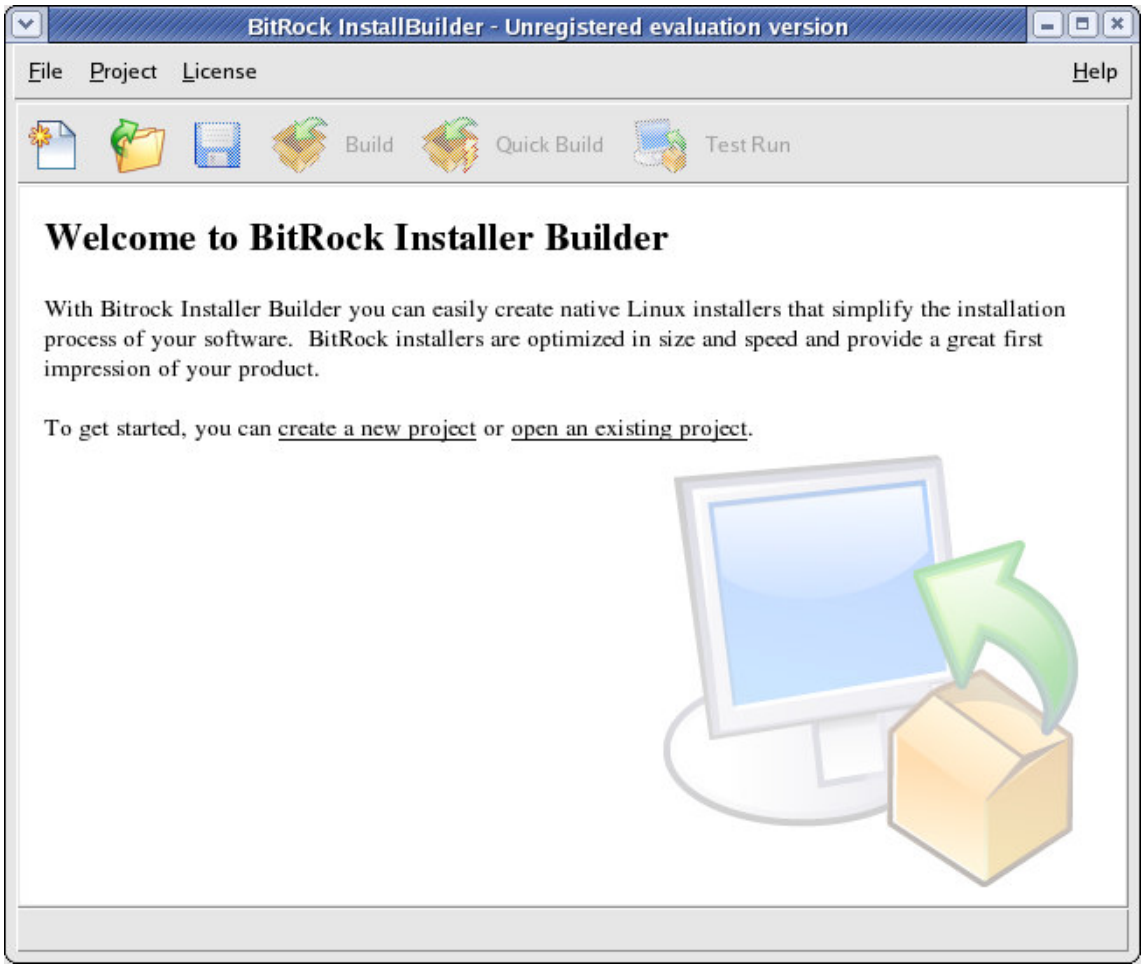
You can also build installers from the command line. Please refer to the section named "Using the Command Line Interface" later in the document.

The initial screen will appear (Figure 13). Press the "New Project" button or select that option from the File menu on the top left corner. A popup Window will appear, asking you for three pieces of information:

- **Product Name**: The full product name, as it will be displayed in the installer
- **Product Filename**: Short version of product name, will be used for naming certain directories and files, and can only contain alphanumeric characters
- **Version Number**: Product version number, will be used for naming certain directories and files.

The rest of this tutorial assumes you kept the default values: "Sample Project", "sample" and "1.0".

Figure 13 : Main Screen



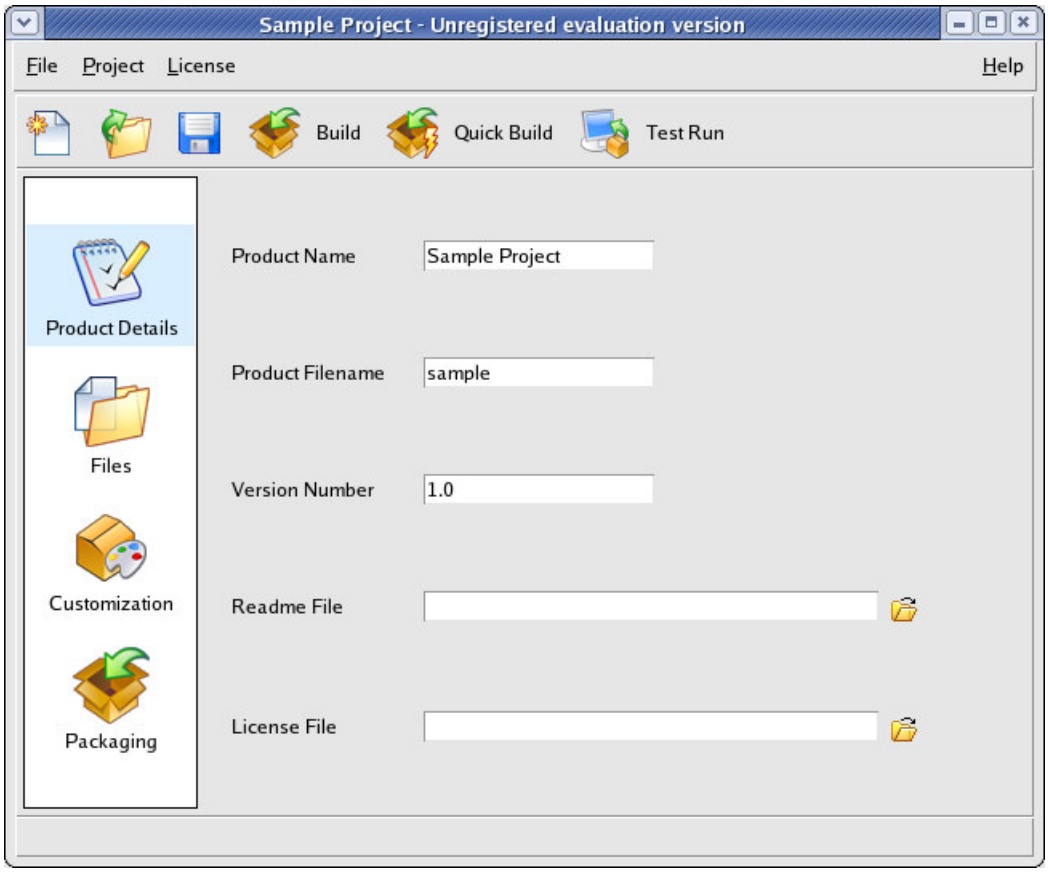
Once you enter the information the "Basic settings screen" (Figure 14) will appear. Here you can specify additional settings:

- **License File**: Path to license file that the user must accept in order to install the software
- **Readme File**: Path to README file that can be shown to the user after installation is completed
- **Save Relative Paths**: Whether to convert absolute paths to relative when saving project files. This is important if the same project file is used by multiple developers. The path will be relative to the location of the project file.

If you do not want to display a license agreement or a README file during installation, you can leave those fields blank.

When is it necessary to use the Save Relative Paths option? It is necessary when the same project file is shared by multiple developers on different machines or when using the same project file on Windows and Unix. This is due to the differences in how paths are specified on each platform.

