BitRock InstallBuilder User Guide 5

Release 5.4.6 2008.04.18

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, Vista, Mac OS X, FreeBSD, OpenBSD, Solaris (Intel & Sparc), AIX, HP-UX, IRIX, and Linux (Intel x86/x64, Itanium, s390 & PPC).
- **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 and DEB generation (beta)** : In addition to creating native executables that can register with the RPM subsystem, BitRock InstallBuilder can generate RPM and Debian packages that can be installed using the native package management tools.
- **Optimized** : BitRock installers are optimized in size and speed and do not require a self-extraction step, reducing download, startup and installation time. Built-in LZMA support provides great compression ratios.
- **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, OpenBSD, IRIX, and Linux (Intel x86/x64, Itanium, 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. It is possible to ask for multiple parameters, like username and passwords, in the same installer screen. These helps simplify the installation process for end-users.
- **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.
- **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
- **Rollback Functionality** : BitRock installers by default perform a backup of all the files overwritten during installation, so in case there is an error, the system is recovered to its previous state.
- **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 Programs 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, Ruby, C/C++ and .NET/Mono.
- **Multiple Language Support** : BitRock installers support a variety of installation languages, including English, German, Japanese, Spanish, Italian, French, Portuguese, Traditional Chinese, Dutch, Polish, Valencian, Catalan, Estonian, Slovenian, Romanian, Hungarian, Russian and Welsh. You can specify a default language or let the user decide. Please contact us if you require additional language support.

What's new on InstallBuilder 5

In addition to the bugfixes, new actions and rules detailed in the Changelog available at **http://bitrock.com/download_installbuilder_changelog.html**, the main features for this release are:

- **LZMA support** : Optional LZMA support provides great compression ratios.
- **Improved startup speed** : Optimized startup time of installers.
- **Improved Vista integration** : Including UAC integration and recommended per-user directories.

Specifications

Supported Platforms

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.

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

- Windows 98, ME, 2000, XP, 2003, Vista
- 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 (both PA-RISC and Itanium)
- IRIX 6.5
- FreeBSD 6.x, 5.x. 4.x
- OpenBSD 3.x, 4.x
- Ubuntu Linux PPC
- Ubuntu 6.10, 7.04
- Red Hat 8.x, 9.x series
- Red Hat Enteprise Linux 3, 4, 5
- Fedora Core 1, 2, 3, 4, 5, 6, 7
- Suse 9.x series
- Suse Enterprise Server 8, 9, 10

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

Authoring Environment Requirements (Windows)

- Windows 98, ME, 2000, XP, 2003, Vista (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 (both PA-RISC and Itanium supported)
- IBM AIX 4.3 or later (including 5.x)
- IRIX SGI 6.5 or later
- FreeBSD 6.x, 5.x, 4.x
- OpenBSD 3.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.
- ia64 (Itanium) 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 distributions 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, Vista (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 6.x, 5.x, 4.x
- OpenBSD 3.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.
- ia64 (Itanium) 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.

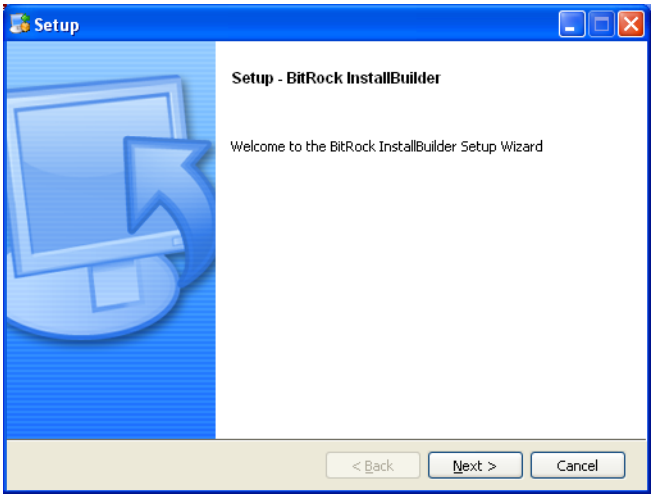
Installing BitRock InstallBuilder

Installing on Windows

You can download the BitRock InstallBuilder binary from the BitRock website. It should have a name similar to `installbuilder-professional-5.1.1-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

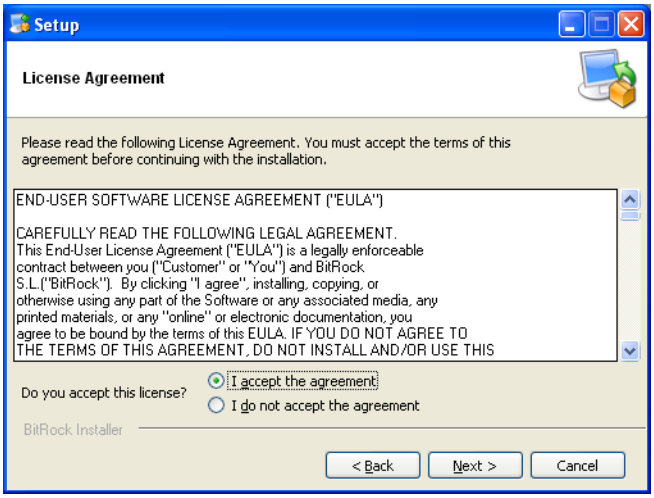
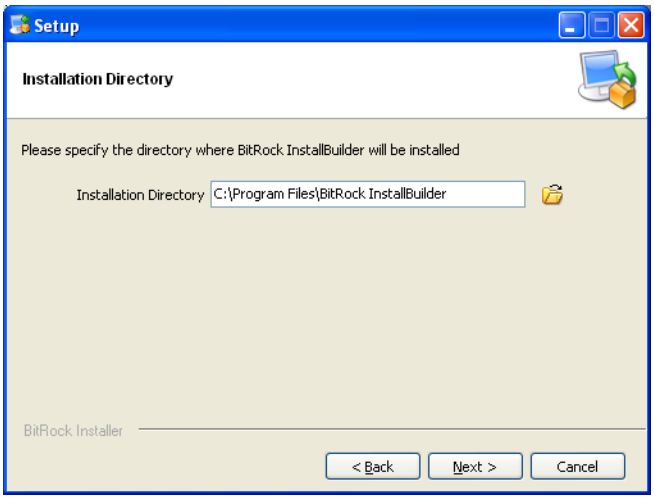


