ej-technologies

install4j Manual

© 2008 ej-technologies GmbH. All rights reserved.

Index

install4j help 5

Licensing 6

A Help topics 7
A.1 Concepts 7
A.1.1 Projects 7
A.1.2 File sets and components 9
A.1.3 Screens and actions 11
A.1.4 Form screens 14
A.1.5 Variables 17
A.1.6 VM parameters 22
A.1.7 JRE bundles 25
A.1.8 Auto-update functionality 28
A.2 Generated installers 32
A.2.1 Installer modes 32
A.2.2 Command line options 34
A.2.3 Response files 36
A.2.4 JRE search 38
A.2.5 Updates 40
A.2.6 Error handling 42
A.3 Extending install4j 44
A.3.1 Using the install4j API 44
A.3.2 Extensions 47

B Reference 49
B.1 Configuration steps 49
B.2 Step 1: General Settings 50
B.2.1 Overview 50
B.2.2 Application info 51
B.2.3 Java version 52
B.2.4 Languages 54
B.2.5 Media file options 56
B.2.6 Compiler variables 58
B.2.7 Project options 59
B.2.8 Dialogs 60
B.2.8.1 Search sequence dialog 60
B.2.8.2 Language selection dialog 60
B.2.8.3 Variables selection dialog 60
B.2.8.4 Compiler variables edit dialog 61
B.2.8.5 Input dialog 61
B.2.8.6 Configure JDKs dialog 61
B.3 Step 2: Files 63
B.3.1 Overview 63
B.3.2 Defining the distribution tree 64
B.3.2.1 Overview 64
B.3.2.2 File wizard 66
B.3.2.3 Wizard steps 68
B.3.2.3.1 Select directory 68
B.3.2.3.2 Select files 68
B.3.2.3.3 Install options 68
B.3.2.3.4 Uninstall options 70

B.3.2.3.5 Exlude files and directories ... 71
B.3.2.3.6 Exlude suffixes ... 71
B.3.3 Viewing the results ... 72
B.3.4 Defining installation components ... 73
B.3.5 Dialogs ... 75
B.3.5.1 Distribution file chooser dialog ... 75
B.3.5.2 Folder properties dialog ... 75
B.4 Step 3: Launchers ... 76
B.4.1 Overview ... 76
B.4.2 Launcher wizard ... 78
B.4.3 Wizard steps ... 79
B.4.3.1 Executable ... 79
B.4.3.2 Icon ... 81
B.4.3.3 Java invocation ... 82
B.4.3.4 Splash screen ... 84
B.4.3.5 Advanced options ... 85
B.4.3.5.1 Redirection ... 85
B.4.3.5.2 Service options ... 86
B.4.3.5.3 Windows version info ... 88
B.4.3.5.4 Vista execution level ... 89
B.4.3.5.5 UNIX launcher script ... 90
B.4.3.5.6 Menu integration ... 91
B.4.3.5.7 Native libraries ... 92
B.4.3.5.8 Preferred VM ... 93
B.4.3.5.9 Splash screen options ... 94
B.4.3.6 Dialogs ... 96
B.4.3.6.1 Main class selection dialog ... 96
B.4.3.6.2 Class path dialog ... 96
B.4.3.6.3 Native libraries entry dialog ... 97
B.4.3.6.4 Visual positioning ... 97
B.5 Step 4: Installer ... 98
B.5.1 Overview ... 98
B.5.2 Screens & actions ... 99
B.5.3 Configuring applications ... 102
B.5.4 Configuring screens ... 105
B.5.5 Available screens ... 106
B.5.6 Configuring actions ... 117
B.5.7 Available actions ... 118
B.5.8 Screen and action groups ... 142
B.5.9 Configuring form components ... 143
B.5.10 Available form components ... 145
B.5.11 Custom code ... 168
B.5.12 Update options ... 169
B.5.13 Dialogs ... 170
B.5.13.1 Custom code entry ... 170
B.5.13.2 Class selector dialog ... 170
B.5.13.3 Registry dialog ... 170
B.5.13.4 Application template dialog ... 171
B.5.13.5 String edit dialog ... 171
B.5.13.6 Java code editor ... 172
B.5.13.7 Java editor settings ... 174
B.5.13.8 Code gallery ... 175
B.5.13.9 Key map editor ... 175
B.6 Step 5: Media ... 177

B.6.1 Overview 177
B.6.2 Media file types 178
B.6.3 Media file wizard 180
B.6.4 Wizard steps 181
B.6.4.1 Platform 181
B.6.4.2 Installer options 182
B.6.4.3 Data files 184
B.6.4.4 Bundled JREs 186
B.6.4.5 Customize project defaults 188
B.6.4.6 32-bit or 64-bit (Windows) 190
B.6.4.7 Code signing (Windows) 191
B.6.4.8 Launcher (Mac OS X single bundle) 192
B.7 Step 6: Build 193
B.7.1 Overview 193
B.7.2 Build options 193
B.8 JRE download wizard 194
B.9 JRE bundle wizard 195
B.10 Preferences 196
B.11 Command line compiler 197
B.11.1 Overview 197
B.11.2 Options 198
B.11.3 Using install4j with ant 201
B.11.4 Relative resource paths 203

Welcome to install4j

Thank you for choosing install4j. To help you get acquainted with install4j's features, this manual is divided into two sections:

- Help topics [p. 7]

  Help topics present important concepts in install4j. They are not necessarily tied to a single configuration step. Help topics are recommended reading for all install4j users.

  The help topics section does not cover all aspects of install4j. Please turn to the reference section for an exhaustive explanation of all features that can be found in install4j.

- Reference [p. 49]

  The reference section covers all configuration step, all dialogs and all features of install4j. It is highly hierarchical and not optimized for systematic reading.

  The reference section is the basis for install4j's context sensitive help system. Each configuration step and each dialog have one or more corresponding items in the reference section.

We appreciate your feedback. If you feel that there's a lack of documentation in a certain area of if you find inaccuracies in the documentation, please don't hesitate to contact us at support@ej-technologies.com.

install4j Licensing

With a 60-day evaluation license you can integrate install4j into your build process before purchasing it. The evaluation period can be renewed until you actually start distributing installers.

install4j licenses can be purchased easily and securely online. We accept a large variety of payment methods including credit cards, checks and purchase orders. Pricing information is available online.

install4j licenses are either

- **Per-developer licenses**

  With one license a single user is allowed to install install4j on multiple machines.

  For automated builds, you have to ensure that the install4j IDE is not running when running your build, otherwise the GUI will terminate and the build will fail.
- **Floating licenses**

  A floating license purchase includes a license server which allows a maximum concurrent user count. An arbitrary number of developers may install install4j.

  A floating license includes an **unlimited number of automated build agents**.

install4j comes in two editions

- **Windows Edition**

  This edition can only generate installers for Microsoft Windows. The install4j IDE and the command line compiler themselves can run on other supported platforms as well.
- **Multi-Platform Edition**

  This edition can generate installers for all supported platforms.

Please read the included file `license.txt` to learn more about the scope of the license. For licensing questions, please contact sales@ej-technologies.com.

You can enter your license key by invoking *Help->Enter license key* from install4j's main menu. To make it easier for you to enter the license key, you can use the **[Paste from clipboard]** button, after copying any text fragment which contains the license key to your system clipboard. If a valid license key can be found in the clipboard content, it is extracted and displayed in the dialog.

If a license has been entered, the licensing information is visible in the about dialog (*Help->About install4j*). The install4j command line compiler [p. 197] also prints licensing information except when invoked with the quiet option [p. 198] .

Your license contains the information whether is is a license for the Multi-Platform or Windows edition. If the evaluation mode is different than the scope of your license, you will have to restart install4j.