Figure 14 : Basic Settings

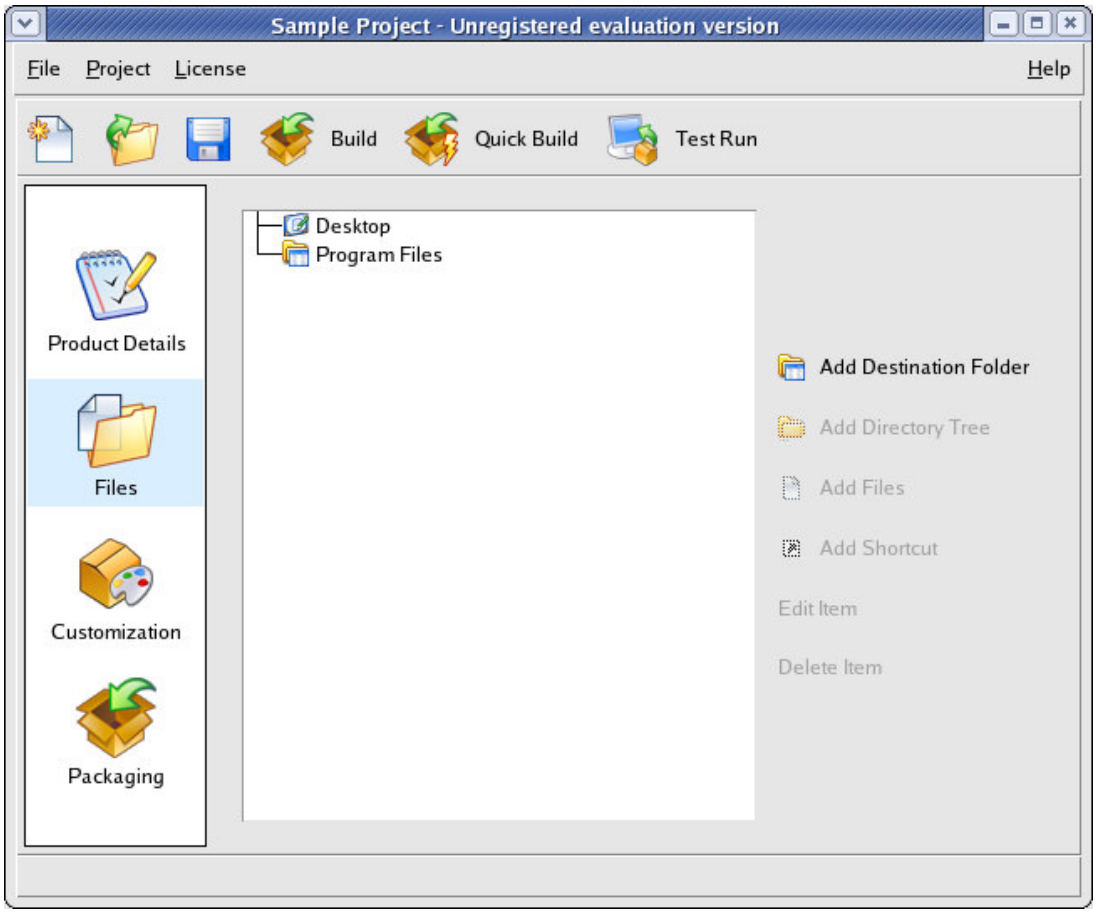


Selecting the Files

The next step is to click on the "Files" icon and go to the screen shown in Figure 15.

The "Program Files" folder represents the target installation directory. You can add files and directories to this folder by selecting the "Program Files" folder and using the "Add File" and "Add Directory Tree" buttons. You can add multiple files pressing down the Control key and clicking on them in the File selection dialog. Multiple selection is not available for directories at this time. The selected files and directories will be copied to the destination the user chooses during installation. If a folder only supports a particular target platform, such as Linux, FreeBSD, IRIX, AIX, Mac OS X, HP-UX, Solaris or Windows, it will only be included in installers for that particular platform.

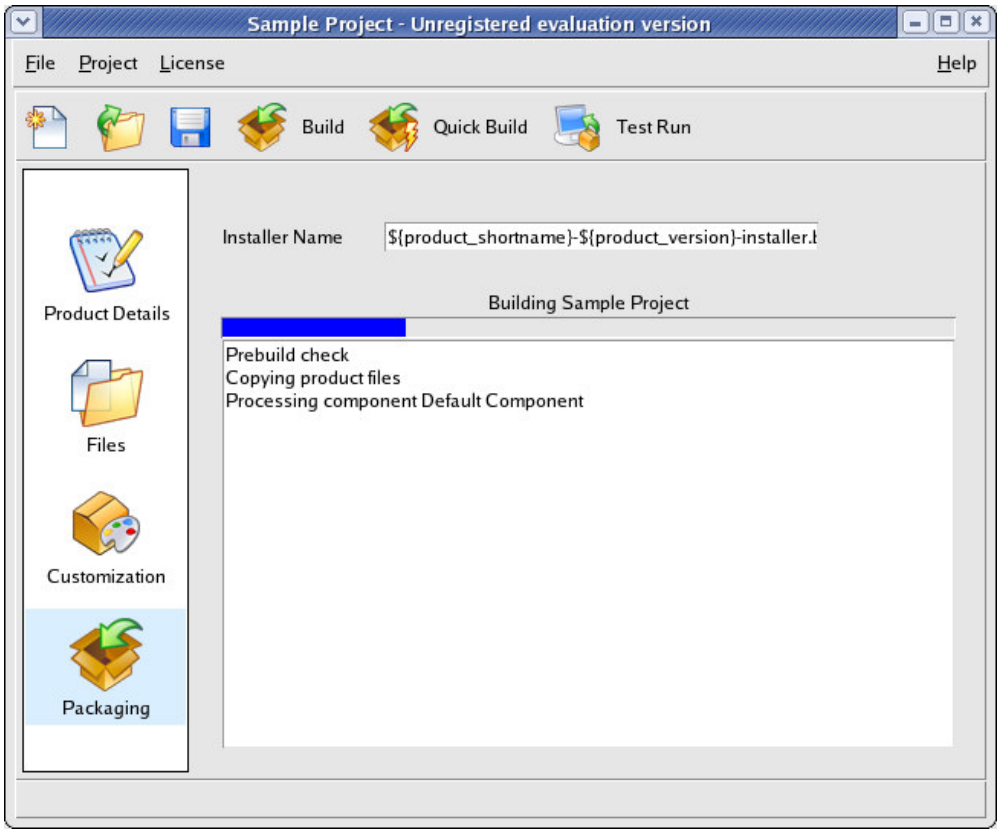
Figure 15 : Files Screen



Most applications only need to add files to the main installation directory. The "Advanced Functionality" section covers how to specify additional installation folders and how to create application shortcuts.

You can now build the installer by pressing the "Build" button. This will take you to the Packaging screen and start the installer building process, as shown in Figure 16. If the build process succeeds, an installer named `sample-1.0-linux-installer.bin` will be placed at the output directory. If you are building a Windows installer, the file will be named `sample-1.0-windows-installer.exe` and if you are building a Mac OS X installer, its name will be `sample-1.0-osx-installer.app`. If any problem is found, such as a file not being readable, a message will be displayed in red and the build will stop.