Figure 3 : Windows Select Installation Directory



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

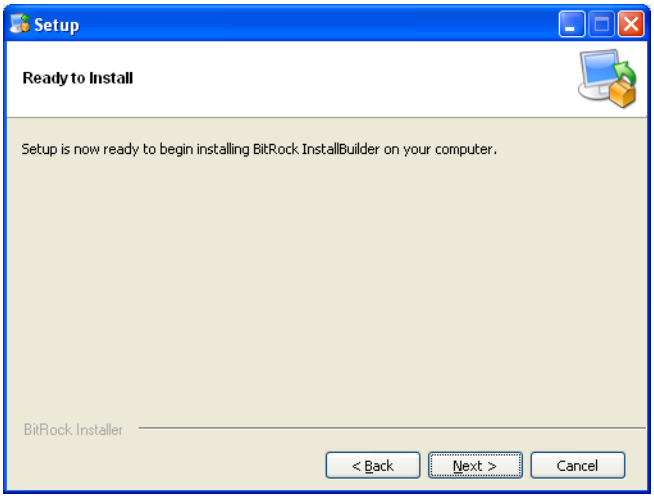


Figure 5 : Windows Installation Under Way

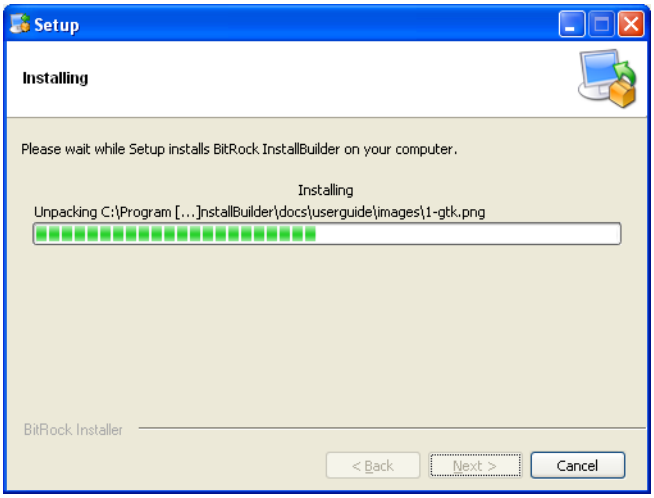
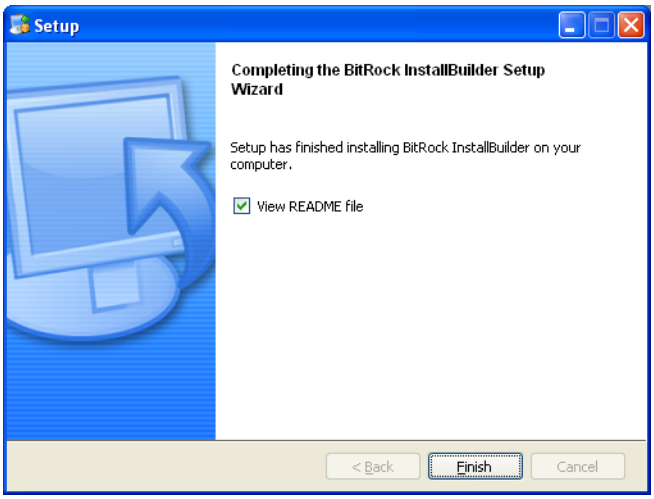


Figure 6 : Windows Installation Completed



Installing on Unix

The process for installing on Linux, OpenBSD, 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-5.1.1-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.

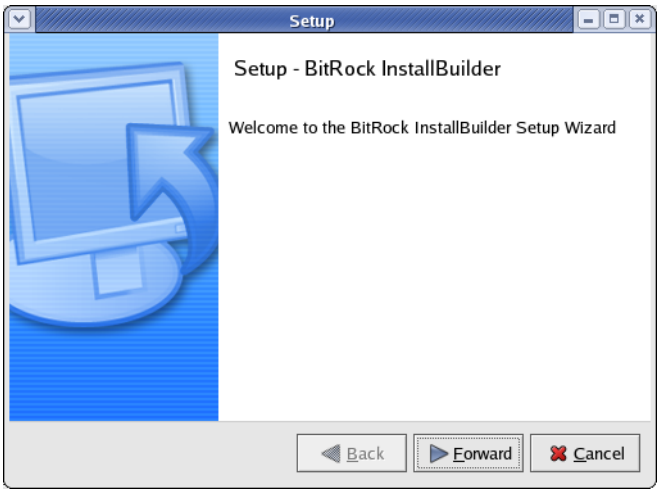
```
$
chmod 755 installbuilder-professional-5.1.1-linux-installer.bin
```

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-5.1.1-linux-installer.bin
```

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

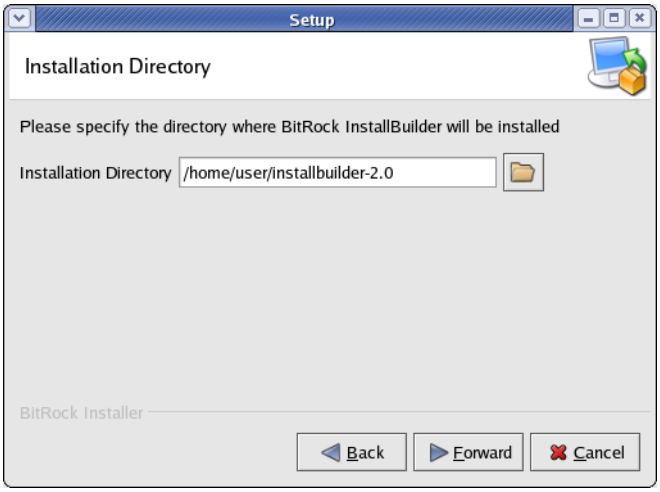
Figure 7 : Linux Welcome Screen



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-5.1.1/` 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-5.1.1/

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

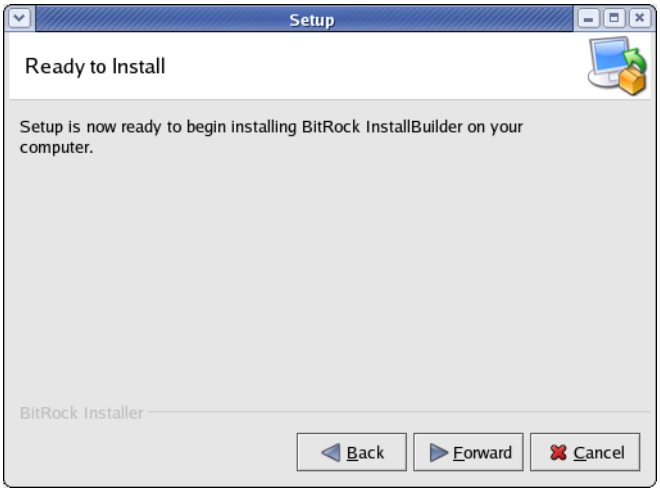


Figure 11 : Linux Installation Under Way

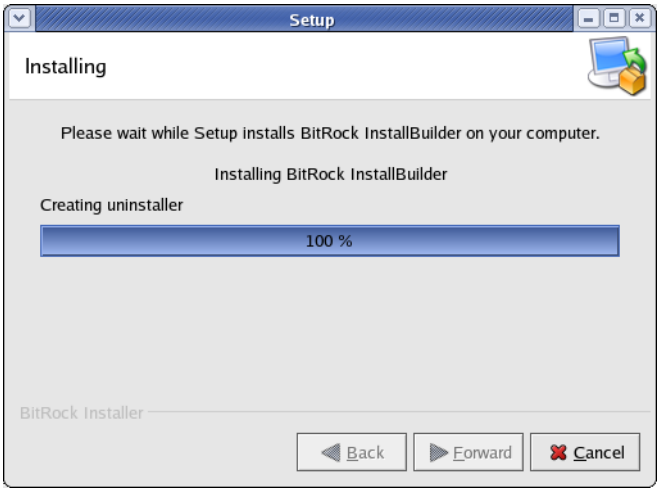
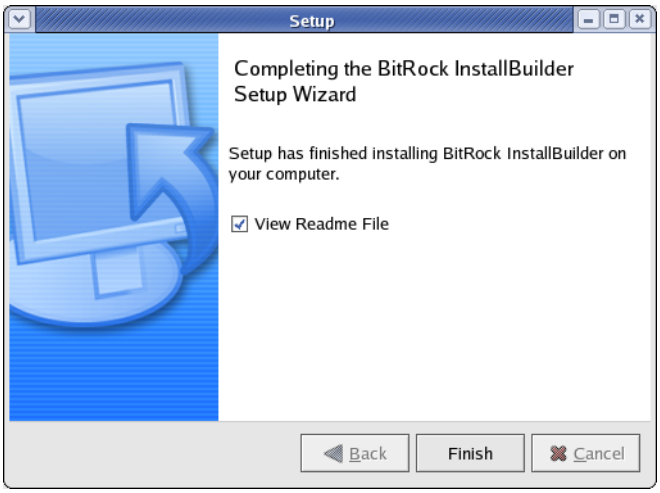


Figure 12 : Linux Installation Completed



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. See note below for Windows Vista.
- `docs`: Product documentation.
- `demo`: Files for the sample demo project.
- `output`: Finished installers. See note below for Windows Vista.

On Windows Vista, in line with the Application Development Requirements for User Account Control (UAC), the `projects` and `output` directories are installed under the user `Documents` folder, so usually they can be found at `C:\Users\user\Documents\InstallBuilder\projects` and `C:\Users\user\Documents\InstallBuilder\output`, respectively

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-5.1.1/bin/builder
```