A Help topics

A.1 Concepts

A.1.1 install4j Projects

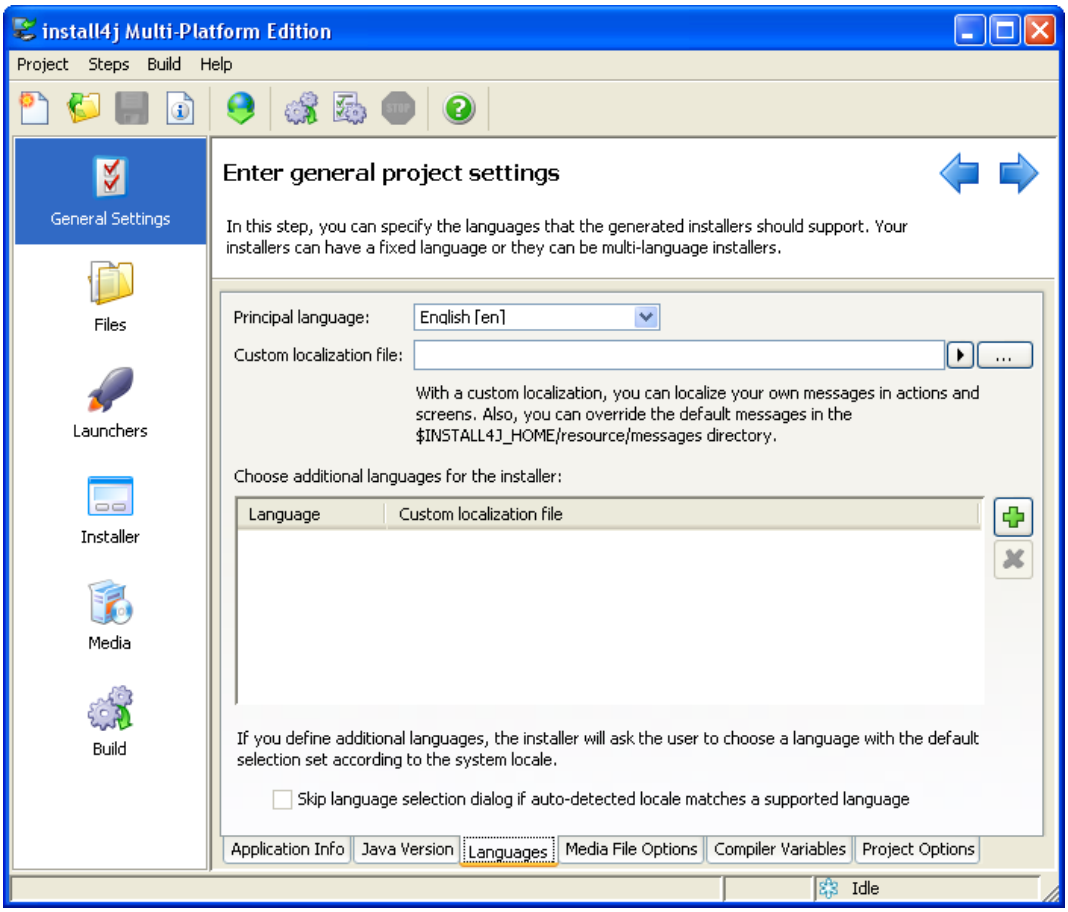
Project files

A project in install4j is the collection of all information required to build media files. A project is saved to a single XML file with an `.install4j` extension. Project files are platform-independent, you can open and compile them on any supported platform. Any paths that you enter in install4j are saved as absolute paths by default. This allows you to move the project file to a different location on your computer and the compilation will still work. If you wish to use your project file on multiple computers or platforms or compile your launchers by automatic build agents, it is more convenient to use relative paths. install4j provides an option to convert all paths to relative paths [p. 59] when you save your project.

install4j keeps a list of recently opened projects under *Project->Reopen*. By default, install4j opens the last project on startup. This behavior can be changed in the preferences dialog [p. 196] . You can pass the name of a project file as a command line parameter to install4j to open it on startup. Also, the command line compiler [p. 197] expects the project file name as its argument.

Contents of a project

The following paragraphs give a high-level overview of the elements that you can configure in install4j. Each of the configuration sections in install4j as seen in the screenshot below represents a top-level concept in install4j.



Typically, a project defines the distribution of a single application. An application has an automatically generated application ID [p. 169] that allows installers to recognize previous installations.

At the core of the project definition is the sequence of installer screens and actions [p. 11] . They determine what the users see, what information they can enter and what the installer does. install4j offers a lot of flexibility regarding the configuration of of your installer. Besides creating traditional application installers, install4j is equally suited to create small applications that modify the target system in some way. The install4j runtime is localized into many languages. You can configure your installers to support one or multiple languages [p. 54] .

Most installers install files to a dedicated directory and optionally to several existing directories on the target computer. That's what the "Files" section [p. 63] in the install4j IDE is for. Here, you define a "distribution tree", and optionally "installation components" which can also be downloaded on demand [p. 184] . The actual installation of these files is handled by a special action (the "Install files" action) which is part of the default project template. If your installers should not install any files, you can remove that action and ignore the "Files" configuration section. When the the "Install files" action is executed, it creates an uninstaller. The uninstaller offers the same flexibility as the installer and is configured in the same way.

Unless the installed files are only static data, you will need application launchers to allow the user to start your application. You can define one or several application launchers in the "Launchers" section [p. 76] . Launchers generated by install4j have a rich set of configuration options including an optional splash screen or advanced features like a single instance mode. Configured launchers can also be "services" that run independently of logged-on users. install4j offers special installation screens and actions for services.

install4j has many advanced features concerning the runtime-detection or bundling of JREs. You define Java version constraints and a search sequence [p. 52] for both your installers and your generated launchers. In this way, you ensure that the launchers run with the same JRE as your installers. Bundling of JREs is configured on a per-media set basis [p. 186] and includes an optional on-demand download of a JRE.

Finally, the media file definitions define the actual executables that you distribute. They capture platform-specific information and provide several ways to override project settings. You typically define one media file for each platform. Multiple media files for the same platform can be added as needed. Media files are either installers or archives. Archives simply capture the launchers and the distribution tree. Archives are a limited way to create a distribution and might not be suitable if you rely on the flexibility that is offered by installers.

Project reports

A project, and especially the definition of the installer and uninstaller, is very hierarchical and possibly quite complex. In order to check all your projects settings on a single page, or to print out your project definition, install4j offers a project report. This action is available from the menu and toolbar. When you generate a report, an HTML file is written to disk. In addition, a directory named `install4j_images` is created which holds all required icons. The export directory for project reports is remembered across restarts of install4j. install4j will suggest a file name based on the project name. If that file already exists, a number will be appended to make the file name unique.

A.1.2 File Sets and Installation Components

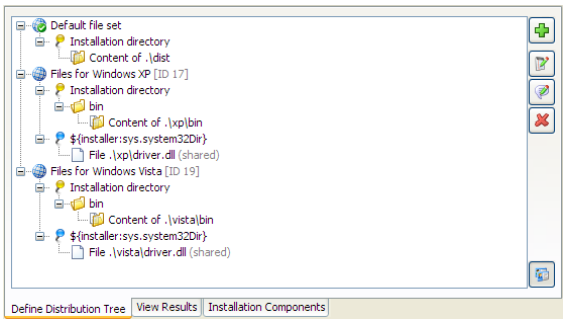
Introduction

install4j offers two mechanisms to group files: File sets and installation components. File sets are configured in the distribution tree and can be used in a variety of use cases as building blocks for your installers. Installation components are presented to the user at runtime and mark certain parts of the distribution tree that have to be installed if the user chooses an installation component.

Both file sets and installation components are optional concepts that can be ignored if they are not required for an installer project: There is always a "Default file set" to which you can add files in the distribution tree and on the installation components tab you do not have to add any components.

File sets

File sets are a way to group files in the distribution tree. When you need to select files in other parts of the install4j IDE, you can select the file set node instead of selecting single files and directories. Each file set has a special "Installation directory" child node that maps to the installation directory selected by the user at run time. Custom installation roots are defined separately for different file sets. If you require the same installation root in two different file sets, you simply define the same root twice.



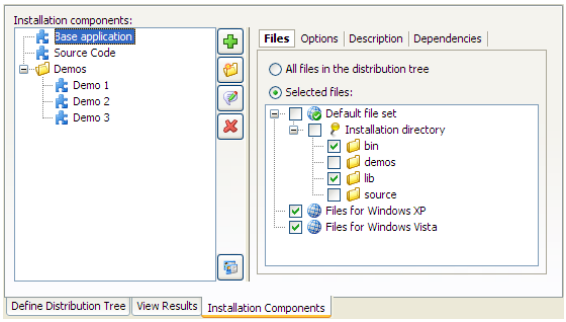
The installation of file sets can be toggled programmatically at run time. The code snippet to disable the installation of a file set at run time is `context.getFileSetById("123").setSelected(false);` if the ID of the file set is "123". You could insert this snippet into a "Run script" action that is placed before the "Install files" action on the screens & actions tab [p. 99] . File set IDs are displayed when the "Show IDs" toggle button in the lower right corner of the distribution tree is selected.

A common use case is to exclude platform-specific files from certain media files. You can define file sets for different platforms and exclude all unneeded file sets in the media wizard [p. 188] . This is an example of how to use file sets at design time in the install4j IDE.

Within one file set, all relative paths must be unique. However, the same relative path can be present in different file sets. Suppose you have different DLL files for Windows XP and Windows Vista. You can create two file sets so that the installer contains both alternative versions. Once you find out whether you run on Windows XP or on Windows Vista, you can disable the file set that should not be installed with the code snippet shown above. By default, all included file sets are installed. If the same relative path occurs twice, it is undefined which version is used. In this case you have to make sure to disable the file sets that are not appropriate.

Installation components

If you define installation components. the installer can ask the user which components should be installed. In the configuration of an installation component you mark the files that are required for this component. A single file or directory can be required by multiple installation components.



Installation components are defined in a folder hierarchy. This means you can have groups of installation components that are enabled or disabled with a single click. Most options in the configuration of an installation component are used by the "Installation components" screen [p. 105] . They decide how the installation component is presented to the user, whether it should be initially selected or mandatory, and if it has dependencies on other installation components that should be automatically selected.

Another important feature of installation components is that they can be marked as "downloadable". If you configure the download option in the media wizard [p. 184] , separate data files will be created for the downloadable components.

install4j also offers a two-step selection for installation components: In the first step, the user is asked for the desired "installation type". An installation type is a certain selection of installation components. Typical installation type sets are [Full, Minimum, Customize] or [Server, Client, All]. The display and the configuration of installation types is handled by the "Installation type" screen. For each configured installation type, you can decide whether the user should be able to further customize the associated installation component selection in the "Installation components" screen or not.

A.1.3 Screens and Actions

Introduction

With screens and actions you configure two separate aspects of the installer: the user interface that is displayed by your installer and uninstaller and the actual installation and uninstallation. Every screen can have a list of actions attached that are executed when the user advances to the next screen. install4j offers a wide variety of pre-defined screens and actions that you can arrange according to your needs. Some of these screens and actions are quite generic and can be used as programming elements, such as the "Configurable form" [p. 14] screen and the "Run script" action.

Installer applications

Building an install4j project creates media files which are either installers or archives. An installer is defined as a sequence of screens an actions and is executed when the user executes the media file. Installers usually install an uninstaller which removes the installation. The uninstaller, too, is a freely configurable sequence of screens and actions. Archives do not have an installer or uninstaller and the user extracts the contained data with other tools.

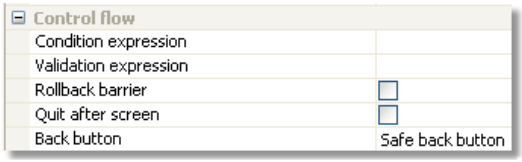
In addition to the installer and uninstaller, you can define custom installer applications [p. 102] that are added to the distribution tree. These custom installer applications can use the same screens and actions that the installer can use. Unlike installer and uninstaller, they are also added to archives. They can be used to write separate maintenance applications for your installations that are either invoked directly by the user or programatically by your application.