Figure 16 : Building the installer

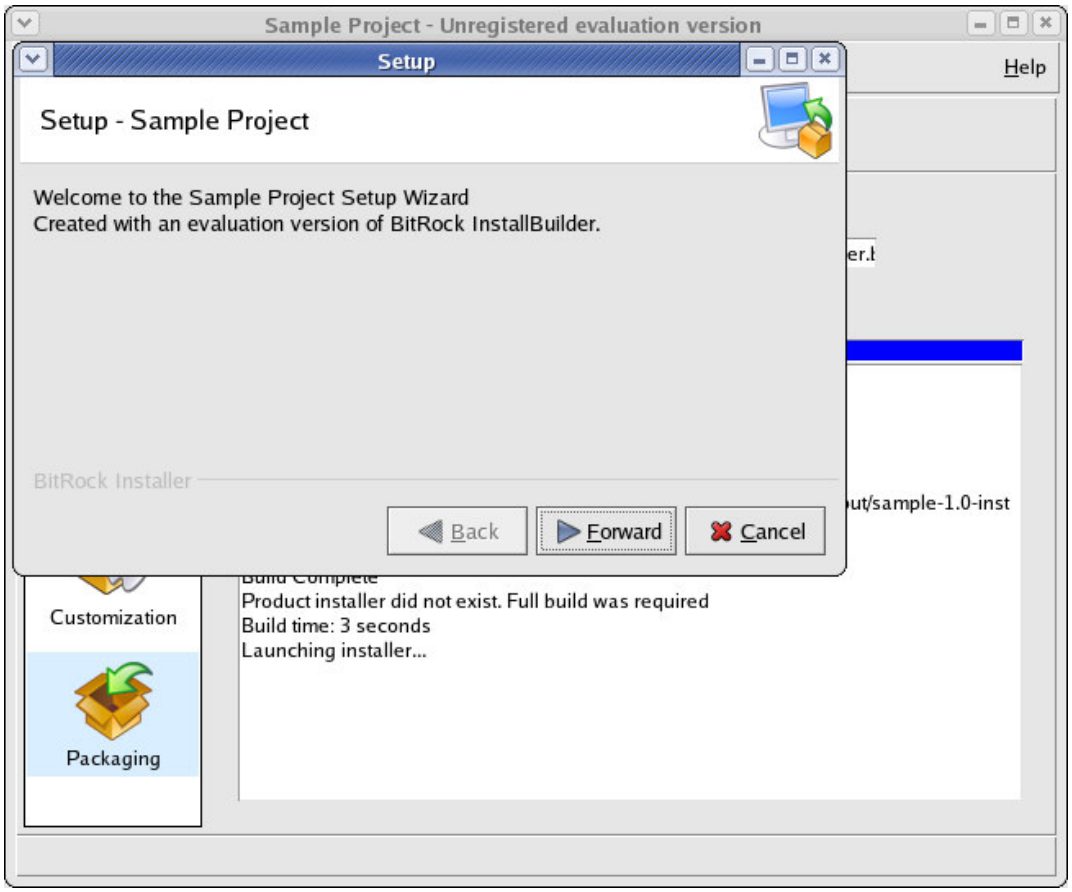


You can test the generated installer by pressing the "Test Run" button, as seen in Figure 17.

What is the difference between Full Build and Quick build ? Creating an installer can take a long time if your product is hundreds of megabytes in size. You can use the Quick Build button to avoid rebuilding an installer from scratch if you are just making changes to installer-specific settings: license and readme file, default installation path, logo image, installation required by root and product name. Those settings will be updated without having to pack again every product file. Of course, you will need to do a regular Build if you make changes to the product files themselves (the ones you added in the Files screen)

You can customize additional installer functionality as explained in the following section.

Figure 17 : Testing the installer



CDROM

It is possible to select a CDROM build target. In this case, a directory is created that includes a folder with common installer files and a setup file for each one of the architectures. This allows you to provide a single CDROM for all platforms, avoiding duplication of data.

Advanced Functionality

This section explains how to customize the generated installers in different ways. The first thing you should know about are installer variables. They can be included in different settings as `${variablename}` and they will be substituted for their values during installation.

For example, if you write the default installation directory as `/opt/${shortName}-${version}` then during installation time the user will see `/opt/sample-1.0` If you update the product version to 2.0 later on, the change will be reflected automatically. So whenever possible, avoid hardcoding references and use installer variables instead.

The current version of the installer supports the following installer variables.

- **${installdir}**: Directory where the product will be installed.
- **${product_fullname}**: Product Name. The full product name, as it will be displayed in the installer
- **${product_shortname}**: Product Filename. Short version of product name, will be used for naming certain directories and files, and can only contain alphanumeric characters
- **${product_version}**: Version Number.
- **${platform_install_prefix}**: Platform-dependant default installation location. In Unix systems, when running as root it will be `/opt` and when running as a regular user, the home directory for that user.
- **${platform_exec_suffix}**: Platform-dependant executable file extension for the generated installer. In Unix systems it is `.bin` and on Windows `.exe`
- **${platform_name}**: Target platform for the installer. Currently it can be `linux`, `linux-ppc`, `freebsd`, `solaris-sparc`, `solaris-intel`, `irix-n32`, `osx`, `windows`, `aix` or `hpux`.
- **${linux_distribution}**: When the installer is running on Linux, it will contain the specific Linux flavor name. Currently, one of `debian`, `suse`, `mandrake`, `redhat`, `fedora`, `slackware` or `unknown` for another distribution.
- **${installation_langcode}**: The ISO code for the language the installer was run with.
- **${installer_is_root_install}**: Whether or not the installer is being run as root or not.
- **${installer_ui}**: Whether the installer is being root in 'text', 'gui' or 'unattended' mode.
- **${installer_directory}**: Directory where the installer binary is located.
- **${machine_hostname}**: Machine hostname
- **${machine_ipaddr}**: Machine hostame IP address

Additionally, it is possible to access any environment variable using the ${env(*varname*)} construct, where *varname* is the name of an environment variable. For example, on Windows you can refer to the system drive with ${env(SYSTEMDRIVE)} and in Linux, Mac OS X, AIX, HP-UX, IRIX, FreeBSD and Solaris to the user home directory with ${env(HOME)}

Installer Customization

In the Customization (Figure 18) and the Packaging screens you can change the default installation settings to match your needs:

User Interface Settings

- **Logo image**: 48x48 GIF or PNG logo image that will be placed at the top right corner of the installer. If no image is specified, the default one will be used
- **Left side image**: 163x314 GIF or PNG image that will be placed at the left side of the installer in the Welcome and Installation Finished pages. If no image is specified, the default one will be used
- **Default installation language**: Default language for the installer
- **Allow language selection**: Allow language selection. If this setting is enabled, the user will be required to specify the language for the installation
- **Wrap License File Text**: Wrap license file text displayed to the user

Installer Settings

- **Require install by administrator?**: Whether installation will require super user privileges (root on Linux, Administrator user on Windows)
- **Installer Name**: Name of the installer created by the build process. If it contains ${product_shortname}, ${product_version}, ${platform_name} or ${platform_exec_suffix} they will be replaced by the appropriate values
- **CDROM files Directory**: Name of the directory that will contain the CDROM files created by the build process
- **Uninstaller directory**: Directory where the uninstaller will be created
- **Installation directory**: Default installation directory

Script Settings

- **Post Install script**: Program that will be executed as the final installation step. The program or script must have execution permissions and you need to include it as part of the installation. Since you do not know beforehand where the user will decide to install the software, you need to prefix it with ${installdir}. You can also pass additional arguments to the script. For example : ${installdir}/bin/myscript.sh somevalue ${installdir} will invoke the myscript.sh program with two arguments, 'somevalue' and the installation directory
- **Post Install script arguments**: Command line arguments to pass to the Post Installation Script
- **Show Post Install result?**: Whether to show the output result of the post installation script, even if it completed successfully
- **Pre Uninstallation script**: Program that will be executed before uninstallation. The program or script must have execution

permissions and you need to include it as part of the installation. Since you do not know beforehand where the user will decide to install the software, you need to prefix it with ${installdir}. You can also pass additional arguments to the script. For example : ${installdir}/bin/myscript.sh somevalue ${installdir} will invoke the myscript.sh program with two arguments, 'somevalue' and the installation directory