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

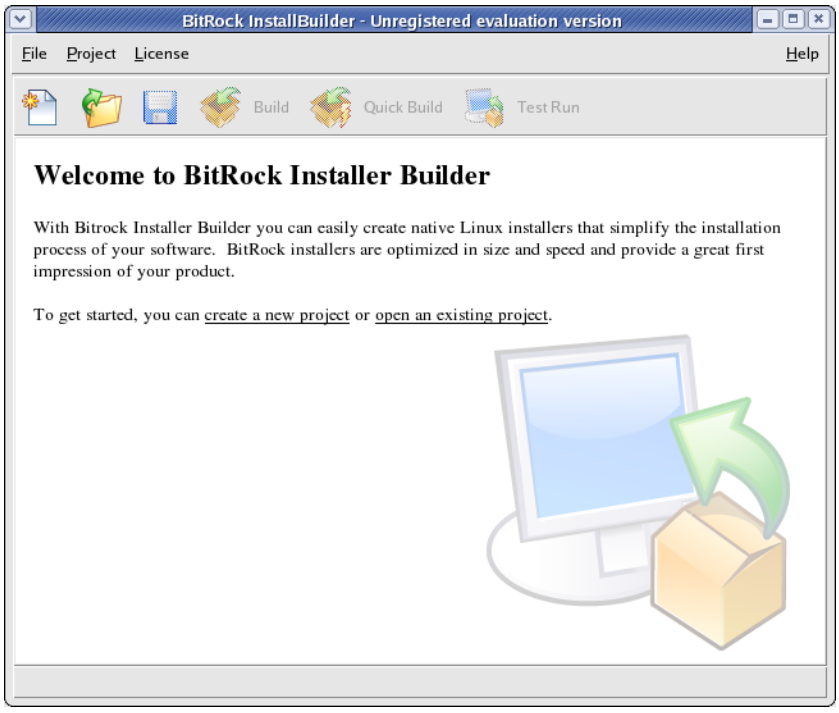
You can also build installers from the command line. Please refer to the section named "Using the Command Line Interface" later in the document.

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



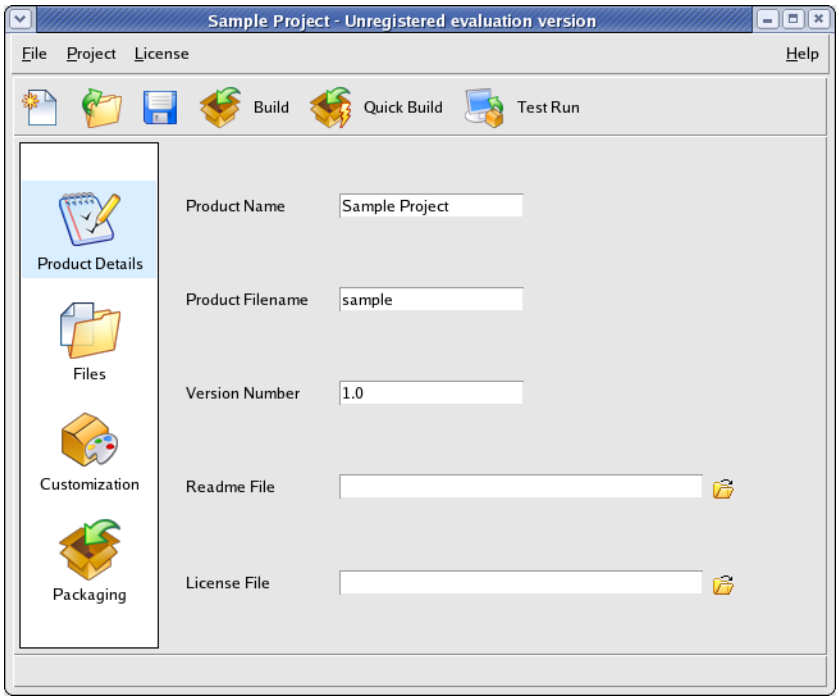
Once you enter the information, the "Basic settings screen" (Figure 14) will be shown. Here you can specify additional settings:

- **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

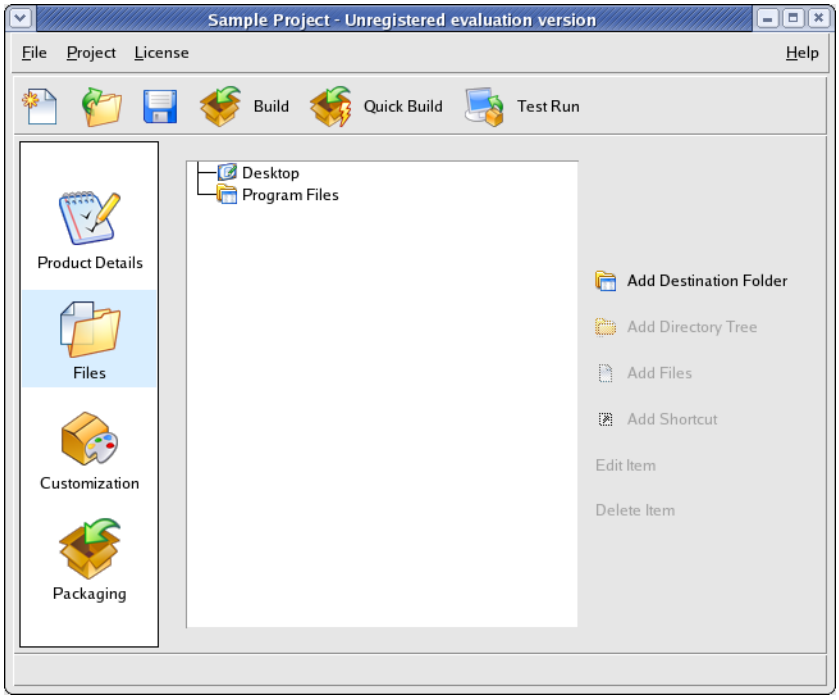


Selecting the Files

The next step is to click on the "Files" icon, which will lead 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, OpenBSD, FreeBSD, IRIX, AIX, Mac OS X, HP-UX, Solaris or Windows, it will only be included in installers for that particular platform.