An important use-case for custom installer applications is to create a first-run installer for archives. While there is no need to install files to the installation directory in the case of an archive, there will usually be screens and actions that set up the environment of your application. In order to avoid the duplication of screens and actions, install4j offers the possibility to create links to screens and actions. In this way, a custom installer application can include a partial set of the screens and actions in the installer. Such a first-run installer should be added to the `.install4j` runtime directory in order to no expose it as part of the application. This is done by specifying its executable directory property as the empty string. You can invoke the first-run installer programatically with the `com.install4j.api.launcher.ApplicationLauncher` utility class. Please see the Javadoc for more information. When any of the generated launchers of an installed archive are run for the first time, the system property `install4j.firstRun` will be set. You can query that property with `Boolean.getBoolean("install4j.firstRun")` at the beginning of your main method to decide whether to launch the first-run installer or not.

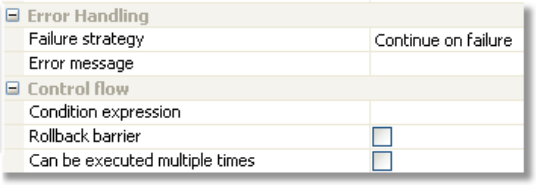
Another common use case for custom installer applications is to create auto-updaters. Auto-updaters are described in detail in a separate help topic [p. 28] .

Control flow

At runtime, install4j instantiates all screens and actions and organizes the screen flow and action execution. There are a number of aspects regarding this control flow that you can customize in the install4j IDE. Both screens [p. 105] and actions [p. 117] have an optional "Condition expression" property that can be used to conditionally show screens and execute actions. Screens have a "Validation expression" property that is invoked when the user clicks on the "Next" button allowing you to check whether the user input is valid and whether to advance to the next screen. These are the most commonly used hooks in the control flow for "programming" the installer. All "expression" properties in install4j can be simple Java expressions or scripts of Java code as described on the help page for the Java code dialog [p. 172] .



Common properties of screens



Common properties of actions

If you use a series of screens to query information from the user, the users expect to be able to go back to previous screens in order to review or change their input. This is fine as long as no actions are attached to the screen. When actions have been executed, the questions is what should happen if the user goes back to a screen with actions and clicks on "Next" again. By default, install4j executes actions only once, but that may not be what you want, if they operate on the user input in a screen. Since install4j has no way of knowing what should happen in this case, it applies a "Safe back button" policy by default: if the previous screen had actions attached, the back button is not visible. You can change this policy for each screen, either making the back button always visible or always hidden. The "Can be executed multiple times" property of each action is relevant in the case where you you make the back button always visible for the next screen.

Another hook into the control flow is the ability to declare every screen as a "Finish" screen, i.e. the "Next" button will be replaced with a "Finish" button and the installer will quit after that button is pressed. Consider to use a "banner" screen in that case since it alerts the user that a special screen has been reached.

Rollback behavior

At any time in the installation sequence the user can hit the "Cancel" button. The only exception in the standard screens is a customizable progress screen where the "Cancel" button has been disabled. install4j is able to completely roll back any modification performed by its standard actions. However, the expectation of a user might not be that the installation is rolled back. Consider a series of post-installation screens that the user doesn't feel like filling out. Depending on the installer, the user might feel that installation will work even if the installer is cancelled at that point. A complete rollback would then irritate the user. That's why install4j has the concept of a "rollback barrier". Any action or screen can be a rollback barrier which means that any actions before and including that action or screen will not be rolled back if the user cancels later on.

Be default, only the "Installation screen" is a rollback barrier. This means that if the user cancels while the installation is running, everything is rolled back. If the user cancels on any of the following screens, nothing that was performed on or before the installation screen is rolled back. With the "Rollback barrier" property of actions and screens you can make this behavior more fine-grained and customize it according to your own needs.

Error handling

Every action has two possible outcomes: failure or success. If an action succeeds the next action is invoked. When the last action of a screen is reached, the next screen is displayed. What should happen if an action doesn't succeed? This depends on how important the action is to your installation. If your application will not be able to run without the successful execution of this action, the installer should fail and initiate a rollback. However, many actions are of peripheral importance, such as the creation of a desktop link. Declaring that the installer has failed because a desktop link could not be created and rolling back the entire installation would be counterproductive. That's why the failure of an action is ignored by install4j by default. If a possible failure of an action is critical, you can configure its "Failure strategy" to either ask the user on whether to continue or to quit immediately.

Standard actions in install4j fail silently, i.e. the "Create a desktop link" action will not display an error message if the link could not be created. For all available failure strategies, you can configure an

error message that is displayed in the case of failure. The "Install files" action has its own, more granular failure handling mechanism that is automatically invoked after the installation of each file.

Standard and customizable screens

install4j offers a series of standard screens that are fully localized and serve a specific purpose. These standard screens have a preferred order, when you insert such a screen it will insert itself automatically in the correct position. This order is not mandated, you can re-order the screens in any way you like, however they may not yield the desired result anymore. If for example you place the "Services" screen after the screen with the "Install service" actions (typically the "Installation" screen), the "Services" screen will not be able to modify the service installations anymore and the default values are used.

The customizable screens don't have a fully defined purpose, their messages are configurable and empty by default. For example the "Display progress" screen is similar to the "Installation" screen, however the title and the subtitle are configurable. Customizable screens also do not have any restriction with respect to how many times they can occur. While the "Installation" screen (and other screens) can occur only once for an installer, the "Display progress" screen could be used multiple times.

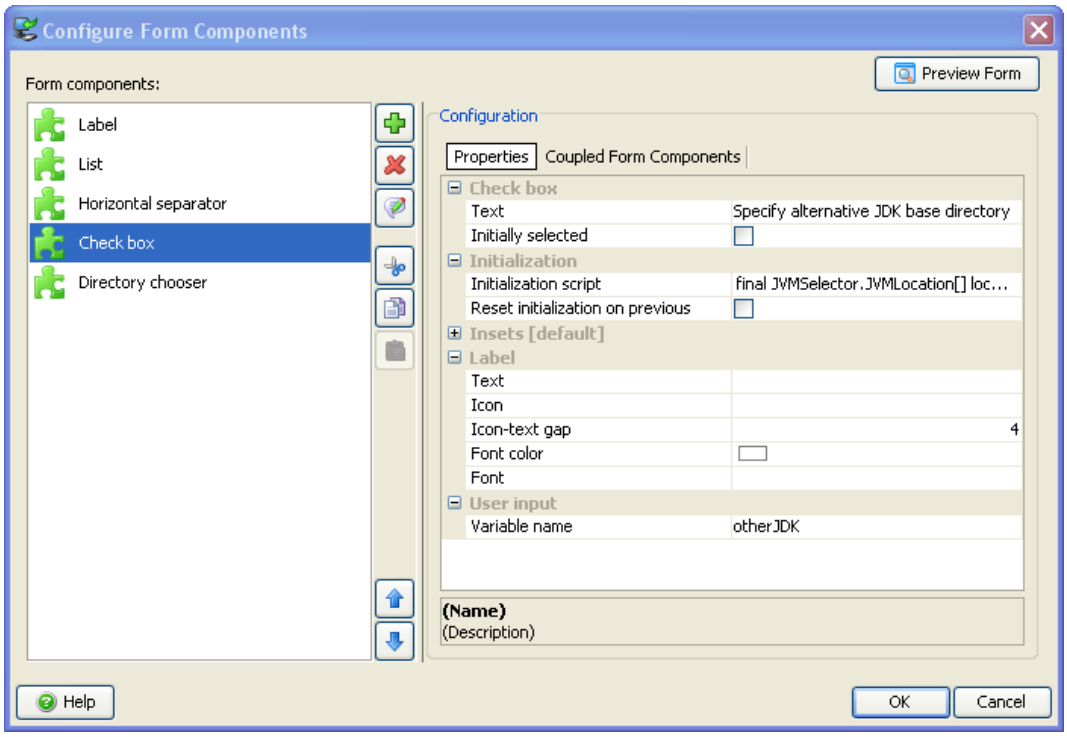
The "Welcome" and "Finish" screens have a special layout that is called "banner screen" in install4j. There are customizable banner screens to help you reproduce this layout if you require it in a different context. The most flexible of all customizable screens are the "configurable form" screens. They allow you to freely define the contents of a screen and are described in a separate help topic [p. 14] .

A.1.4 Form Screens

Introduction

Some screens in install4j contain a configurable form. In these screens, you can configure a list of form components [p. 143] along the vertical axis of the form. install4j provides you with properties to control the initialization of form components and the way the user selection is bound to installer variables [p. 17] . With this facility you can easily generate good-looking installer screens that display arbitrary data to the user and request arbitrary information to be entered.