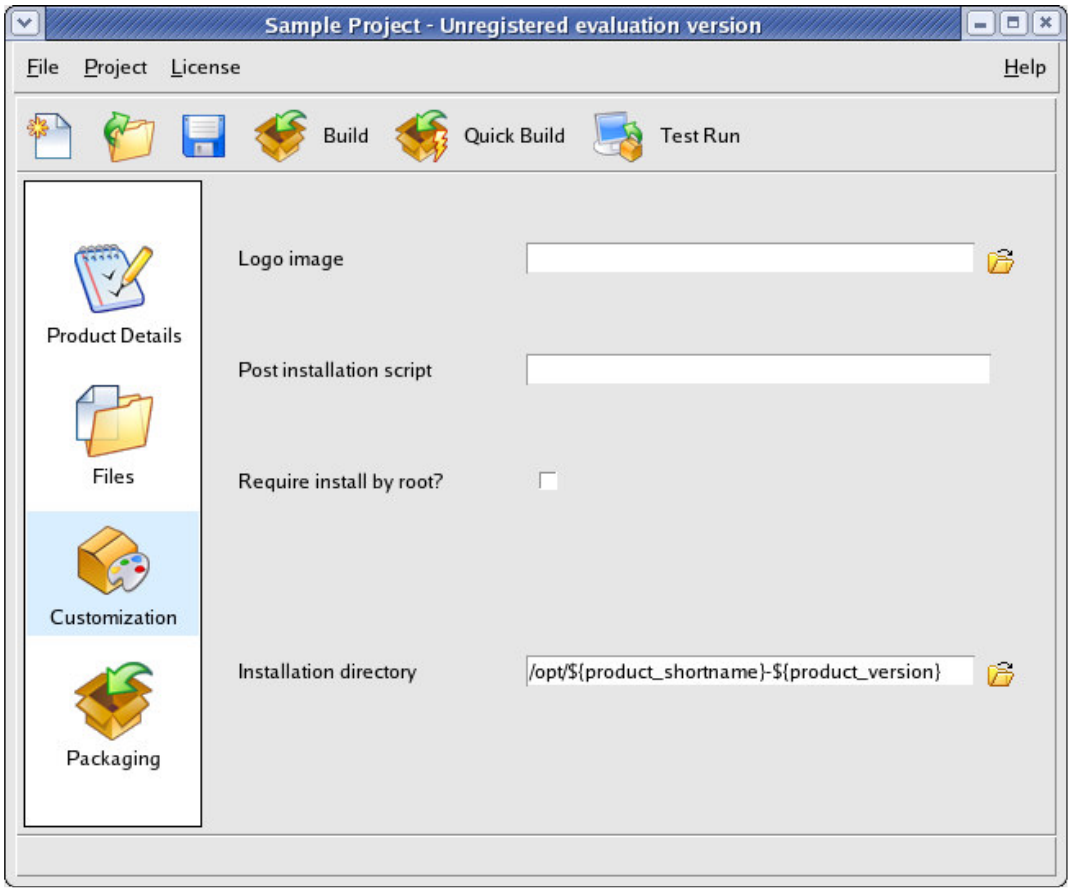
- **Pre Uninst. script arguments**: Command line arguments to pass to the Pre Uninstallation Script

Particularly important is the **Post Install script** setting. It allows you to perform specific actions required to correctly finish installing your product, such as initializing a database or starting a server in the background.

Platform Specific Settings

- **Default Unix File Permissions**: Default Unix file permissions in octal form
- **Default Unix Directory Permissions**: Default Unix directory permissions in octal form

Figure 18 : Customization screen



Additional Settings

The installer supports a number of features that are not yet available through the GUI. For this, you will need to edit the XML project file directly.

- **readmeFileEncoding**: Readme file encoding. Default value: iso8859-1, valid values: iso8859-1 iso8859-2 utf-8 cp1252 ascii macRoman unicode.
- **licenseFileEncoding**: License file encoding. Default value: iso8859-1, valid values: iso8859-1 iso8859-2 utf-8 cp1252 ascii macRoman unicode.
- **deleteOnExit**: Whether to delete the installer binary once the installation has completed
- **rebootRequired**: Whether to ask the user to reboot after installation is completed (Windows-specific option).

Additional Installation Folders

Most applications only install files under the installation directory ("Program Files" folder in the Files screen). It is possible, however, to add additional folders to copy files and directories to, such as `/usr/bin` or `/etc/` by pressing the "Add Destination Folder" button in the Files screen. If you need special permissions to write to the destination folders, you may need to require installation by root (see previous section)

Shortcuts

You can create application, document and URL shortcuts in the Files screen. Program shortcuts are special files containing information such as a program to execute and an icon to display. If you are distributing a GUI program that runs on Windows, KDE or Gnome, you can place a shortcut for your executable in the Desktop or in a folder and the associated icon will be displayed. When the user clicks on the icon, the associated program, document or URL will be launched. Figure 19 shows the prompt you get when adding an Application shortcut to your product installer. It has the following fields:

Common

- **Shortcut text**: Shortcut text
- **Tooltip**: Tooltip text for the shortcut
- **Platforms**: Platforms in which this link shortcut will be created

Unix settings

- **Unix Icon**: GIF or PNG Image to use for the shortcut
- **Program to execute**: Program to execute, including command line arguments
- **Working directory**: Working directory for the program being executed

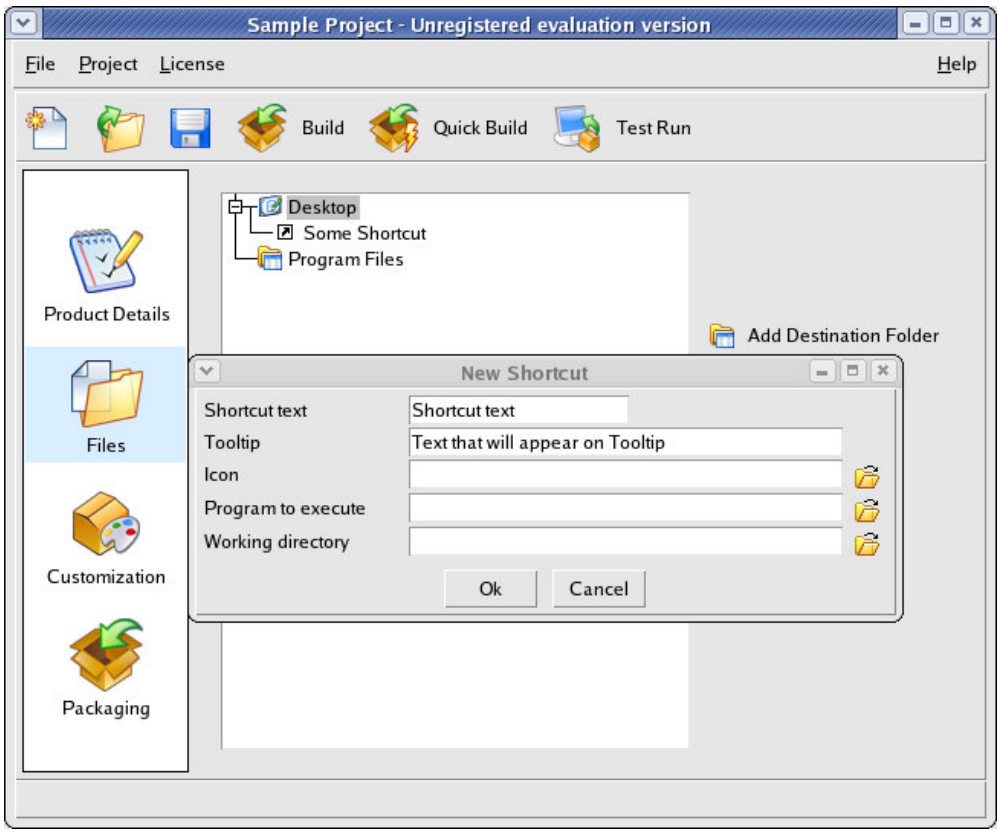
Windows settings

- **Windows Icon**: File containing .ico image
- **Program to execute**: Program to execute
- **Working directory**: Working directory for the program being executed

Notice that the target program to execute must have been installed with your product, so the value for **Program to execute** and/or **winExec** should include a reference to the installation directory and look similar to: `${installdir}/foo/bar/program` where `foo/bar/program` is the path to your program relative to the installation directory. At installation time, `${installdir}` will be substituted by the appropriate value.

It is also possible to create shortcuts that point to directories, documents or URLs. Select the "Document" or "URL" option when creating a shortcut

Figure 19 : Adding a shortcut



Using the Command Line Interface

You can build projects from a shell script or the command line issuing the following command:

```
$ /home/user/installbuilder-3.8/bin/builder build
/path/to/project.xml
```

For example

```
$ /home/user/installbuilder-3.8/bin/builder build
/home/user/installbuilder-3.8/projects/project.xml
```

will compile the Sample Project mentioned earlier in this document.

By default, it will build a Linux installer. You can pass an optional argument to the command line to indicate the target platform. For example:

```
$ /home/user/installbuilder-3.8/bin/builder build
/home/user/installbuilder-3.8/projects/project.xml
windows
```

RPM Integration

BitRock InstallBuilder allows you to register the software installed with the RPM package database.