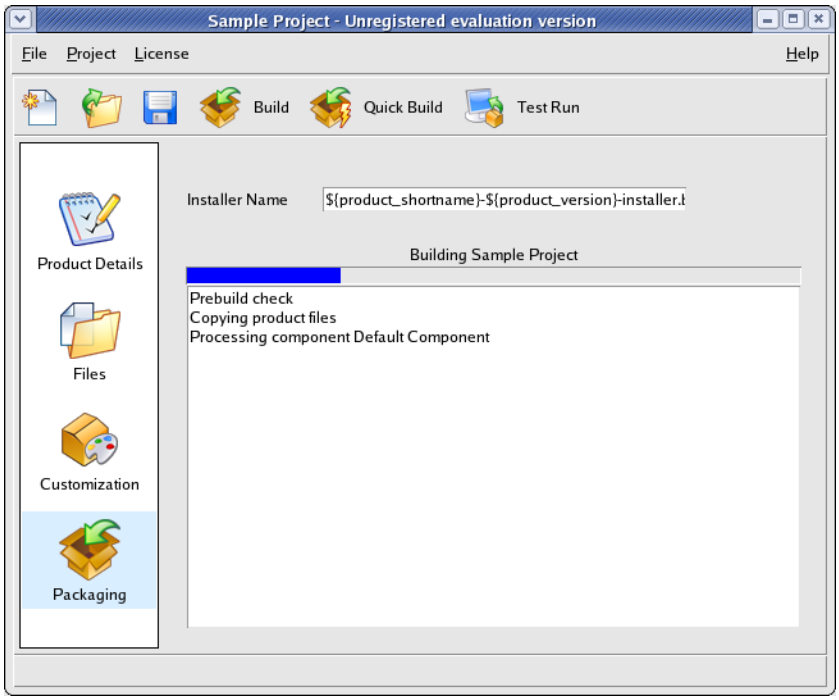
Figure 15 : Files Screen



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 (`C:\Users\user\Documents\InstallBuilder\projects` under Windows Vista, as explained earlier). 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

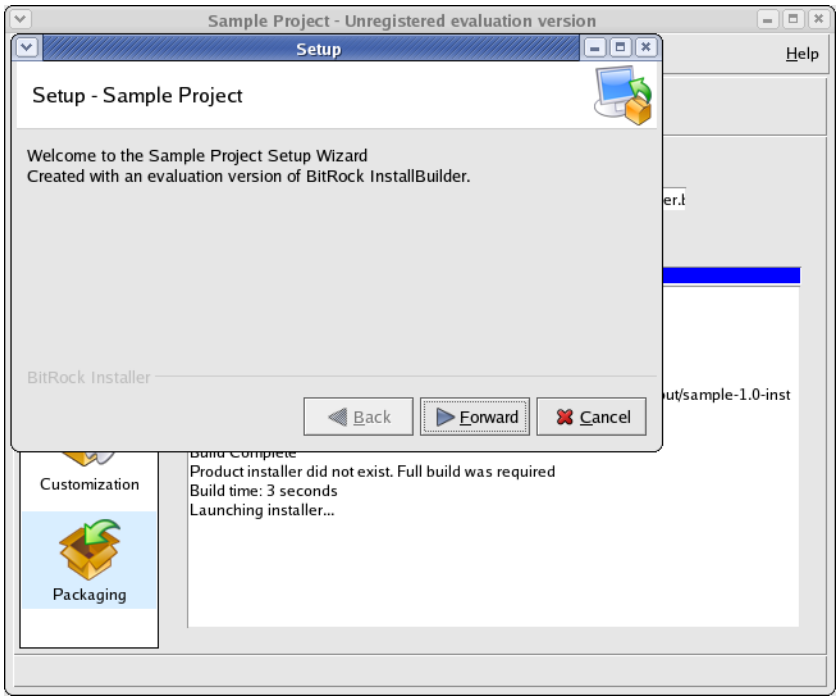


You can test the generated installer by pressing the "Test Run" button, as seen in Figure 17.

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.

Starting with InstallBuilder 5, variables are case-insentitive. This means that you can use any of the following variants, `${variablename}`, `${VariableName}` or `${VARIABLENAME}`, obtaining exactly the same value on each case.

The current version of InstallBuilder 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.
- **${platform_install_prefix}**: Platform-dependent default installation location. In Unix systems, when running as root it will be `/opt` and when running as a regular user, the home directory for that user.
- **${platform_exec_suffix}**: Platform-dependent executable file extension for the generated installer. In Unix systems it is `.bin`, on Windows it is `.exe`, and on OSX it is `.app`
- **${platform_name}**: Target platform for the installer. Currently it can be `linux`, `linux-ppc`, `linux-x64`, `linux-ia64`, `linux-s390`, `freebsd`, `freebsd4`, `freebsd6`, `freebsd6-x64`, `solaris-sparc`, `solaris-intel`, `irix-n32`, `osx`, `windows`, `aix` or `hpux`.
- **${linux_distribution}**: When the installer is running on Linux, it will contain the specific Linux flavor name. Currently, one of `debian`, `suse`, `mandrake`, `redhat`, `rhelfedora`, `slackware` or `unknown` for another distribution.
- **${installation_langcode}**: The ISO code for the language the installer was run with.
- **${installer_is_root_install}**: Whether the installer is being run as root or not.
- **${installer_ui}**: Whether the installer is being run in 'text', 'gui' or 'unattended' mode.
- **${installer_directory}**: Directory where the installer binary is located.
- **${machine_hostname}**: Machine hostname
- **${machine_ipaddr}**: Machine IP address (derived from the hostname)
- **${required_diskspace}**: Required size in KB of the files that will be installed. Only files in components that are selected for installation will be taken into account when making the calculation. Notice also that this reports the size in KB of the files, not the actual disk space that will be taken, which may vary depending on the block size of the target filesystem.
- **${system_username}**: The name of the user who is running the installer.
- **${system_temp_directory}**: Path to the system's temporary directory.
- **${user_home_directory}**: Path to the home directory of the user who is running the installer.

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 and other Unix systems 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
- **Windows Executable Icon**: ICO file with an specific format -see below- to set the icon for the installer executable file on Windows systems. The icon file can contain up to three different icons that must match one of the following formats: 16x16 pixels and 256 colors, 32x32 pixels and 256 colors, 48x48 pixels and 256 colors.
- **Default installation language**: Default language for the installer. Use 'auto' for autodetection of the current language
- **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
- **Splash screen delay**: Extra display time of the splash screen

Installer Settings

- **Require install by administrator**: Whether installation will require super user privileges (root on Linux, Administrator user on Windows and OS X). In all OSes but OS X this setting will prevent the installer from running if the user is not root or Administrator. In OS X, the regular authentication dialog window will be shown, asking the user for the administrator password so the installer can be run with root privileges
- **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
- **Compression algorithm**: Compression algorithm that will be used to pack the files inside the installer. LZMA compression is available on Linux, Windows and OS-X platforms
- **Backup Directory**: Path to a directory where old files will be stored prior to overwritting
- **Installation Scope**: Whether to install Start Menu and Desktop links for All Users or for the current user

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

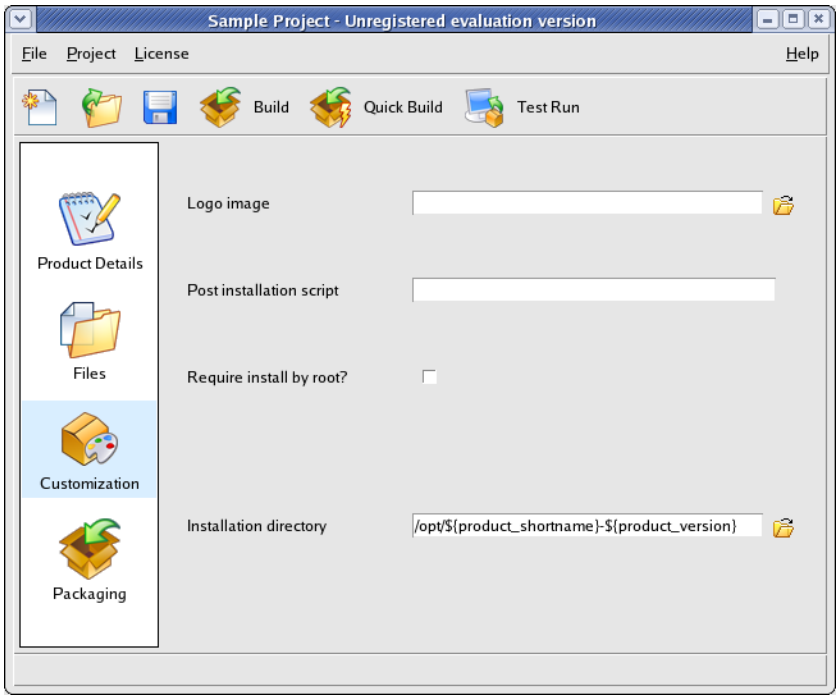
Particularly important is the **Post Install script** setting. It allows you to perform specific actions required to correctly finish installing your product, such as initializing a database or starting a server in the background.