Standard screens that have a configurable form include the "Additional confirmations" and the "Finish" screen. In addition, install4j offers a customizable form screen (similar to the "Additional confirmations" screen) and a customizable banner form screen (similar to the "Finish" screen). For screens that have a configurable form, a "Form Components" tab is shown in the "Configuration" section of the screen configuration [p. 105] . The actual configuration of the form components is performed in a separate dialog:



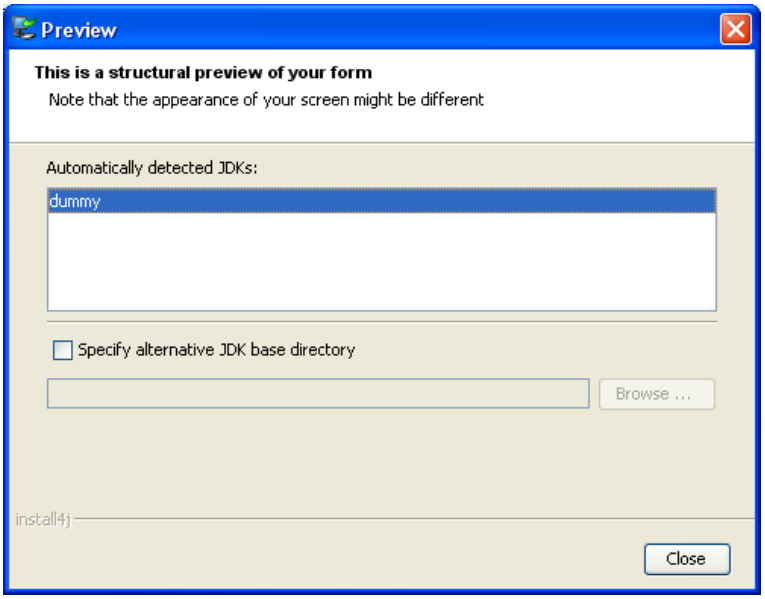
By default, a form is top-aligned and fills the entire available horizontal space. You can change this default behavior in the properties of the containing screen. For example, for a set of radio buttons that should be centered horizontally and vertically, the "Fill horizontally" and "Fill vertically" properties of the screen must be set to "false".

Form components

install4j offers a large number of form components that represent most common components available in Java and some other special components that are useful in the context of an installer. All components that expect user input have an optional leading label. The components themselves are left-aligned on the entire form. If you leave the label text empty, the form component will occupy the entire horizontal space of the form.

Every form component has configurable insets. For vertical gaps that are meant to separate groups of form components, consider using a "Vertical spacer" form component since it makes the grouping clearer and allows to to reorder form components more easily.

You can preview your form at any time with the **[Preview Form]** button. The preview dialog performs all variable replacements of compiler variables and custom localization keys, but not of installer variables. No initialization scripts are run. The preview is intended to give you quick feedback about visual aspects of your form. It does not show the actual screen where the form mights be smaller and other elements might be present. For example, the "Finish" screen is a banner screen where form occupies a relatively limited space in the bottom right corner and is intended to show a few check boxes at most.



Every form component always has its preferred vertical height. For some form components such as the "List" form component, this preferred vertical size is configurable. If the vertical extent of the form exceeds the available vertical space, a scrollbar is shown.

User input

If a form component can accept user input, you need some way to access the user selection afterwards. install4j saves user input for such form components to the installer variable [p. 17] whose name is specified in the "Variable name" property. That variable can then be used later on, for example in condition expressions for screens and actions. If you have a check box that saves its user input to a variable called "userSelection", the condition expression

```
context.getBooleanVariable("userSelection")
```

will skip the screen or action for which that condition expression is used. The user selection in form components is written to the variables before the validation expression for the screen is called. If you have a text field that saves its input to the variable "fileName", the validation expression

```
Util.showOptionDialog("Do you really want to delete " +
context.getVariable("fileName"),
new String[] {"Yes", "No"}, JOptionPane.QUESTION_MESSAGE) == 0
```

used on the same screen will block the advance to the next screen if the user answers with "No".

The values of installer variables accomodate the general type `java.lang.Object`. Every form component saves its user input in its "natural" data type, for example:

- For check boxes, the type `java.lang.Boolean` is used. For this special case the context offers the convenience method `getBooleanVariable`.

- For text fields, the type `java.lang.String` is used.
- For drop down lists the type `java.lang.Integer` is used (the selected index).
- For date spinners, the type `java.lang.Date` is used.

The description of the value type for each form component that accepts user input is shown in the registry dialog [p. 170] when you select the form component.

Initialization

For each form component, install4j offers several properties that allow you to customize its initial state. However, there may be other advanced properties or a more complex logic is required for modifying the form component. For this purpose, the "Initialization script" property is provided. Form components can expose a well-known component in the initialization script that allows you to perform these modifications. This so-called "configuration object" is usually contained in the form component itself. For example a "Check box" form component exposes a configurationObject parameter of type `javax.swing.JCheckBox` and a "Text field" form component exposes a `javax.swing.JTextField`.

As with actions and screens [p. 11] in general, the possibility that the user moves back and forth in the screen sequence presents a dilemma to install4j. Any form components that accepts user input has a configurable initial value and any form component can have an initialization script. This initialization is performed when the user enters the screen for the first time. Should this initialization be performed again when the user moves back and then enters the screen once again? Since install4j does not know, it initializes every form component only once by default. This policy can be changed with the "Reset initialization on previous" property for each form component.

A.1.5 Variables

Introduction

With variables you can customize many aspects of install4j. They can be used in all text fields and text properties in the install4j IDE as well as from the install4j API [p. 44] . The general variable syntax is

```
${prefix:variableName}
```

where prefix denotes the functionality scope of the variable and is one of

- **compiler**

  Compiler variables are replaced by the install4j compiler when the project is built.
- **installer**

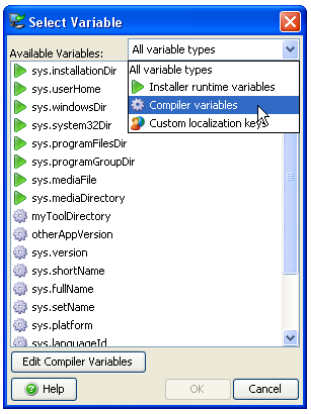
  Installer variables are evaluated when the installer or uninstaller is running.
- **launcher**

  Launcher variables are evaluated when a generated application launcher is started.
- **i18n**

  Custom localization keys are evaluated at runtime and depend on the chosen installer language.
- **(no prefix)**

  Variables with no prefix resolve to environment variables when used in the launcher configuration.

Text fields in the install4j IDE where you can use variables have a ▸ variable selector [p. 60] next to them. The variable selection dialog shows all known variables that can be used in the current context.



The above dialog, for example, is shown when clicking on the ▸ button in a text property of an installer element [p. 99] or form component [p. 143] . There, you can use compiler variables, installer variables and custom localization keys, but not launcher variables.

For both compiler and installer variables install4j offers a fixed set of "system variables". These variables are prefixed with "sys.". These variables are not writable and it is discouraged to use this prefix for your own variables.

Compiler variables

Compiler variables are written as

```
${compiler:variableName}
```

The value of a compiler variable is a string that is known and replaced at compile time. The installer runtime or the generated launchers do not see this variable, but just the value that was substituted at runtime.

You can use compiler variables for various purposes. The most common usage of a compiler variable is the possibility to define a string in one place and use it in many other places. You can then change the string in one place instead of having to look up all its usages. An example of this is the pre-defined "sys.version" variable that contains the value of the text field where you enter the application version [p. 51] . Another usage for compiler variables is to override certain project settings on a per-media file basis. For example, if you want to include one directory in the distribution tree for Windows but another one for Mac OS X, you can use a compiler variable for that directory and override it [p. 188] in the media file wizard. Finally, compiler variables can be overridden from the command line compiler [p. 197] and the ant task [p. 201] .

When you use a compiler variable in your project that is not a system variable, it must be defined in on the Compiler Variables tab [p. 58] of the General Settings step [p. 50] . If an unknown variable is encountered, the build will fail. You can use other variables in the value of a variable. Recursive definitions are detected and lead to a failure of the build. It is not possible to define compiler variables with the name of a system variable.

install4j provides a number of system compiler variables. They include:

- **sys.version**

  The application version as entered on the Application Info tab [p. 51] .
- **sys.shortName**

  The short name of the application as entered on the Application Info tab [p. 51] .
- **sys.fullName**

  The full name of the application as entered on the Application Info tab [p. 51] .
- **sys.setName**

  The name of the media file definition. If the default name of the media set is not suitable, you can rename the media set [p. 177] .
- **sys.platform**

  A short identifier for the platform (windows, linux, unix, macos). The value of this variable depends on your choice in the platform step [p. 181] of the media file wizard [p. 180] .
- **sys.languageId**

  The 2-letter ISO 639 code (see http://www.w3.org/WAI/ER/IG/ert/iso639.htm) for the principal language of the installer. This variable can be overidden on the command line or the ant task [p. 201] which is useful if you build different installers for different languages.
- **sys.withJre**

  A variable that contains "_with_jre" if a JRE is statically bundled with a media file and the empty string if not. This is useful if media files with and without JRE are built.
- **sys.date**

  The current date in the format `YYYYMMDD` (e.g. "20060910")
- **sys.javaMinVersion**

  The minimum Java version as entered on the Java Version tab [p. 52] .
- **sys.javaMaxVersion**

  The maximum Java version as entered on the Java Version tab [p. 52] .

You can access environment variables on the build machine with the syntax

```
${compiler:env.environmentVariableName}
```

where "environmentVariableName" is the name of an environment variable. This only works if no compiler variable with the same name is defined on the Compiler Variables tab. This is resolved at build time, not at run time.

In order to debug problems with compiler variables, you can switch on the `extra verbose output` flag in the Build step [p. 193] . All variable replacements will be printed to the build console.

Installer variables

Installer variables are written as

```
${installer:variableName}
```

The value of an installer variable is an arbitrary object that is not known at compile time. Installer variables are evaluated when requested in the installer or uninstaller. Installer variables are not pre-defined in the install4j IDE like compiler variables. Unlike with compiler variables, it is not an error if an installer variable is accessed that has not been defined. Undefined installer variables have a `null` value.