To enable RPM support add <registerWithPackageDatabase>1</registerWithPackageDatabase> to your installer project file. This will register your installation with the RPM database. From this point on, you will be able to query data about your application and its installed files using your distribution's rpm-based tools as with any other existing rpm package. You will also be able to uninstall the application using your distribution's rpm-based tools.

RPM database integration requires installation as root in an RPM-based distribution. Otherwise, the setting will be ignored.

Additionally, to successfully register an RPM, the following tags must be also present in the XML project file:

```
    <vendor>Your Company Name</vendor>
    <summary>Detailed description of your software</summary>
    <release>0</release>
    <description>A one-line description of your
software</description>
```

The name of the RPM package registered will be ${product_shortname}-${product_version}-${release}

The XML Project File

BitRock InstallBuilder stores all the information about the installer project in an XML file, usually located under `/home/user/installbuilder-3.8/projects/`. The XML format of the file is designed to be reasonably easy to edit by hand, allow automated manipulation using scripts and track changes

using a source control tool such as CVS. You can find a sample XML project file in the Appendix.

Actions

There are a number of installation tasks that are common to many installers, such as changing file permissions, substituting a value in a file, and so on. BitRock InstallBuilder includes a number of useful built-in actions. Currently, the actions can only be accessed by editing the XML project file directly, support for the built-in actions in the GUI building tool is planned for the near future. Actions are either attached to a particular folder tag in the project file (<actionList>) and will be executed after the contents of the folder have been installed, or can be part of specific action lists that are run at specific points during installation . Actions usually take one or more arguments. If one of those arguments is a file matching expression (<files>), the matching will occur only against the contents of folder. If the arguments contain references to installer variables, such as the installation directory, they will be properly expanded before the action is executed.

Change Permissions

This action allows you to change Unix user and group permissions for files and directories. It takes a list of glob file patterns separated by ';' and the octal value for file permissions, as defined in the Unix manual page for the `chmod` command

Supported Platforms: Linux, FreeBSD, HP-UX, IRIX, AIX, Solaris and Mac OS X.

Example:

```
<changePermissions>
   <files>*/*.sh;*/*.bin</files>
   <permissions>755</permissions>
</changePermissions>
```

DOS To Unix

This action allows you to convert plain text files in DOS/Mac format to Unix format. For more infomation refer to the Unix manual page for the `dos2unix` command

Supported Platforms: All Platforms.

Example:

```
<dos2unix>
   <files>*/*.sql;*/*.sh;*.ascii</files>
</dos2unix>
```

Add Environment Variable

This action allows you to create or change the value of an environment variable. On Linux, FreeBSD, AIX, IRIX, HP-UX and Solaris, the variable will be added to any of the shell configuration files, with the specific syntax for each shell: `~/.bashrc`, `~/.profile`, `~/.cshrc`, `~/.tcshrc`, `~/.zprofile`.

On Windows, the optional <scope> field allows you to specify whether the environment variable should be added to the current user's environment ("user") or globally ("system", which is the default). If adding the environment variable globally fails, it will try to add it to the current user's environment.

Supported Platforms: All Platforms.

Example:

```
<addEnvironmentVariable>
      <name>MYAPP_HOME</name>
      <value>${installdir}</value>
      <scope>user</scope>
</addEnvironmentVariable>
```

Delete Environment Variable

This action allows you to delete an environment variable, specified by <name>. The optional <scope> field allows you to specify whether the environment variable should be deleted in the current user's environment ("user") or globally ("system", which is the default).

Supported Platforms: Windows

Example

```
<deleteEnvironmentVariable>
      <name>MYAPP_HOME</name>
      <scope>user</scope>
</deleteEnvironmentVariable>
```

Add Directory to PATH

This action allows you to add a directory to the system PATH. On Linux, FreeBSD, HP-UX, AIX, IRIX and Solaris, the variable will be added to the PATH definition in the shell configuration files: `~/.bashrc`, `~/.profile`, `~/.cshrc`, `~/.tcshrc`, `~/.zprofile`.

Supported Platforms: All Platforms.

Example:

```
            <addDirectoryToPath>
               <path>$MYAPP_HOME/bin</path>
            </addDirectoryToPath>
```

Backup File

This action allows you to backup a file or directory. It will create a new file or directory, named after the path specified, with the suffix `.bak0`. If a backup file with that name already exists, it will create a new one ending in `.bak1` (or `.bak2`, etc.)

Supported Platforms: All Platforms.

Example:

```
            <createBackupFile>
               <path>${installdir}/conf/myapp.conf</path>
            </createBackupFile>
```

Delete File

This action allows you to delete a file or directory. The path value can take a glob style pattern.

Supported Platforms: All Platforms.

Example:

```
            <deleteFile>
               <path>/tmp/mytempfiles*.*</path>
            </deleteFile>
```

Copy File

This action allows you to copy files or directories.

Supported Platforms: All Platforms.

Example:

```
            <copyFile>

         <origin>${installdir}/conf/myfile.template</origin>
```

```
        <destination>${installdir}/conf/myfile.conf</destination>
            </copyFile>
```

Rename File

This action allows you to rename a file or directory.

Supported Platforms: Windows, Linux, FreeBSD, HP-UX, AIX, IRIX, Solaris and Mac OS X.

Example:

```
            <renameFile>

<origin>${installdir}/conf/myfile.template</origin>

<destination>${installdir}/conf/myfile.conf</destination>
            </renameFile>
```

Get Free Disk Space

This action allows you to get the disk space left for a particular directory. The directory is specified in the 'path' argument and the value, in kb, will be stored in the variable specified by 'variable'

Supported Platforms: All Platforms.

Example:

```
            <getFreeDiskSpace>
               <variable>diskspace</variable>
               <path>${installdir}</path>
            </getFreeDiskSpace>
```

Substitute Value in Text

This action allows you to specify pattern/value pairs that will be substituted in certain files. It takes a list of glob file patterns separated by ';' and a list of the pattern/value pairs

Supported Platforms: All Platforms.

Example:

```
            <substitute>
              <files>*/launchMyJavaApp.sh</files>
```

```
            <substitutionList>
              <substitution>
                <pattern>@@INSTALLDIR@@</pattern>
                <value>${installdir}</value>
              </substitution>
              <substitution>
                <pattern>@@JAVADIR@@</pattern>
                <value>${installdir}/jre</value>
              </substitution>
            </substitutionList>
          </substitute>
```

Append Text to File

This action allows you to add a text at the end of the file specified.

Supported Platforms: All Platforms.

Example:

```
          <addTextToFile>
             <file>${installdir}/bin/appstart.sh</file>
             <text>export JAVA_HOME=${installdir}/jre
             export PATH=$JAVA_HOME/bin:$PATH</text>
          </addTextToFile>
```

Throw Error

Supported Platforms: All Platforms.

This action allows you to throw an error. If used inside a parameter <validationActionList> section an error message will be displayed to the user and the user will be prompted again for the required information. If used in any other action list section, such as <preInstallationActionList>, it will display an error message to the user and the application will exit.

The following example code, when placed in <preInstallationActionList> will throw an error if the Apache configuration file is not writable by the current user:

```
          <throwError>
             <text>The Apache configuration file is not
writable by the current user</text>
             <ruleList>
               <fileTest>
                 <path>/usr/local/apache/conf/httpd.conf</path>
                 <condition>not_writable</condition>
               </fileTest>
             </ruleList>
          </throwError>
```

Java (tm) Autodetection

Supported Platforms: All Platforms.

This action autodetects an existing Java (tm) installation in the system and creates the corresponding installer variables.

The action is usually placed at the <preInstallationActionList> and if no valid JRE is found, the installer will abort with an error listing the supported JREs.

Each element in the <validVersionList> contain the following fields:

- **vendor**: sun to allow only Sun Microsystems JREs, ibm for IBM JREs and empty for any.
- **minVersion**: Minimum supported version of the JRE. Leave empty to not require a minimum version
- **maxVersion**: Maximum supported version of the JRE. Leave empty to not require a maximum version. If specified only with major and minimum version numbers then it will match any number in the series. For example, 1.4 will match any 1.4.x version (1.4.1, 1.4.2, ...) but not a 1.5 series JRE.