Permissions

Please note that these options only take effect when creating installers for Unix platforms from Windows.

- **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 cp1251 cp1252 ascii macRoman unicode**.
- **licenseFileEncoding**: License file encoding. Default value: **iso8859-1**, valid values: **iso8859-1 iso8859-2 utf-8 cp1251 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).
- **installationLogFile**: This project property allows you to set an alternative path to store the installation log file. Please note that the installation log will only be written to the location specified once the installation has completed. Otherwise, it will still be available at the system's temporary directory (usually /tmp on Unix systems)
- **enableRollback**: Enable temporary backup of old files that will be saved in case of overwritting, and that will be restored if the installation fails. The implementation only handles files overwritten by the installer during the files installation step (i.e., all files specified under any of the sections). It does not support rollback for files overwritten as a result of the execution of actions or scripts. This feature is enabled by default.
- **rollbackBackupDirectory**: Path to a directory where old files will be stored prior to overwritting.
- **removeLogFile**: This project property controls automatic deletion of the generated log file after installation. It is set to 0 by default. If set to 1, the installer will remove the log file.
- **defaultInstallationMode**: Default installation mode. Available installation modes can be found by running the installer from command line using the --help option.
- **productDisplayIcon**: Application Icon (.ico format) that will be shown in Add/Remove Programs on Windows.
- **disableSplashScreen**: Disable the initial splash screen.

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

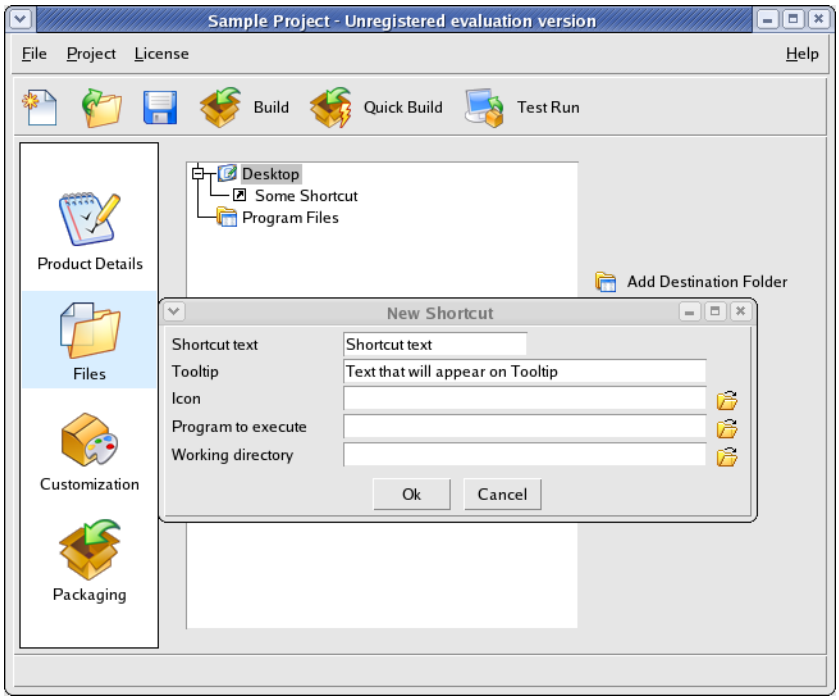
- **Windows Icon**: File containing .ico image
- **Program to execute**: Program to execute
- **Working directory**: Working directory for the program being executed

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

On Windows, Start Menu and Desktop shortcuts are by default created for All Users, or for the current user in case there are not enough privileges. InstallBuilder allows to modify this behavior via the project property <installationScope>, which can be set to "auto" (default), "user" or "allusers".

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-5.1.1/bin/builder build
/path/to/project.xml
```

For example

```
$/home/user/installbuilder-5.1.1/bin/builder build
/home/user/installbuilder-5.1.1/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-5.1.1/bin/builder build
/home/user/installbuilder-5.1.1/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-5.1.1/projects/` (or `C:\Users\user\Documents\InstallBuilder\projects` under Windows Vista). 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.

Additional information can be found in the Reference Guide bundled with the installer in the `docs/` directory and in the online FAQ at **http://bitrock.com/support_installbuilder_faq.html**

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, OpenBSD, FreeBSD, HP-UX, IRIX, AIX, Solaris and Mac OS X.

Example:

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

Exit

This action will exit the installer (or uninstaller). Its syntax is very simple.

Supported Platforms: All Platforms.

Example:

```
    <exit>
    </exit>
```

Generate Random Value

This action allows you to generate a random value, storing it in a variable. You can define the length (in characters) for the generated value.

Supported Platforms: All Platforms.

Example:

```
<generateRandomValue>
    <variable>randomvalue</variable>
    <length>8</length>
</generateRandomValue>
```

Log Message

This action allows you to write a message to the installation log: the path to the installation log file can be retrieved from the installer variable **installer_installation_log**. This feature is useful for debugging purposes.

Supported Platforms: All Platforms.

Example:

```
<logMessage>
    <text>Debug message.</text>
</logMessage>
```

MD5

This action allows you to generate a MD5 hash from a given text, storing the result into a variable.

Supported Platforms: All Platforms.

Example:

```
<md5>
    <text>text to process</text>
    <variable>md5hash</variable>
</md5>
```

Set Installer Variable From RegEx

This action allows you to set an installer variable to the result of a substitution. If the name of the variable matches a parameter name, the value of the parameter will be updated.

Supported Platforms: All Platforms.

Example:

```
<setInstallerVariableFromRegEx>
    <name>newVariable</name>
    <text>c:\a\b</text>
```

```
            <pattern>\</pattern>
            <substitution>\\</substitution>
        </setInstallerVariableFromRegEx>
```

Wait

This action will pause the installer for a given number of milliseconds.

Supported Platforms: All Platforms.

Example:

```
        <!-- wait for 5 seconds -->
        <wait>
            <ms>5000</ms>
        </wait>
```

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, OpenBSD, 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, OpenBSD, 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`.

If executed on Windows, you can select to perform the action over the system path or the current user path: if not specified, the directory will be added to the system path. This feature can be controlled using the **scope** property.

Supported Platforms: All Platforms.