Installer variables are used to wire together actions, screens and form components at runtime. The user input in screens is saved to variables, which can be used in the properties of certain actions. Furthermore, variables are routinely used in condition and validation expressions. Some examples are given in the help topic on form screens [p. 14] . In expression/script properties, you retrieve variables by invoking

```
context.getVariable(String variableName)
```

Variable value can be set with the installer API by invoking

```
context.setVariable(String variableName, Object variableValue)
```

A common scenario is the need to calculate a variable value at runtime with some custom code and use the result as the initial value of a component in a screen. To achieve this you can add a "Set a variable" action to the startup screen and set its "Variable name" property to some variable name. In contexts where a variable name is expected by install4j, you must not use the `${installer:variableName}` syntax but specify `variableName` only. The return value of the "Script" property is written to the variable. If, for example, the variable represents the initial directory that is displayed for a customizable "Directory selection" screen, you then set the "Initial Directory" property of that screen to `${installer:variableName}`. In this way you have wired an action with a screen.

Another important use of installer variables is for the locations of custom installation roots [p. 64] . In most cases a custom installation root contains an installer variable that is resolved at runtime. Often, one of the system installer variables that represent a "magic" folder can be used, such as the Windows *`system32`* directory.

Installer variables can be passed to the installer or uninstaller from the command line prefixed with -V (for example -VmyVar=test). Alternatively, you can specify a property file containing installer variables with -varfile (for example -varfile myfile.prop). The variables will be String objects.

install4j provides a number of system installer variables. They include:

- **sys.installationDir**

  The installation directory for the current installation as a string. The value of this variable can change in the installer as the user selects an installation directory in the "Installation directory" screen or

the installation directory is set via `context.setInstallationDirectory(File installationDirectory)`.

- **sys.userDir**

  The user home directory as a string, typically something like *`C:\Documents and Settings\$USER`* on Windows or *`/home/$USER`* on Unix platforms.

- **sys.windowsDir**

  The Windows installation directory as a string, typically *`C:\WINDOWS`*.

- **sys.system32Dir**

  The system32 directory of your Windows installation as a string, typically *`C:\WINDOWS\system32`*.

- **sys.programFilesDir**

  The directory in your Windows installation where programs are installed as a string, typically something like *`c:\Program Files`*.

- **sys.programGroupDir**

  The directory of the program group that will be or was created by the "Create standard program group" action as a string. If this action is not present, the value will be `null`. The value of this variable can change in the installer as the user selects a program group on the "Create program group" screen.

- **sys.mediaFile**

  The path of your media file as a string. Not available for uninstallers.

- **sys.mediaDirectory**

  The path of the directory where your media file is located as a string. Not available for uninstallers.

Launcher variables

Launcher variables are written as

```
${launcher:variableName}
```

The value of a launcher variable is a string that is not known at compile time. Launcher variables are evaluated when a generated application launcher is started. Launcher variables can only be used in the VM parameters text field [p. 82] of the launcher wizard [p. 78] . No user-defined launcher variables exist, the available system launcher variables include:

- **sys.launcherDirectory**

  The directory in which your launcher is located.

- **sys.jvmHome**

  The home directory of the JVM that your launcher is running with. This is useful to put JAR files from the JRE into your boot classpath.

- **sys.pathlistSeparator**

  The platform-dependent separator for lists of directories. On Window, this is a semicolon (";"), on Unix a colon (":").

Custom localization keys

Custom localization keys are written as

```
${i18n:keyName}
```

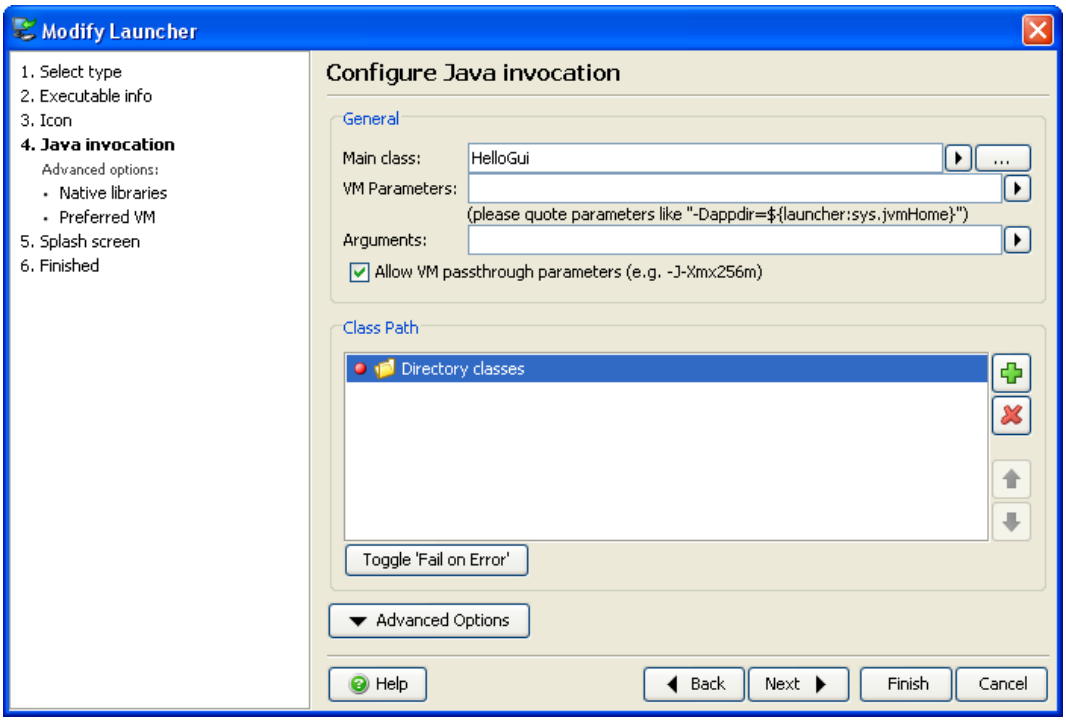
The value of a custom localization key depends on the language that is selected for the installer. You can use this facility to localize messages in your installers if they support multiple languages [p. 54] . You can supply key value pairs for internationalization in the custom localization file. The variable selection dialog shows all keys in the custom localization file for the principal language of your project.

All standard messages displayed by install4j can be referenced with this syntax as well. You can locate the key name in one of the *message_*.utf8* files in the *$INSTALL4J_HOME/resource/messages* directory and use it anywhere in your project. The standard messages can be overwritten by your custom localization files.

A.1.6 VM parameters

Fixed VM parameters

Fixed VM parameters can be configured in the launcher wizard [p. 82] where you can use compiler variables [p. 17] to handle platform-specific changes or launcher variables [p. 17] to use runtime-dependent variable in your VM parameters.



*.vmoptions files

A common requirement is to adjust the VM parameters of your application launchers depending on the runtime environment like the target platform or some user selection in the installer.

In addition to the fixed VM parameters, a parameter file in the same directory as the executable is read and its contents are added to the existing VM parameters. The name of this parameter file is the same as the executable file with the extension `.vmoptions`. For example, if your executable is named `hello.exe`, the name of the VM parameter file is `hello.vmoptions`. In this file, each line is interpreted as a single VM parameter. The last line must be followed by a line feed. install4j adapts your .vmoptions files during the compilation phase so that the line endings are suitable for all platforms. For example, the contents of the VM parameter file could be:

```
-Xmx128m
-Xms32m
```

The `*.vmoptions` files allow the installer as well as expert users to modify the VM parameters for your application launchers.

In addition to the VM parameters you can also modify the classpath in the `*.vmoptions` files with the following options:

- **-classpath [classpath]**

  Replace the classpath of the generated launcher.
- **-classpath/a [classpath]**

Append to the classpath of the generated launcher.

- **-classpath/p [classpath]**

  Prepend to the classpath of the generated launcher.

For GUI launchers on Mac OS X, the VM options are stored in a file called *`Info.plist`* inside the application bundle. The "Add VM options" action described below handles these platform-specific differences.

Environment variables

You can use environment variables in the VM parameters and the *`.vmoptions`* file with the syntax `${variableName}` where you replace `variableName` with the desired environment variable.

This environment variable syntax also works in the arguments text field and the classpath configuration.

"Add VM options" action

In order to handle VM parameter additions in the installer in a cross-platform fashion, install4j includes an "Add VM options" action [p. 117] that adds VM parameters to the *`*.vmoptions`* file on Microsoft Windows and Unix and modifies the *`Info.plist`* file on Mac OS X.

The Add VM options action creates a *`*.vmoptions`* file if necessary or adds your options to the *`*.vmoptions`* file if it already exists. However, a number of VM parameters can only occur once so the action replaces the following parameters if they already exist:

- -Xmx
- -Xms
- -Xss
- -Xloggc
- -Xbootclasspath
- -verbose
- -ea / -enableassertions
- -da / -disableassertions
- -agentlib
- -agentpath
- -javaagent
- -splash

as well as the install4j-specific classpath modification options (see above).

To set an -Xmx value that depends on the total memory of the target system, you can use a "Set a variable action" to calculate the numeric part of the -Xmx value using the utility method `SystemInfo.getPhysicalMemory()` and use that variable in the "VM options" property of the "Add VM options" action. For example, in order to use 50% of the total memory for the maximum heap size, you do the following after the "Install files" action:

1 Add a "Set a variable" action with variable name "xmx" and expression

```
"-Xmx" + Math.round(SystemInfo.getPhysicalMemory() * 0.5 / 1024 / 1024)
+ "m"
```

2 Add a "Add VM options" action with VM options

"${installer:xmx}".