The following example will select any Sun Microsystems JRE 1.3 or newer (for example, 1.3, 1.4, 1.5) or any IBM JRE with version number equal or greater than 1.4.2 but inside the 1.4 series (1.5 will not work).

```
<autodetectJava>
  <validVersionList>
    <validVersion>
      <vendor>sun</vendor>
      <minVersion>1.4.2</minVersion>
      <maxVersion>1.4</maxVersion>
    </validVersion>
    <validVersion>
      <vendor>ibm</vendor>
      <minVersion>1.3</minVersion>
      <maxVersion></maxVersion>
    </validVersion>
  </validVersionList>
</autodetectJava>
```

Upon successful autodetection, the following installer variables wil be created:

- ${java_executable}: Path to the java command line binary (java.exe in Windows). For example /usr/bin/java, C:\Program Files\Java\j2re1.4.2_03\java.exe.
- ${javaw_executable}: Path to javaw.exe binary, if found. Otherwise defaults to the value of `java_executable`.
- ${java_version}: For example, 1.4.2_03
- ${java_version_major}: For example, 1.4
- ${java_vendor}: sun or ibm.

The installer will look for valid JREs in the following places and select the first one that meets all the requirements:

- Standard installation paths.
- Windows Registry, default environment PATH.
- Using JAVA_HOME, JAVAHOME or JDK_HOME environment variables, if present.

You can allow the end-user to select the appropriate JVM by adding <promptUser>1</promptUser> inside the <autodetectJava> tag

Windows Registry

Supported Platforms: Windows

This action allows you to create a new registry key or to modify the value of an existing registry key. The <type> tag is optional, and specifies the type of registry entry to create. It can be `REG_BINARY`, `REG_NONE`, `REG_SZ`, `REG_EXPAND_SZ`, `REG_DWORD`, `REG_BIG_ENDIAN`, `REG_LINK`, `REG_MULTI_SZ`, `REG_RESOURCE_LIST`. The default value is `REG_SZ`.

```
            <registrySet>

<key>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager\Environment</key>
               <name>MY_APPDIR</name>
               <value>${installdir}</value>
               <type>REG_SZ</type>
            </registrySet>
```

This action allows you to store the value of a registry key in an installer variable. If the key or name does not exist, then the variable will be created empty.

```
            <registryGet>
               <variable>installdir</variable>

<key>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager\Environment</key>
               <name>MY_APPDIR</name>
            </registryGet>
```

This action allows you to delete a registry key. If the key to delete does not exist the action will be ignored.

```
            <registryDelete>

<key>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager\Environment</key>
```

```
                <name>MY_APPDIR</name>
            </registryDelete>
```

Access Properties Files

Supported Platforms: All Platforms.

This action allows accessing Java-style properties files. You can store the values into an installer variable by referencing the key.

The following example looks for the serverport key in the property file and store its value in the 'port' installer variable.

```
            <propertiesFileGet>
                <file>/path/to/startup.conf</file>
                <variable>port</variable>
                <key>serverport</key>
            </propertiesFileGet>
```

Change Owner and Group of a File or Directory

Supported Platforms: Linux, FreeBSD, HP-UX, AIX, IRIX, Solaris and Mac OS X.

This action allows you to change the owner of a file or directory and its group. Because these changes require administrative privileges, you will need to require installation by administrator.

```
            <changeOwnerAndGroup>
                <files>*/somefile.conf;*/var/somefile</files>
                <owner>nobody</owner>
                <group>nobody</group>
            </changeOwnerAndGroup>
```

Create Symbolic Link

This action allows you to create symbolic links to files or directories.

Supported Platforms: Linux, FreeBSD, HP-UX, AIX, IRIX, Solaris and Mac OS X.

```
            <createSymLink>

<linkName>${installdir}/bin/somelinkname</linkName>
                <target>${installdir}/bin/somefile</target>
            </createSymLink>
```

Set Installer Variable

This action allows you to create Installer Variables that can be included in different settings of the installer as `${variablename}` and that will be substituted for their values during installation.

Supported Platforms: All Platforms.

```
            <setInstallerVariable>
                <name>BD_SERVER_PORT</name>
                <value>3306</value>
            </setInstallerVariable>
```

Set Installer Variable from Script Output

This action allows you to set the value of an installer variable based on the output of a script or program.

Supported Platforms: All Platforms.

```
            <setInstallerVariableFromScriptOutput>
                <name>myhostname</name>
                <exec>hostname</exec>
                <execArgs>-f</execArgs>
            </setInstallerVariableFromScriptOutput>
```

Run Program

Supported Platforms: All Platforms.

This action allows you to run an external program or script.

```
            <runProgram>

<program>${installdir}/utils/scripts/mysql_dump_mysql.bat</program>
                <programArguments></programArguments>
            </runProgram>
```

If the installer is running as the administrator user on Unix platforms, you can use the optional `<runAs>` tag to specify an user to run the command as. If the installer is running on Windows or as a regular user, the tag will be ignored

Run Console Program

Supported Platforms: Linux, FreeBSD, HP-UX, AIX, IRIX, Solaris and Mac OS X.

This action allows you to run an external text-based program or script which requires user input. This only makes sense if running the installer in text mode, so you may want to check this is the case using a <compareText> rule and the ${installer_ui} installer variable

```
           <runConsoleProgram>

<program>${installdir}/utils/scripts/ask_user.sh</program>
               <programArguments></programArguments>
           </runConsoleProgram>
```

Add User

Supported Platforms: Linux, FreeBSD, AIX, HP-UX, Solaris.

This action allows you to add a user to the system. It can take an optional <homedir> parameter to specify the home directory of the newly created account.

```
           <addUser>
                <username>John</username>
           </addUser>
```

Delete User

Supported Platforms: Linux, FreeBSD, HP-UX, AIX, Solaris.

This action allows you to delete a user from the system.

```
           <deleteUser>
                <username>John</username>
           </deleteUser>
```

Add Group

Supported Platforms: Linux, FreeBSD, HP-UX, AIX, Solaris.

This action allows you to add a group to the system.

```

```

```
<addGroup>
    <groupname>developers</groupname>
</addGroup>
```

Add Group To User

Supported Platforms: Linux, FreeBSD, HP-UX, AIX, Solaris.

This action allows you to add a supplementary group to a user. This way, the user is also member of that group. However, make sure that the group already exists. If no username is given, then the current logged in user is selected.

```
<addGroupToUser>
    <username>John</username>
    <groupname>admin</groupname>
<addGroupToUser>
```

Delete Group From User

Supported Platforms: Linux, FreeBSD, HP-UX, AIX, Solaris.

This actions allows you to delete a supplementary group from a user.

```
<deleteGroupFromUser>
     <username>John</username>
     <groupname>admin</groupname>
</deleteGroupFromUser>
```

Action Group

You can use action groups when you want to execute more than one action based on the same set of rules. See the Conditional Evaluation section for details on rule-based execution of actions.

```
<actionGroup>
    <actionList>
    <setInstallerVariable>
        <name>installapache</name>
        <value>1</value>
    </setInstallerVariable>
    <setInstallerVariable>
        <name>installtomcat</name>
        <value>1</value>
    </setInstallerVariable>
    </actionList>
    <ruleList>
```

```
                <compareText>
                    <text>${installtype}</text>
                    <logic>equals</logic>
                    <value>server</value>
                </compareText>
            </ruleList>
        </actionGroup>
```

Additional Actions

We are continuosly adding new actions based on customer feedback. Let us know your suggestions for custom actions that could make your installer development easier.

Pre Installation Actions

You can specify a <preInstallationActionList> section in the project file. It can include a list of actions to execute before the installation process takes place, such as setting user-defined installer variables that will be used later on or detecting a Java (tm) Runtime Environment.

Pre Uninstallation Actions

You can specify a <preUninstallationActionList> section in the project file. It can include a list of actions to execute before the uninstallation process takes place, such as unsetting user-defined installer variables or deleting files created after installation occured.

Parameter Validation Actions

You can specify a <validationActionList> section inside a parameter section. It can include a list of actions to execute once the user has specified a value in the user interface page associated with the parameter and has pressed the Next button (or Enter in a text-based interface). The actions can be used to check that the value is valid (for example, that it specifies a path to a valid Perl interpreter). If any of the actions result in an error, the error message will be displayed back to the user and the user will be prompted to enter a valid value.