Example:

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

        <!-- For Windows, allowed scope values are "user" and "system" -->
        <addDirectoryToPath>
            <scope>user</scope>
            <path>${installdir}/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, OpenBSD, 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>
```

Find File

This action allows you to define a file name or pattern (eg. "*.txt") to be searched inside a given directory and all its

subdirectories. It will save in the specified installer variable the full path to the first file that matches the provided file name or pattern.

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

Example:

```
<findFile>
    <pattern>*.txt</pattern>
    <baseDirectory>/path/to/base/directory</baseDirectory>
    <variable>textFile</variable>
</findFile>
```

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 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>
```

Please note that <throwError> will not display an error message or abort execution when run from the uninstaller. This behavior is intentional and should be interpreted as a feature, the reason being that unlike an installer, an uninstaller should never fail (for example because the user may have already manually deleted some files), so by default we disable error throwing during uninstallation. In case you need to display an error message and abort the uninstaller, one possible workaround is to combine <showWarning> and <exit> actions inside an <actionGroup>, as shown below:

```
   <preUninstallationActionList>
       <actionGroup>
           <!-- Executing two actions as if they were one -->
           <actionList>
               <showWarning>
                   <text>InstallBuilder is running, aborting uninstallation.</tex
```

```
t>
                </showWarning>
                <exit></exit>
            </actionList>
            <ruleList>
                <!-- processTest rule is currently supported for Linux, Windows an
d OSX only -->
                <processTest>
                    <name>builder</name>
                </processTest>
            </ruleList>
        </actionGroup>
    </preUninstallationActionList>
```

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

allows you to create a new registry key or 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>
```

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>
```

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>
```

registryGetKey

Store in 'variable' the first registry key that matches the given pattern, or empty otherwise. The search is case-sensitive for the whole key provided.

Properties:

variable: Variable to store result
key: Registry key
wowMode: Determines whether we want to access a 32-bit or 64-bit view of the Registry - Allowed values: **none, 64** (32,)

registryGetMatch

Store the value of the first match of a registry key matching a certain expression in an installer variable. If the key or name does not exist, then the variable will be created empty. The name can contain a wildcard expresion (using *)

Properties:

name: Entry name to read value from
variable: Variable name to store registry value to
key: Registry key
wowMode: Determines whether we want to access a 32-bit or 64-bit view of the Registry - Allowed values: **none, 64** (32,)

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>
```

Configure Properties Files

Supported Platforms: All Platforms.

This action allows you to create and modify Java-style properties files. You can specify one key and its

corresponding value. If the ini file does not exist, it will be created.

The following example adds the 'serverport' key in the property file with the value '80'.

```
            <propertiesFileSet>
                <file>/path/to/startup.conf</file>
                <key>serverport</key>
                <value>80</value>
            </propertiesFileSet>
```

The code above will create or modify an existing property file to include the following:

```
serverport=80
```

Access INI Files

Supported Platforms: All Platforms.

This action allows accessing ini-style files. You can store the value into an installer variable by referencing the key.

The following example looks for the 'port' key under the 'server' section in the ini file and stores its value in the 'portObtained' installer variable.

```
            <iniFileGet>
                <file>/path/to/initialization/file.ini</file>
                <variable>portObtained</variable>
                <key>port</key>
                <section>server</section>
            </iniFileGet>
```

Configure INI Files

Supported Platforms: All Platforms.

This action allows you to create and modify ini-style files. You can specify one key and its corresponding value, and optionally you can configure them to be placed within a given section. If the ini file does not exist, it will be created.

The following example adds the 'port' key along with the '80' value, under the 'server' section.

```
        <iniFileSet>
          <file>/path/to/initialization/file.ini</file>
          <key>port</key>
          <value>80</value>
          <section>server</section>
        </iniFileSet>
```

The code above would create a INI file or modify a pre-existing one, so it would include the following:

```
[server]
port=80
```

Obtain Disk Usage

Supported Platforms: All Platforms.

This action calculates the space on disk taken by one or multiple files or folders, and stores the result in a variable. The semicolon character is used to separate the files, and you can include the asterisk and question mark wildcard characters to match groups of files. You can also specify the size units for the returned value; allowed size units are: `KB` (Kilobytes), `MB` (Megabytes) and `GB` Gigabytes, with `KB` being the default.

The following example calculates the total disk space in megabytes taken by text and image files under a given folder.

```
        <getDiskUsage>
          <files>/some/path/*.txt;/some/path/*.jpg</files>
          <variable>totalSize</variable>
          <units>MB</units>
        </getDiskUsage>
```

Change Owner and Group of a File or Directory

Supported Platforms: Linux, OpenBSD, 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, OpenBSD, 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, OpenBSD, 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, OpenBSD, 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, OpenBSD, 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, OpenBSD, 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, OpenBSD, 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, OpenBSD, 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 about 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>
```

HTTP GET

This action allows you to perform an HTTP GET request, storing the retrieved page into a file.

Supported Platforms: All Platforms.

Example:

```
<httpGet>
    <filename>/tmp/http.html</filename>
    <url>http://www.bitrock.com</url>
</httpGet>
```

HTTP POST

This action allows you to perform an HTTP POST request, storing the retrieved page into a file.

Supported Platforms: All Platforms.

Example:

```
<httpPost>
    <url>http://www.bitrock.com/post.php</url>
    <filename>/tmp/postresult.html</filename>
    <queryParameterList>
        <queryParameter name="productname" value="${product_fullname}" />
        <queryParameter name="variable2" value="value2" />
        <queryParameter name="variable3" value="value3" />
    </queryParameterList>
</httpPost>
```

Launch Browser

This action allows you to launch a web browser pointing to a specified url.

Supported Platforms: All Platforms.

Example:

```
<finalPageActionList>
    <launchBrowser>
        <url>http://example.com/congratulations/</url>
        <progressText>Visit Example website.</progressText>
    </launchBrowser>
</finalPageActionList>
```

Read File

This action allows you to read a text file and store its contents into a variable.

Supported Platforms: All Platforms.

Example:

```
<readFile>
```

```
        <name>releasenotes</name>
        <path>/path/to/releasenotes.txt</path>
    </readFile>
```

Write File

This action allows you to store the contents of a given variable into a text file.

Supported Platforms: All Platforms.

Example:

```
    <writeFile>
        <text>${important_information}</text>
        <path>/path/to/information.txt</path>
    </writeFile>
```

Show Info

This action allows you to display a popup window with a message. It is very useful for debugging purposes.

Supported Platforms: All Platforms.

Example:

```
    <showInfo>
        <text>Debug message. myvariable = ${myvariable}</text>
    </showInfo>
```

Show Text

This action allows you to display text in a popup window. It can be useful to display the contents of a long text file.

Supported Platforms: All Platforms.

Example:

```
    <showText>
        <text>${release_notes}</text>
        <title>Release notes</title>
    </showText>
```

Show Question

This action allows you to prompt a yes/no question to the user. The result will be stored as 'yes' or 'no' in the specified variable.

Supported Platforms: All Platforms.

Example:

```
        <showQuestion>
            <text>Do you want to remove the existing database?</text>
            <variable>remove_database</variable>
        </showQuestion>
```

Show Password Question

This action allows you to ask the user to provide a password in a popup window.

Properties:

text: Question message that will be shown.
title: Dialog window title.
variable: Variable name where the password will be stored.

Supported platforms: All Platforms.

```
        <initializationActionList>
            <showPasswordQuestion>
                <text>This installer is password protected. Please
enter the passphrase</text>
                <variable>password</variable>
            </showPasswordQuestion>
            <throwError>
                <text>Incorrect Password</text>
                <ruleList>
                    <compareText>
                        <text>${password}</text>
                        <value>secret123</value>
                        <logic>does_not_equal</logic>
                    </compareText>
                </ruleList>
            </throwError>
        </initializationActionList>
```

Show Warning

This action allows you to display a message in a warning dialog.

Properties:

text: Warning message that will be shown.

Supported platforms: All Platforms.