A.1.7 JRE Bundles

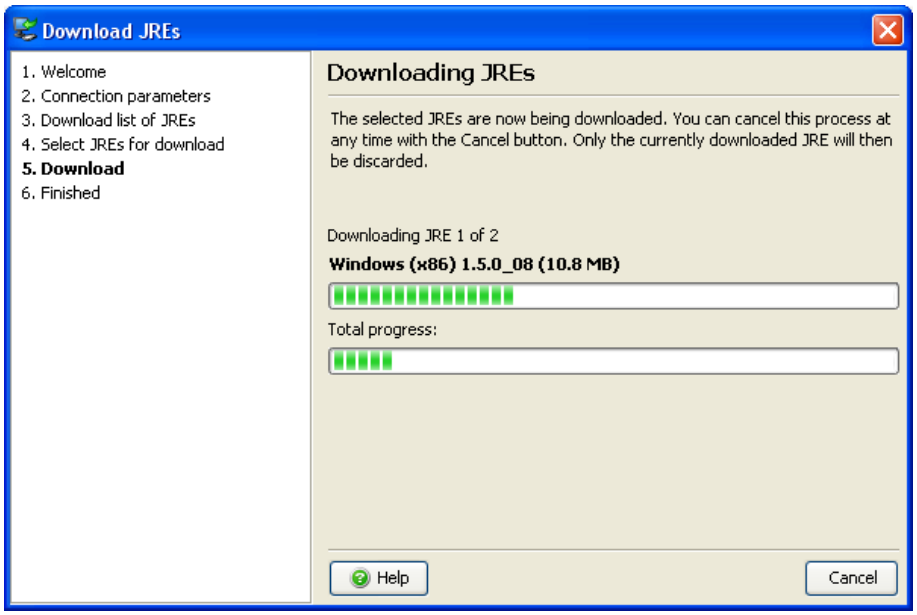
Introduction

When deploying a Java application to users that are not all in the same environment, it is advisable to bundle a JRE with your application or at least to offer a download with a bundled JRE. install4j offers you a number of strategies for JRE bundling. A statically bundled JRE is always distributed along with your application. Dynamical bundling means that if the installer cannot find a suitable JRE on the target computer, it will download a JRE bundle from your web server.

Any JRE bundle that is installed by install4j will not interfere with default JRE installations. In particular, it will not be integrated into browsers and no registry entries will be written. However, it is possible to install JRE bundles as "shared", meaning that other installers generated by install4j will be aware of these bundles. A shared JRE bundle will not be uninstalled when the application that has installed the bundle is uninstalled itself. If you dynamically bundle a JRE for multiple installers and install it as a shared JRE, only the first time when a user installs one of your installers, a JRE will be downloaded. Subsequent installations of other installers will find that shared JRE.

Obtaining JRE bundles

ej-technologies offers a JRE bundle download service [p. 194] that is invoked from the install4j IDE.

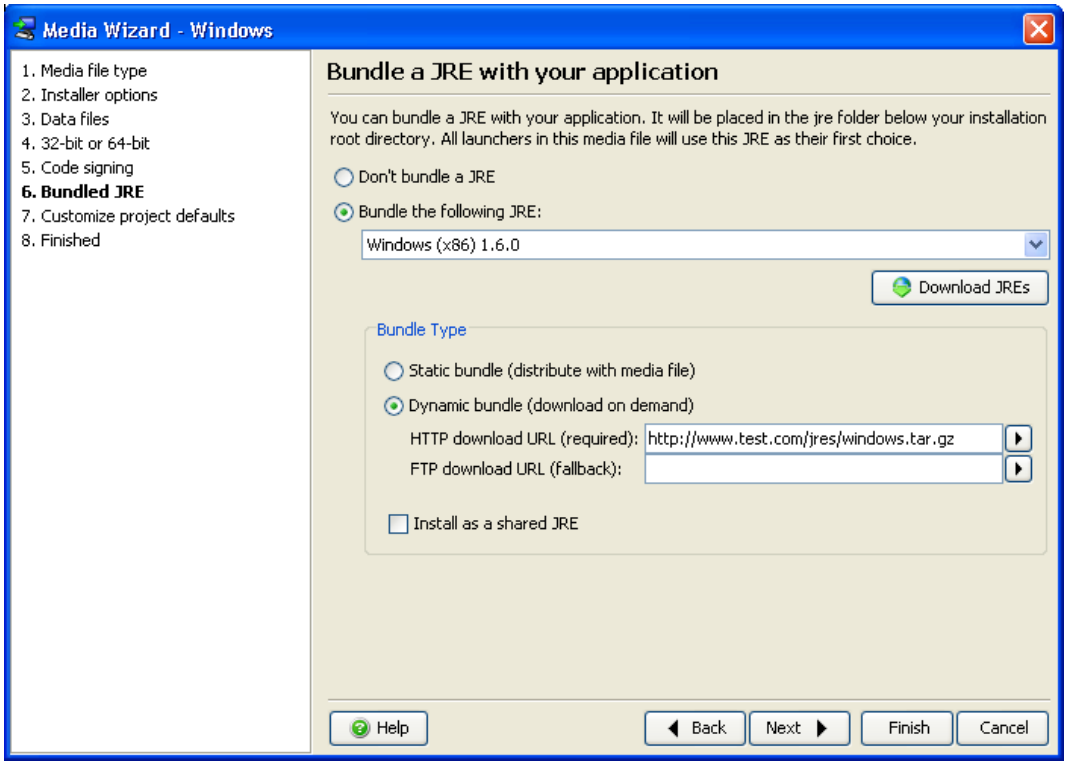


All JREs are saved with a *tar.gz* extension to the directory *$INSTALL4J_HOME/jres* or, if that directory is not writable, to *$HOME/.install4j4/jres*. If you require JRE bundles on a computer without an internet connection, you can transfer these files to the equivalent location of that computer.

Please note that on Mac OS X the JRE installations are part of the operating system. For technical and licensing reasons it is not possible for applications to install their own JREs. Mac users who keep their system updated always have the latest JRE provided by Apple.

Using JRE bundles

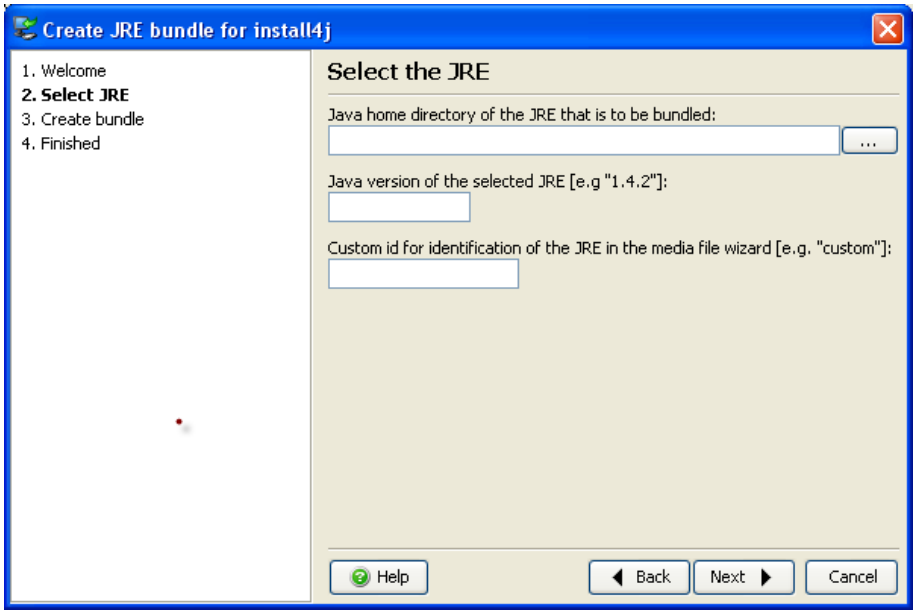
Downloaded JREs can be selected for bundling in the Bundled JRE [p. 186] step of the media wizards [p. 180] . All generated launchers [p. 76] use the bundled JRE as their first choice.



If you would like to put your JRE bundles in a different directory, such as a directory in a version-controlled location, you can copy the `.tar.gz` file (see above) to that directory and choose the "Manual entry" JRE bundle to enter the path to the bundle file.

Creating JRE bundles

If the JRE bundles created by ej-technologies do not satisfy your needs, you can create a JRE bundle from any installed JRE on your file system. install4j offers the "Create a JRE bundle" wizard [p. 195] to make this task as simple as possible.



Packaging your own JRE can be useful if you want to add standard extensions such as the Java Communications API to your JRE. The JRE bundle wizard only works for the platform you are running on. That means, to create a JRE bundle for Windows, you have to run install4j on Windows, to create a bundle for Linux, you have to run install4j on Linux.

In special cases you might want to create a JRE bundle programmatically, i.e. without using the install4j IDE. This can be done with the standard GNU tools `tar` and `gzip`. A JRE bundle for install4j is simply a file with the naming scheme:

```
[operating system]-[architecture]-[JRE version].tar.gz
```

For windows bundles, the operating system name must be "windows", for other platforms any name can be used. The *`.tar.gz`* file directly contains the JRE, i.e. the *`bin`* and *`lib`* folders. The steps to create a bundle are outlined below:

```
cd jre
tar cvf minix-x86-1.5.0.tar *
gzip minix-x86-1.5.0.tar
cp minix-x86-1.5.0.tar.gz /usr/install4j/jres
```

First you change into the top-level directory of the JRE, then you tar all files and directories and gzip the tar archive. The last step copies the bundle into the directory *`$INSTALL4J_HOME/jres`*. You have to restart install4j for the JRE to be listed in the "Bundled JRE" step of the media file wizard.

If you choose to bundle your JRE this way on Microsoft Windows, you have to install the tar and gzip tool available at

- tar: http://gnuwin32.sourceforge.net/packages/tar.htm
- gzip: http://gnuwin32.sourceforge.net/packages/gzip.htm

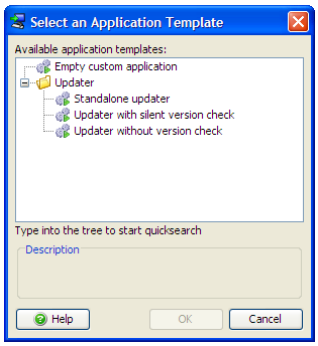
A.1.8 Auto-Update Functionality

Introduction

install4j can help you to include auto-updating functionality into your application. Auto-updating means two things: First, there must be a way to check if there is a newer version available for download. This check can be initiated by the user in various ways or the check can be triggered automatically by your application. Second, there must be a way to download and execute an appropriate installer for the new version.

install4j creates a special file *updates.xml* in the media output directory when you build the project. This file describes the media files of the current version. If you want to use install4j's auto-update functionality, you have to upload this file to a web server. This file is downloaded by deployed installations as described below and delivers information about the current version.