Pre Show Page Actions

You can specify a <preShowPageActionList> section inside a parameter section. It can include a list of actions to execute before the corresponding parameter page is displayed. This can be useful for changing the value of the parameter before it is displayed.

Post Show Page Actions

You can specify a <postShowPageActionList> section inside a parameter section. It can include a list of actions to execute after the corresponding

parameter page has been displayed. This can be useful for performing actions or setting environment variables based on the value of the parameter.

Final Page Actions

You can specify a <finalPageActionList> section in the project file. It can include a list of actions to execute after the installation has completed and the final page has been displayed to the user. These actions usually include launching the program just installed, whether a desktop or server application. The text displayed can be specified in the <progressText> property.

Custom Pages

BitRock InstallBuilder allows you to create custom installer pages to ask the user for additional information, validate that information, and use it during the installation process. An example of this would be to pass the information to post-installation scripts.

You can define such pages by adding parameters to the <parameterList> section in the XML project file. There are different types of parameters: strings, booleans, option selection, and so on. Each one of them will be displayed to the user appropriately through the GUI and text interfaces. For example, a file parameter will be displayed with a graphical file selection button next to it and a option selection parameter will be displayed as a combobox. The parameters will also be available as command line options and as installer variables.

Parameter example:

```
 <fileParameter>
     <name>apacheconfig</name>
     <cliOptionName>apacheconfig</cliOptionName>
     <ask>yes</ask>
     <default>/etc/httpd/conf/httpd.conf</default>
     <title>Configuring Apache</title>
     <explanation>Please specify the location of the Apache
configuration file</explanation>
     <description>Apache Configuration File</description>
     <mustBeWritable>yes</mustBeWritable>
     <mustExist>1</mustExist>
     <value></value>
 </fileParameter>
```

This will create the appropriate GUI screens for the graphical installers and make the parameter available as the command line option --apacheconfig and as the installer variable ${apacheconfig}.

Common Fields

A number of fields are common across all parameters:

- **name**: Name of the parameter. This will be used to create the corresponding installer environment variable and command line option. Because of that, it may only contain alphanumerical characters.
- **value**: Value for the parameter.
- **default**: Default value, in case one is not specified by the user.
- **explanation**: Long description for the parameter.
- **description**: Short description for the parameter.
- **title**: Title that will be displayed for the corresponding installer page. If none is specified, the **description** field will be used instead.

- **cliOptionName**: Text to use for setting the value of the parameter through the command line interface. If none is used, it will default to the value of the **name** field.
- **ask**: Whether to show or not the page to the end user (it can still be set through the command line interface).
- **width**: Width in characters of the corresponding field in the GUI page. If not specified, it defaults to 40.

Each one of the fields can reference installer variables (${product_fullname}, ${installdir}, and so on) and they will be substituted at runtime.

String Parameter

The string parameter allows you to request a text string from the user. It accepts all the common options.

Example:

```
    <stringParameter>
      <name>hostname</name>
      <default>localhost</default>
      <value></value>
      <ask>1</ask>
      <description>Hostname</description>
      <explanation>Please enter the hostname for your application
server.</explanation>
    </stringParameter>
```

File and Directory Parameter

File and directory parameters ask the user to enter a file or directory. They support additional fields:

- **mustExist**: Whether to require or not that the file or directory must already exist.
- **mustBeWritable**: Whether to require or not that the file or directory must be writable. If the file or directory does not already exist, whether it can be created.

Example:

```
     <directoryParameter>
            <name>installdir</name>
            <value></value>
            <description>Installation Directory</description>
            <explanation>Please specify the directory where
${product_fullname} will be installed</explanation>
```

```
<default>${platform_install_prefix}/${product_shortname}-
${product_version}</default>
            <cliOptionName>prefix</cliOptionName>
            <ask>yes</ask>
            <mustBeWritable>yes</mustBeWritable>
        </directoryParameter>
```

Boolean Parameter

Boolean parameters are identical to string parameters, only that they take a boolean value.

Example:

```
    <booleanParameter>
      <name>createdb</name>
      <ask>yes</ask>
      <default>1</default>
      <title>Database Install</title>
      <explanation>Should initial database structure and data be
created?</explanation>
      <value>1</value>
    </booleanParameter>
```

Text Display Parameter

This parameter will display a read-only text information page. It does not support the **ask** or **cliOptionName** fields

Example:

```
    <infoParameter>
      <name>serverinfo</name>
      <title>Web Server</title>
      <explanation>Web Server Settings</explanation>
      <value>Important Information! In the following screen you
will be asked to provide (...)</value>
    </infoParameter>
```

Choice Parameter

A choice parameter allows the user to select a value among a predefined list. In GUI mode, it will be represented by a combobox. It takes an extra field, **optionList**, which contains a list of value/text pairs. The **text** will be the description presented to the user for that option and the **value** the value of the associated installer variable if the user selects that option

Example:

```
 <choiceParameter>
      <ask>1</ask>
      <default>http</default>
      <description>Which protocol?</description>
      <explanation>Default protocol to access the login
page.</explanation>
      <title>Protocol Selection</title>
      <name>protocol</name>
      <optionList>
        <option>
          <value>http</value>
          <text>HTTP (insecure)</text>
        </option>
        <option>
          <value>https</value>
          <text>HTTPS (secure)</text>
        </option>
      </optionList>
 </choiceParameter>
```

Password Parameter

A password parameter allows the user to input a password and confirm it. The password will not be echoed back to the user in text mode installations and will be substituted by '*' characters in GUI mode installations. The user needs to retype the password and if the entries do not match, an error will be displayed.

Example:

```
      <passwordParameter>
        <ask>yes</ask>
        <name>Master password</name>
        <description>Password</description>
        <descriptionRetype>Retype password</descriptionRetype>
        <explanation>Please provide a password for the database
user</explanation>
        <cliOptionName>password</cliOptionName>
        <default/>
        <value/>
      </passwordParameter>
```

Group Parameter

A group parameter allows you to logically group other parameters. They will be presented in the same screen on GUI and text installers. You need to place the grouped parameters in a parameterList section, as shown in the example.

Example:

```
   <parameterGroup>
     <explanation>Please enter the username and password for your
database.</explanation>
     <parameterList>
     <stringParameter>
        <name>Username</name>
        <default>admin</default>
        <description>Userna,e</description>
      </stringParameter>
      <passwordParameter>
        <ask>yes</ask>
        <name>Master password</name>
        <description>Password</description>
        <descriptionRetype>Retype password</descriptionRetype>
        <explanation>Please provide a password for the database
user</explanation>
        <cliOptionName>password</cliOptionName>
      </passwordParameter>
      </parameterList>
   </parameterGroup>
```

Conditional Evaluation

BitRock InstallBuilder allows you to control whether certain actions take place, pages are shown or files are installed. You can include a <ruleList> section in action, parameter and folder sections. Each <ruleList> contains a set of rules or conditions that are evaluated, and depending on the result, the action is executed, the page associated with the parameter shown or the folder installed. By default, rules are evaluated with 'and' logic; that is, all rules must be true for the result of the evaluation to be true. You can change that by adding a <ruleLogic>or</ruleLogic> section. In that case it will only be necessary that one of the rules be true for the result of the evaluation to be true.

Each ruleList can contain in turn one or more <compareText>, <fileTest>, <compareValues>, <compareTextLength>, <fileContentTest> sections.

<compareText> text comparison rules can contain three fields:

- **text**: The text to apply the logic comparison to, usually the value of an installer or environment variable.
- **logic**: One of `equals`, `contains`, `does_not_contain` or `does_not_equal`.
- **value**: The value that the text will be compared with.

Example:

```
<compareText>
    <text>${server}</text>
    <logic>equals</logic>
    <value>Apache</value>
</compareText>
```

<compareValues> value comparison rules contain three fields:

- **value1**: Left side value of the logic expresion, usually the value of an installer or environment variable.
- **logic**: One of `equals`, `greater_or_equal`, `greater`, `less`, `less_or_equal` or `does_not_equal`.
- **value2**: Right side value of the logic expresion.