```
      <showWarning>
          <text>${product_fullname} requires at least 200MB
of freedisk space. Currently, there are only
${available_disk_space}MB left.</text>
          <ruleList>
              <compareValues>
                  <value1>${available_disk_space}</value1>
                  <logic>less</logic>
                  <value2>200</value2>
              </compareValues>
          </ruleList>
      </showWarning>
```

Remove Directory from PATH

This action allows you to remove a directory from the system PATH. For Windows, you can choose to modify the system path or the user path using the **scope** property.

Properties:

scope: Select user path or system path. - Allowed values: **system, user** (System, User)
path: Path to the directory

Supported platforms: All Platforms.

```
    <removeDirectoryFromPath>
         <path>${installdir}/bin</path>
    </removeDirectoryFromPath>
```

Add Fonts

This action allows you to install fonts in Windows systems.

Properties:

files: Files to apply action to.

Supported platforms: Windows.

```
      <addFonts>
          <files>${installdir}/Fonts/font1.ttf</files>
      </addFonts>
```

Remove Fonts

This action allows you to remove fonts on Windows. The action accepts only file names or patterns (as opposed to file paths, either relative or absolute), matching them inside the system fonts folder.

Properties:

files: Files to apply action to.

Supported platforms: Windows.

```
<removeFonts>
    <files>font1.ttf;*userfont*.ttf;font2.ttf</files>
</removeFonts>
```

Add Files to Uninstaller

This action allows you to add new files to the uninstaller, so they will be removed during the uninstallation process.

Properties:

files: Files to apply action to.

Supported platforms: All Platforms.

```
<addFilesToUninstaller>
    <files>${installdir}/profiles/*;${installdir}/userdata/*</files>
</addFilesToUninstaller>
```

Remove Files from Uninstaller

This action allows you to remove files from the uninstaller, so they will not be removed during the uninstallation process.

Properties:

files: Files to apply action to.

Supported platforms: All Platforms.

```
<removeFilesFromUninstaller>
    <files>${installdir}/commonfiles/profiles.inf</files>
</removeFilesFromUninstaller>
```

Delete Group

This action allows you to remove a group from the system.

Properties:

groupname: Group name to delete.

Supported platforms: Linux, OpenBSD, FreeBSD, HP-UX, AIX, Solaris.

```
<deleteGroup>
    <groupname>developers</groupname>
</deleteGroup>
```

Add Shared DLL

This action allows you to increment the reference count for a shared DLL.

Properties:

path: Path to the shared DLL

Supported platforms: Windows.

```
<addSharedDLL>
    <path>${installdir}/libraries/mylib.dll</path>
</addSharedDLL>
```

Remove Shared DLL

This action allows you to decrement the reference count for a shared DLL. If it reaches zero, the file will be removed.

Properties:

path: Path to the shared DLL

Supported platforms: Windows.

```
<removeSharedDLL>
    <path>${installdir}/libraries/mylib.dll</path>
</removeSharedDLL>
```

Add Unix Service

This action allows you to create a new service in a GNU/Linux based system. Notice you will need to run the installer as root to be able to create new services.

Properties:

program: Path to the program
description: Product description
name: Service Name

Supported platforms: Linux (RedHat/Debian based distributions, others may -or may not- work).

```
<addUnixService>
```

```
        <name>customservice</name>
        <description>custom service</description>
        <program>${installdir}/bin/myservice</program>
    </addUnixService>
```

Create Windows Service

This action allows you to create a new Windows service.

Properties:

description: Program description
program: Path to program
startType: Specify how the service should be started - Allowed values: **auto, manual, disabled** (Auto, Manual, Disabled)
programArguments: Arguments to pass to the program
dependencies: Comma separated list of services that the created service depends on
serviceName: Internal service name
displayName: Name displayed in Windows services control panel

Supported platforms: Windows.

```
    <createWindowsService>
        <program>${installdir}/bin/customservice.exe</program>
        <serviceName>myservice</serviceName>
    </createWindowsService>
```

Delete Windows Service

This action allows you to remove a specified Windows service.

Properties:

serviceName: Internal service name
displayName: Name displayed in Windows services control panel

Supported platforms: Windows.

```
    <deleteWindowsService>
        <serviceName>myservice</serviceName>
    </deleteWindowsService>
```

Get Unique Windows Service Name

This action allows you to get a unique Windows service name.

Properties:

separator: Separator string, it is '-' by default
selectedServiceNameVariable: Variable to store the Service name
selectedDisplayNameVariable: Variable to store the service display name
serviceName: Initial name for the service
displayName: Initial display name for the service

Supported platforms: Windows.

```
        <getUniqueWindowsServiceName>
            <serviceName>myservice</serviceName>
            <displayName>My Service</displayName>
            <selectedServiceNameVariable>unique_servicename</selectedServiceNameVa
riable>
            <selectedDisplayNameVariable>unique_displayname</selectedDisplayNameVa
riable>
        </getUniqueWindowsServiceName>
```

Restart Windows Service

This action allows you to restart a specified Windows service.

Properties:

delay: Amount of milliseconds to wait for the service to start.
serviceName: Internal service name
displayName: Name displayed in Windows services control panel

Supported platforms: Windows.

```
        <restartWindowsService>
            <serviceName>myservice</serviceName>
        </restartWindowsService>
```

Start Windows Service

This action allows you to start a specified Windows service.

Properties:

delay: Amount of milliseconds to wait for the service to start.
serviceName: Internal service name
displayName: Name displayed in Windows services control panel

Supported platforms: Windows.

```
        <startWindowsService>
            <serviceName>myservice</serviceName>
        </startWindowsService>
```

Stop Windows Service

This action allows you to stop a specified Windows service.

Properties:

serviceName: Internal service name
displayName: Name displayed in Windows services control panel

Supported platforms: Windows.

```
      <stopWindowsService>
          <serviceName>myservice</serviceName>
      </stopWindowsService>
```

Create Windows File Associations

This action allows you to create file associations for Windows, defining commands such as "open" for a given file extension.

Properties:

extensions: Space-separated list of extensions for which the given commands will be available.
icon: Path to the icon file that contains the icon to display.
progID: Programmatic Identifier to which the extensions are attached, contains the available commands to be invoked on each file type.
mimeType: MIME type associated to all the file extensions.
friendlyName: Friendly Name for the progID.
commandList: List of commands that can be invoked on each given file type.

Supported Platforms: Windows.

Example:

```
  <associateWindowsFileExtension>
      <extensions>.ext</extensions>
      <progID>mycompany.package</progID>
      <icon>${installdir}\images\myicon.ico</icon>
      <mimeType>example/mycompany-package-ext</mimeType>
      <commandList>
          <!-- Defining the "open" command -->
          <command>
              <verb>open</verb>
              <runProgram>${installdir}\yourprogram.exe</runProgram>
              <runProgramArguments>%1</runProgramArguments>
          </command>
      </commandList>
  </associateWindowsFileExtension>
```

Create Directory

This action allows you to create a new directory.

Properties:

path: Path to the new directory

Supported platforms: All Platforms.

```
<postInstallationActionList>
    <createDirectory>
        <path>${installdir}/userdata</path>
    </createDirectory>
</postInstallationActionList>
```

Unpack File

This action allows you to unpack a file before files are unpacked during the installation phase. This can be helpful to extract files to a temporary folder such as ${env(TEMP)} if you need to run a pre-installation check script or program. If you need to unpack a directory, you may want to try using **unpackDirectory** action instead.

Properties:

origin: File name you want to extract
folder: Project folder name where where the file you want to extract is located
destination: Path to the location where you want to extract the file
component: Project component where the file you want to extract is located

Supported platforms: All Platforms.