Downloading and installing the new version is done with a custom installer application [p. 102] . install4j offers several templates for updaters that correspond to the update strategies outlined below in this help topic.



updates.xml

The *updates.xml* file is created in the media output directory each time you build the project. You can use this file as is, however, some situations require that you modify the file before uploading it to the web server. The file looks like the sample below:

```
<?xml version="1.0" encoding="UTF-8"?>
<updateDescriptor baseUrl="">
<entry targetMediaFileId="8" updatableVersionMin="" updatableVersionMax=""
fileName="hello_windows_4_0.exe"
newVersion="4.0" newMediaFileId="8" fileSize="2014720" bundledJre="">
<comment />
</entry>
<entry targetMediaFileId="9" updatableVersionMin="" updatableVersionMax=""
fileName="hello_linux_4_0.rpm"
newVersion="4.0" newMediaFileId="9" fileSize="817758" bundledJre="">
<comment />
</entry>
<entry targetMediaFileId="10" updatableVersionMin="" updatableVersionMax=""
fileName="hello_macos_4_0.dmg"
newVersion="4.0" newMediaFileId="10" fileSize="1359872" bundledJre="">
<comment />
</entry>
</updateDescriptor>
```

The root of the *updates.xml* file is the updateDescriptor element. It contains the baseUrl attribute that can be used to specify an alternate download URL for the installers. By default, it is empty which means that the installers must be located in the same directory as the *updates.xml* file. The updateDescriptor element contains one or more entry elements which correspond to the media files that were created by the build.

When install4j determines whether an entry in the update descriptor is a match for the current installation, it looks at three attributes of the entry element: Most importantly, the targetMediaFileId attribute has to match the media file ID of the current installation. You can show media file IDs by invoking *Media->Show IDs* in the media section [p. 177] of the install4j IDE. If you discontinue a media file, you can migrate users of that media file to a different media file by duplicating the desired entry in *updates.xml* and changing the targetMediaFileId attribute to that of the discontinued media file. Another criterion is the installed version of the application. Depending on that version, you might want to offer different updates. The updatableVersionMin and the updatableVersionMax attributes can set lower and upper limits for the installed versions that should download the associated entry in the update descriptor. By default, these attributes are empty, so no version restrictions apply.

Attributes that describe the update installer include fileName which is necessary to construct the download URL, and fileSize which contains the size of the file in bytes. newVersion contains the available version while newMediaFileId is the media file ID of the update installer which is the same as targetMediaFileId unless you changed it yourself. Lastly, bundledJre contains the original file name of the JRE bundle without the *.tar.gz* extension or the empty string if no JRE is bundled in the installer. In addition to the above attributes, the nested comment element can contain a description that should be displayed to the user. All of this information can be used for custom logic to select a suitable update installer or be displayed to the user in the updater. In addition, you can add any number arbitrary attributes to the entry element yourself.

The update descriptor

The install4j runtime API contains the com.install4j.api.update.UpdateChecker utility class that can download the *updates.xml* file and translate it to an instance of com.install4j.api.update.UpdateDescriptor. From there, you can get a suitable com.install4j.api.update.UpdateDescriptorEntry with a single method call. Please see the Javadoc for more detailed information. The above API is primarily intended for use in your application. The install4j runtime API contained in *resource/i4jruntime.jar* is always on the class path for a generated launcher.

In a custom installer application, you would rather use a "Check for update" action that performs the same actions as UpdateChecker and saves the downloaded UpdateDescriptor to an installer variable. All updater templates included with install4j execute the "Check for update" action at some point.

Instances of UpdateDescriptorEntry expose all attributes of the corresponding entry element in the *updates.xml* file. They also give access to additional attributes added to the entry element so you can implement custom logic to find a suitable update. The most important method of the UpdateDescriptorEntry class is the getUrl() method that constructs the full URL from which the update installer can be downloaded. If no baseUrl has been specified on the updateDescriptor root element, the URL starts with the the parent directory from which the *updates.xml* file has been downloaded.

Strategy 1: Standalone updater

The easiest way to provide auto-update functionality to your users is to create a self-contained updater application. This is done by adding an application on the screens & actions tab [p. 99] and choosing the "Standalone updater" application template. Such an auto-updater can by invoked manually by the user, on Windows, it can also be added to the start menu. No changes in in your application code are required so far.

If you have a GUI application, you could provide integration with the updater by offering a "Check for update" menu item or similar that invokes the updater. One problem in this scenario is that if the updater downloads and executes the update installer, your application will still be running and the user will receive a corresponding warning message in the installer. The solution to this problem is to use the `com.install4j.api.launcher.ApplicationLauncher` class to launch the updater. With this utility class you can launch the update installer by passing its ID as an argument. IDs can be shown on the screens & action tab by toggling the "Show IDs" tool bar button. If you launch an installer application such as an updater that way, the "Shut down calling launcher" action will be able to close your application. To react to the shutdown, for example, to invoke your own shutdown routine, you can pass a callback to the `ApplicationLauncher.launchApplication(...)` call. After you were notified through the call back, your application will be terminated with a call to `System.exit()`.

Strategy 2: Updater with silent version check

In this scenario, you invoke the updater like in strategy 1, but rather than offering a "Check for update" menu item, you do so on a regular schedule. For example, you automatically check for updates every week or each time the user starts the application. In that case, the standalone updater template is not suitable since you only want to give the user feedback if there is actually a new version available. However, the standalone updater always starts with a "Welcome" screen, verbosely checks for updates and informs the user that no new version is available. Most likely, your users will be bothered if this is done automatically.

The "Updater with silent version check" application template is intended for this use case. It checks for an update in the startup sequence and terminates the updater if no new version is available. This means that if there is no new version available, your users will not see that a check has taken place. Only if a new version is available will the updater display its window and inform the user of the possibility to download the update installer.

For such an automatic check you will likely want to invoke the updater in a blocking fashion. If you call `ApplicationLauncher.launchApplication(...)` with the `blocking` argument set to `true`, the method will not return until the update installer has exited. If the user decides to run the installer on the "Finish" screen, your application will terminate as explained in strategy 1.

Strategy 3: Updater without version check

If you want to take the integration one step further and display the availability of a new version in your application yourself, you can use the `com.install4j.api.update.UpdateChecker` class as explained under the "updates.xml" heading. In this way, you can create your own panel that announces the new version and lets the user decide whether to download it or not. If the user decides to download, the "Updater with silent version check" template is not suitable since it informs the user about the new version once more.

The "Updater without version check" application template is intended for this use case. It immediately starts downloading the new version and then proceeds to the "Finish" screen where the user can decide to start the downloaded installer. In the other two templates the user can choose the directory where the downloaded installer should be saved. That screen is omitted in this template and the installer is downloaded to the user home directory by default. You can change this default directory be passing the argument `-DupdaterDownloadLocation=[directory]` to the `ApplicationLauncher.launchApplication(...)` call. Again, the updater will terminate your application if the user starts the installer as explained for strategy 1.

Update schedule registry

For strategy 2 and 3 above, you check for an update on a regular schedule. install4j comes with a standard implementation of an update schedule registry that frees you of the task to implement one yourself. The `com.install4j.api.update.UpdateScheduleRegistry` class is intended to be used in your application. You configure an `com.install4j.api.update.UpdateSchedule` with a call to `UpdateScheduleRegistry.setUpdateSchedule(...)` and call

`UpdateScheduleRegistry.checkAndReset()` each time your application is started. If you get a positive response, you can start a suitable updater as explained above. Please see the Javadoc for more information.

To facilitate the configuration of the update schedule in your installer, install4j offers a special "Update schedule selector" form component whose initial value is set to the current setting (if any) and automatically updates the setting for the installed application when the user clicks "Next".

A.2 Generated installers

A.2.1 Installer Modes

Introduction

Installers generated by install4j can be run in three modes:

- **GUI mode**

  The default mode for installer and uninstaller executables is to display a GUI installer or uninstaller.
- **Console mode**

  If the installer is invoked with the `-c` argument, the interaction with the user is performed in the terminal from which the installer was invoked. The same applies to the uninstaller.
- **Unattended mode**

  If the installer is invoked with the `-q` argument, there is no interaction with the user, the installation is performed automatically with the default values. The same applies to the uninstaller.

The screen flow and the action sequence is executed in the same way for all three modes. If some actions or screens should not be traversed for console or unattended installations, you can use the methods `context.isConsole()` and `context.isUnattended()` in their "Condition expression" properties.

Also see the command line options [p. 34] reference for installers.

Console mode

Installers generated by install4j can perform console installations, unless this feature has been disabled in the application configuration [p. 102] of the Installer step [p. 98]. In order to start a console installation, the installer has to be invoked with the `-c` argument.

All standard screens in install4j present their information on the console and allow the user to enter all information as in the GUI installer. Not all messages in the GUI installer are displayed to the console installer, for each screen the subtitle is displayed as the first message. All standard screens in install4j have a question as their subtitle, if you add customizable screens to the screen sequence [p. 99], you should phrase their subtitles as questions in order to create a consistent user experience for the console installer.

Also, form screens [p. 143] are fully mapped to console installers, each form component is displayed on the console, form components that expect user input will allow the users to modify or enter values.

On Microsoft Windows the information of whether an executable is a GUI executable or a console executable has to be statically compiled into the executable. Installers are GUI executables, otherwise a console would be displayed when starting the installer from the explorer. This is also the reason why the JRE supplies both the `java.exe` (console) and the `javaw.exe` (GUI) on Windows.