Example:

```
<compareValues>
    <value1>${diskspace}</value1>
    <logic>less</logic>
    <value2>100000</value2>
</compareValues>
```

<compareTextLength> comparison rules contain three fields:

- **text**: Text to compare, usually the value of an installer or environment variable.
- **logic**: One of `equals`, `greater_or_equal`, `greater`, `less`, `less_or_equal` or `does_not_equal`.
- **length**: Length to compare to

Example:

```
<compareTextLength>
    <text>${password}</text>
    <logic>greater</logic>
    <length>8</length>
</compareTextLength>
```

<fileTest> file testing rules contain two fields:

- **path**: The path to the file to test.
- **condition**: One of `exists`, `not_exists`, `writable`, `not_writable`, `readable`, `not_readable`, `executable`, `not_executable`, `is_directory`, `is_not_directory`, `is_file`, `is_not_file`, `is_empty`, `is_not_empty`.

Example:

```
<fileTest>
    <path>/usr/bin/perl</path>
    <condition>executable</condition>
</fileTest>
```

<fileContentTest> file content testing rules contain three fields:

- **path**: The path to the file to test.
- **logic**: One of `contains`, `does_not_contain`
- **text**: The text to apply the logic comparison to, usually the value of an installer or environment variable.

Example:

```
<fileContentTest>
    <path>/etc/group</path>
    <logic>contains</logic>
    <text>apache</text>
</fileContentTest>
```

The following example shows how a particular script will be executed only if the installation type is 'server', it is running on Linux and a certain file is not already present in the system. The value for the installation type was set during the installation process using a user-defined parameter, as explained earlier in this document.

```
<runProgram>
    <ruleEvaluationLogic>and</ruleEvaluationLogic>
        <ruleList>
            <compareText>
                <text>${installtype}</text>
                <logic>equals</logic>
                <value>server</value>
            </compareText>
            <compareText>
                <text>${platform_name}</text>
                <logic>equals</logic>
                <value>linux</value>
            </compareText>
            <fileTest>
                <path>/etc/init.d/myservice</path>
                <condition>not_exists</condition>
            </fileTest>
        </ruleList>
     <program>${installdir}/bin/install_service.sh</program>
</runProgram>
```

Running the Installer

BitRock InstallBuilder can generate a self-contained native Windows binary that can run on all supported Windows platforms, a Solaris Sparc binary and a Solaris Intel binary that can run on all supported Solaris versions, an Mac OS X binary, a FreeBSD 4.x/5.x binary, a HP-UX 11 binary, an IRIX binary, an AIX binary and a Linux PPC, x64, x86 or s390 Linux binary that can run on most Linux platforms. You can run it either by invoking it on the command line or double-clicking it from your desktop environment. On Linux, FreeBSD, HP-UX, AIX, IRIX, Solaris and Mac OS X, the only requirement is that the file has executable permissions, which it has by default when it is created. Sometimes those permissions can be lost, such as when downloading an installer from a website. In that case you can add back the executable permissions with:

```
$ chmod u+w installbuilder-professional-3.8-linux-
installer.bin
```

Although the generated OSX installers are regular .app applications, you may need to add them to a zip file or disk image for distribution over the web.

On Windows, the installer runs graphically with native look and feel or can be invoked in unattended mode (see below). On Mac OS X, the installer runs with the native Aqua look and feel and can also be run in text and unattended modes. On Linux, FreeBSD, HP-UX, AIX, IRIX, and Solaris, there are multiple installation modes:

- **GTK**: This is a native installation mode based on the GTK 2.0 toolkit. The GTK libraries must be present in the system (they are installed by default in most Linux distributions). This is the default installation mode. If the GTK libraries are not available, the X-Window installation mode will be automatically used instead. The GTK mode is available on Linux only.
- **X-Window**: This is a self contained installation mode that has no external dependencies. It will be started when the GTK mode is not available or can be explictly requested with the `--mode xwindow` command line switch.
- **Command line**: Designed for remote installation or installation on servers without X-Window support. You can find a sample installation log in the Appendix. This installation mode is started by default when a graphical environment is not available or by using the `--mode text` command line option to the installer.
- **Unattended Installation**: It is possible to perform unattended or silent installations using the `--mode unattended` command line option. This is useful for automating installations or for inclusion in shell scripts, as part of larger installation processes.

For all modes, you can modify the default installation directory by passing the `--prefix /path/to/installdir` command line option to the installer

On Linux, FreeBSD, HP-UX, AIX, IRIX, Solaris and Mac OS X, an installation log named `bitrock_installer.log` will be created in `tmp` On Windows, it will

be created in the user's local Temp directory, usually `C:Documents and SettingsusernameLocal SettingsTemp`

Appendix

Uninstalling

Sample XML Project File

```
<project>
  <fullName>Demo Project</fullName>
  <shortName>demo</shortName>
  <version>1.0</version>
  <installerFilename>
      ${product_shortname}-${product_version}-${platform_name}-
installer.${platform_exec_suffix}
  </installerFilename>
  <licenseFile>/home/user/installbuilder-
1.1/demo/docs/license.txt</licenseFile>
  <logoImage></logoImage>
  <postInstallationScript>
      ${installdir}/bin/postinstallation.sh ${installdir}
  </postInstallationScript>
  <projectSchemaVersion>1.0</projectSchemaVersion>
  <readmeFile>/home/user/installbuilder-
1.1/demo/docs/readme.txt</readmeFile>

<requireInstallationByRootUser>0</requireInstallationByRootUser>
  <allowComponentSelection>0</allowComponentSelection>
  <componentList>
    <component>
      <description>Default Component</description>
      <name>default</name>
      <selected>1</selected>
      <desktopShortcutList>
      <shortcut>
        <comment>Text that will appear on Tooltip</comment>
        <exec>${installdir}/bin/demo.sh</exec>
        <icon>${installdir}/bin/logo.png</icon>
        <name>Demo Project</name>
        <path></path>
      </shortcut>
      </desktopShortcutList>
      <folderList>
        <folder>
          <description>Program Files</description>
          <destination>${installdir}</destination>
          <name>programfiles</name>
          <actionList/>
          <distributionFileList>
            <distributionDirectory>
              <origin>/home/user/installbuilder-
1.1/demo/bin</origin>
            </distributionDirectory>
            <distributionDirectory>
```

```
              <origin>/home/user/installbuilder-
1.1/demo/docs</origin>
            </distributionDirectory>
            <distributionDirectory>
              <origin>/home/user/installbuilder-
1.1/demo/lib</origin>
            </distributionDirectory>
          </distributionFileList>
          <shortcutList/>
        </folder>
      </folderList>
    </component>
  </componentList>
  <fileList/>
  <parameterList>
    <directoryParameter>
      <ask>yes</ask>
      <cliOptionName>prefix</cliOptionName>
      <default>/opt/${product_shortname}-
${product_version}</default>
      <description>Installation directory</description>
      <explanation>Please specify the directory where
${product_fullname} will be installed</explanation>
      <mustBeWritable>yes</mustBeWritable>
      <mustExist>0</mustExist>
      <name>installdir</name>
      <value>${platform_install_prefix}/${product_shortname}-
${product_version}</value>
    </directoryParameter>
  </parameterList>
</project>
```

Sample Text Based Installation

```
$ ./demo-1.0-installer.bin --mode text
------------------------------------------------------------------
-----------
Welcome to the Demo Project Setup Wizard
Created with an evaluation version of BitRock InstallBuilder.

------------------------------------------------------------------
-----------
Please read the following License Agreement. You must accept the
terms of
this agreement before continuing with the installation.

Press [Enter] to continue:
Sample license


Do you accept this license? [Y/n]: y

------------------------------------------------------------------
-----------
Please specify the directory where Demo Project will be installed
```

```
Installation directory [/opt/demo-1.0]:

-------------------------------------------------------------------------
-----------
Setup is now ready to begin installing Demo Project on your
computer.

Do you want to continue? [Y/n]: y

-------------------------------------------------------------------------
-----------
Please wait while Setup installs Demo Project on your computer.

 Installing Demo Project
 0% _____________ 50% _____________ 100%
 ########################################

-------------------------------------------------------------------------
-----------
Setup has finished installing Demo Project on your computer.

View Readme file? [Y/n]: y


You have successfully installed the Demo Application!
```