```
<unpackFile>
    <component>default</component>
    <destination>${env(TEMP)}/test.exe</destination>
    <folder>programfileswindows</folder>
    <origin>test.exe</origin>
</unpackFile>
```

Unpack Directory

This action allows you to unpack a directory before files are unpacked during the installation phase. This can be helpful to extract a full directory to a temporary folder such as ${env(TEMP)}. If you need to unpack single files, you may want to try using **unpackFile** action instead.

Properties:

origin: Directory name you want to extract.
folder: Project folder name where the directory you want to extract is located.
destination: Path to the location where you want to extract the directory.
component

: Project component where the directory you want to extract is located.

Supported Platforms: All Platforms.

Example:

```
        <unpackDirectory>
            <component>default</component>
            <destination>${env(TEMP)}/directorytounpack</destination>
            <folder>programfileswindows</folder>
            <origin>directorytounpack</origin>
        </unpackDirectory>
```

Modify a String

This action allows you to transform a given text using one of the allowed string manipulation methods.

Properties:

text: Text which will be transformed.
logic: Transformation to perform. - Allowed values: **toupper, tolower, totitle, trimleft, trimright, trim** (Convert to uppercase, Convert to lowercase, Convert the first character to uppercase, Remove leading whitespace, Remove trailing whitespace, Remove leading and trailing whitespace)
variable: Variable name which will store the result.

Supported Platforms: All Platforms.

Example:

```
    <stringModify>
        <text>this text will be converted to uppercase</text>
        <logic>toupper</logic>
        <variable>newtext</variable>
    </stringModify>
```

Add Choice Options

This action allows you to add new options inside an existing **choiceParameter**. By including a ruleList, you can limit the action so it will be executed only if a particular condition is met. If you are populating a choiceParameter dynamically, you may want to remove all the options using the **removeChoiceOptions** action before using **addChoiceOptions**: otherwise, if you go back and forth in the installer page, the options will be added over and over again.

Supported Platforms: All Platforms.

Example:

```
        <addChoiceOptions>
            <name>choiceparameter_name</name>
```

```
        <optionList>
            <option text="Option 1" value="option1"/>
            <option text="Option 2" value="option2"/>
        </optionList>
    </addChoiceOptions>
```

Remove Choice Options

This action allows you to remove specific options from a given choiceParameter. The options to remove can be specified as: **option1,option2,...,optionX**.

If not specified, all options will be removed. This is useful, for example, if you are populating a choiceParameter dynamically: In that case, the choiceParameter can leave the **options** property empty, in order to remove all options.

Supported Platforms: All Platforms.

Example:

```
    <removeChoiceOptions>
        <name>choiceparameter_name</name>
        <options>option1,option3</options>
    </removeChoiceOptions>
```

Component Selection

This action allows you to select or deselect components. The syntax is quite straightforward. Remember a **ruleList** inside any action can be used to execute the action only if a particular condition is met.

Supported Platforms: All Platforms.

Example:

```
    <!-- To enable a component -->
    <componentSelection>
        <select>component1,component2</select>
    </componentSelection>

    <!-- To disable a component -->
    <componentSelection>
        <deselect>component1,component2</deselect>
    </componentSelection>
```

Create Timestamp

This action allows you to create a timestamp using a custom format, storing the result in an installer variable.

Properties:

format: Format string for the generated timestamp. The string allows a number of field descriptors.
variable: Variable that will store the resulting timestamp.

The format string will allow a number of field descriptors to generate the timestamp. A list of currently supported field descriptors follows:

- **%%**: Insert a %.
- **%a**: Abbreviated weekday name (Mon, Tue, etc.).
- **%A**: Full weekday name (Monday, Tuesday, etc.).
- **%b**: Abbreviated month name (Jan, Feb, etc.).
- **%B**: Full month name.
- **%c**: Locale specific date and time.
- **%d**: Day of month (01 - 31).
- **%H**: Hour in 24-hour format (00 - 23).
- **%I**: Hour in 12-hour format (00 - 12).
- **%j**: Day of year (001 - 366).
- **%m**: Month number (01 - 12).
- **%M**: Minute (00 - 59).
- **%p**: AM/PM indicator.
- **%S**: Seconds (00 - 59).
- **%U**: Week of year (01 - 52), Sunday is the first day of the week.
- **%w**: Weekday number (Sunday = 0).
- **%W**: Week of year (01 - 52), Monday is the first day of the week.
- **%x**: Locale specific date format.
- **%X**: Locale specific time format.
- **%y**: Year without century (00 - 99).
- **%Y**: Year with century (e.g. 1990).
- **%Z**: Time zone name.

Supported Platforms: All Platforms.

Example:

```
<!-- Timestamp will look like "Mon Feb 11, 15:05:34 2008" -->
<createTimestamp>
    <variable>timestamp</variable>
    <format>%a %b %d, %H:%M:S %Y</format>
</createTimestamp>
```

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.

Pre Build Actions

You can specify a <preBuildActionList> section in the project file. It can include a list of actions to execute before generating the installer file. These actions usually include setting environment variables, or performing any kind of processing on the files that will go into the installer before they are packed into it. For multi-platform CDROM installers, the preBuildActionList is executed once at the beggining of the CDROM build, and then again for every one of the specific platform installers.

Post Build Actions

You can specify a <postBuildActionList> section in the project file. It can include a list of actions to execute after generating the installer file. These actions are usually useful to revert any changes made to the files during the preBuildActionList, or to perform additional actions on the generated installer, like signing it by invoking an external tool. For multi-platform CDROM installers, the postBuildActionList is executed once for every one of the specific

platform installers and then it is executed once final time for the whole CDROM build.

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 an 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</ex
planation>
     <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.
- **leftImage**: When using **<style>custom</style>** inside the **<project>** tag in your project file, it displays a custom PNG or GIF image at the left side of the installer page associated to this

parameter. Its purpose is the same as the **<leftImage>** property inside the **<project>** tag, but allows you set a different image for each parameter page.

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.</explana
tion>
    </stringParameter>
```

Label Parameter

The label parameter allows you to display a string of read-only text inside an installer page. Optionally, you can include an image to the left side of the text.

Example:

```
    <labelParameter>
        <name>label</name>
        <title>labelParameter test</title>
        <description>This is a warning message inside an installer page.</descript
ion>
        <image>/path/to/icons/warning.png</image>
    </labelParameter>
```

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 ${product_fullname} wi
ll be installed</explanation>
            <default>${platform_install_prefix}/${product_shortname}-${product_ver
sion}</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?</explana
tion>
      <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 p
rovide (...)</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** will be 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>masterpassword</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. Please note that parameter groups also need to contain a <name> tag.

Example:

```
   <parameterGroup>
     <name>userandpass</name>
     <explanation>Please enter the username and password for your database.</expla
nation>
     <parameterList>
     <stringParameter>
        <name>username</name>
        <default>admin</default>
        <description>Username</description>
      </stringParameter>
      <passwordParameter>
        <ask>yes</ask>
        <name>masterpass</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 <ruleEvaluationLogic>or</ruleEvaluationLogic> 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> etc. 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> Text length 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, a Mac OS X binary, a FreeBSD 4.x/5.x binary, an HP-UX 11 binary, an IRIX binary, an AIX binary and a Linux PPC, x64, x86, Itanium 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, OpenBSD, 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 restore the executable permissions with:

```
$
chmod u+w installbuilder-professional-5.1.1-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, OpenBSD, 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 explicitly 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 passing 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, OpenBSD, 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 Settings\username\Local Settings\Temp`

Appendix

Uninstalling

Sample XML Project File

```
<project>
  <fullName>Demo Project</fullName>
  <shortName>demo</shortName>
  <version>1.0</version>
  <installerFilename>
      ${product_shortname}-${product_version}-${platform_name}-installer.${platfor
m_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}</v
alue>
    </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!
```