Since Windows XP there is a way for a GUI executable to attach to a console from which it was started. GUI executables are started in the background by default, therefore you have to use the start command like this to start it in the foreground and be able to enter information:

```
start /wait installer.exe -c
```

On older Windows versions a new console is opened.

If you develop new screens or form components, you have to override the method

```
boolean handleConsole(Console console) throws UserCanceledException
```

Displaying default data on the console and requesting user input is made easy with the `Console` class that is passed as a parameter.

Unattended mode

Installers generated by install4j can perform unattended installations, unless this feature has been disabled on the application configuration [p. 102] of the Installer step [p. 98] . In order to start an unattended installation, the installer has to be invoked with the `-q` argument. The installer will perform the installation as if the user had accepted all default settings.

There is no user interaction on the terminal. In all cases, where the installer would have asked the user whether to overwrite an existing file, the installer will not overwrite it. You can change this behavior by passing `-overwrite` as a parameter to the installer. In this case, the installer will overwrite such files. For the standard case, it is recommended to fine-tune the overwrite policy [p. 68] in the distribution tree instead, so that this situation never arises.

The installer will install the application to the default installation directory, unless you pass the `-dir` parameter to the installer. The parameter after `-dir` must be the desired installation directory. Example:

```
installer.exe -q -dir "d:\myapps\My Application"
```

For the unattended mode of an installer, response files [p. 36] are an important instrument to pre-define user input.

On Windows, the output of the installer is not printed to the command line for unattended installation. If you pass the `-console` parameter after the `-q` parameter, a console will be allocated the displays the output to the user. This is useful for debugging purposes.

If the installation was successful, the exit code of the installer will be `0`, if no suitable JRE could be found it will be `83`, for other types of failure it will be `1`.

If you develop new screens or form components, you have to override the method

```
boolean handleUnattended()
```

in order to support unattended installations.

A.2.2 Command Line Options for Generated Installers

Installers generated by install4j recognize the following command line parameters:

Name	Explanation
-manual	This option applies to Microsoft Windows only. The default JRE search sequence [p. 38] will not be performed and bundled JREs will not be used either. The installer will act as if no JRE has been found at all and display the dialog that lets you choose a JRE or download one if a JRE has been bundled dynamically. If you locate a JRE, it will be used for the installed application. On Unix, you can define the environment variable INSTALL4J_JAVA_HOME_OVERRIDE instead to override the default JRE search sequence.
-c	Executes the installer in the console mode [p. 32] .
-q	Executes the installer in the unattended mode [p. 32] .
-console	If the installer is executed in unattended installation mode [p. 32] and -console is passed as a second parameter, a console will be allocated on Windows that displays the output of the installer.
-overwrite	Only valid if `-q` is set. In the unattended installation mode [p. 32] , the installer will not overwrite files where the overwrite policy [p. 68] would require it to ask the user. If `-overwrite` is set, all such files will be overwritten.
-dir [directory]	Only valid if `-q` is set. Sets a different installation directory for the unattended installation mode [p. 32] . The next parameter must be the desired installation directory.
-Dinstall4j.nolaf=true	Do not set the native look and feel but use the default. In some very rare cases, the Windows look and feel with the classic theme (Windows 2000-like appearance) on Windows XP is broken and prevents the use of the installer or any other Java GUI application. Switching to the default XP theme solves this problem. Alternatively, passing this parameter to the installer will prevent the native look from being set.
-Dinstall4j.debug=true	By default, install4j catches all exceptions, creates a "crash log" and informs the user about the location of that log file. This might be inconvenient when debugging an installer, so this system property switches off the default mechanism and exceptions are printed to stderr. In addition, install4j prints informative messages to stdout each time an installer variable is set. To dump all installer variables to stdout, you can use `com.install4j.api.Util.dumpVariables(Context context)` independently of this system property.
-Dinstall4j.keepLog=true or -Dinstall4j.alternativeLogfile=[path]	install4j creates a log file prefixed *`i4j_log`* for all installations and uninstallation in your temp directory. This log file can be helpful for debugging purposes. If your installer contains an "Install files" action and terminates successfully the log file is copied to *`[installation dir]/.install4j/installation.log`*, otherwise it will be deleted

	after the installer or uninstaller terminates by default. With the `-Dinstall4j.keepLog=true` option, the log file won't be deleted in this case. With the `-Dinstall4j.alternativeLogfile=[path]` the log file will be copied to the file specified with `[path]`. This should be an absolute path name. Note that both options have no effect if the log file has already been copied to the installation directory.
-Dinstall4j.logToStderr=true	In addition to the log file created by the installer or uninstaller, you can duplicate all log messages to stderr with this argument.
-Dinstall4j.logEncoding=[character set name]	By default, the installer will write the log file in the default encoding of the system where the installer is running. Should you wish to choose a different encoding you can pass this VM parameter to the installer. Some common character set names are • UTF-8 • ISO-8859-1 • US-ASCII • UTF-16LE • UTF-16 Most JREs support a large number of charsets. You can execute `java.nio.charset.Charset.availableCharsets()` to check the names of supported character sets for your JRE.
-D`propertyName`=`value`	You can set further arbitrary system properties with the standard command line parameter.
-V`variableName`=`value`	You can set arbitrary installer variables with the -V parameter. The variable name should be used without prefix, so if you have a variable called ${installer:variableName} in the GUI the parameter would be -VvariableName=value. The variable will be a String object.
-varfile [fileName]	Alternatively, you can specify a property file containing the variables you want to set. The variable names should be used without prefix, too, so if you have a variable called ${installer:variableName} in the GUI the entry would be variableName=value. The variables will be String objects. This option shares the same mechanism with response files [p. 36] .

A.2.3 Response files

Introduction

With a response file you can change the default user selection in all screens. A response file is a text file with name value pairs that represent certain installer variables. All screens provided be install4j ensure that they write all user selections to appropriate installer variables and bind their user interface components to these variables. This includes form screens [p. 143] .

Installer variable values are of the general type `java.lang.Object`. In a response file, only variables with values of certain types are included: The default type is `java.lang.String`. In addition the types `java.lang.Boolean`, `java.lang.Integer`, `java.lang.String[]` and `int[]` are supported. In order to let the installer runtime know about these non-default types, the variable name in the response file is followed by a '$' sign and an encoding specifier like 'Integer' or 'Boolean'.

All installer variables live in the same name space. If you use an installer variable more than once for different user inputs, the response file only captures the last user input and may lead to erroneous behavior when the installer is run with a response file. If you would like to optimize your installers for use with a response file, you have to make sure that the relevant variable names are unique within your installer.

A response file can be used to

- Configure the installer for unattended execution mode
- Change the default settings in the GUI and console installer
- Get additional debugging information for an installation

When applying a response file to an installer, all variable definitions are translated into installer variables [p. 17] . The response file shares the same mechanism with the variable file offered by the -varfile [p. 34] installer option. You can add the contents of a response file to a variable file and vice versa.

Generating response files

There are two ways to generate a response file:

- A response file is generated automatically after an installation is finished. The generated response file is found in the *`.install4j`* directory inside the installation directory and is named *`response.varfile`*. When you request debugging information from a user, you should request this file in addition to the installer log file.
- install4j offers a "Create a response file" action [p. 117] that allows you to save the reponse file to a different file in addition to the automatically generated response file. Here, you can also specify variables that you would not like to be included in the reponse file. Together with an appropriate form component on the "Additional confimations" screen you can query the user whether to create such a response file or not.

Applying response files

When an installer is executed, it checks whether a file with the same name and the extension **.varfile** can be found in the same directory and loads that file as the response file. For example, if an installer is called *`hello_setup.exe`* on Windows, the response file next to it has to be named *`hello_setup.varfile`*.

You can also specify a response file explicitly with the -varfile [p. 34] installer option.

Response files work with all three installer modes [p. 32] , GUI, console and unattended.

Response file variables

The variables that you see in the response file exist at runtime independently of the response file. You can use these installer variables to access or change user selections on system screens. For example, the "Create program group" screen on Windows binds the user selection for the check box that asks the user whether to create the program group for all users to the variable **sys.programGroup.allUsers**. To access the current user selection from somewhere else, you can use the expression

```
context.getBooleanVariable("sys.programGroup.allUsers")
```

To change that selection, you can invoke

```
context.setVariable("sys.programGroup.allUsers", Boolean.FALSE)
```

A.2.4 How Installers Find a JRE

Installers generated by install4j are native and can start running without a JRE. However, the installer itself requires a JRE in order to perform its work and so the first action of the installer is to locate a JRE that is suitable for both the installer and your application. In this process it performs the following steps:

- **Look for a statically bundled JRE**. If a statically bundled JRE is included with the installer, it will unpack it and use it. First, this JRE is unpacked to a temporary directory, later it is copied to a location that depends on whether the bundled JRE is configured as shared or not.

  - **Not shared**

    It is copied to the *jre* directory in the installation directory of your application. No other installer generated by install4j will find this JRE. It will not be made publically available (e.g. in the Windows registry).

  - **Shared**

    It is copied to the *i4j_jres* directory in a common folder which depends on the operating system:

    - *C:\Program Files\Common Files* on Microsoft Windows with an English locale.
    - */opt* if it exists, otherwise */usr/local* on Unix.

    If the above folder is not writable, the *i4j_jres* directory will be created in the use home directory and the shared JRE will only be shared for the current user.

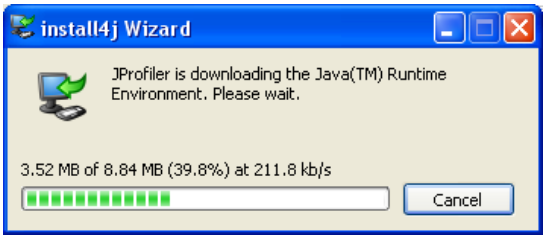
    Other installers generated by install4j will find this JRE. It will not be made publically available (e.g. in the Windows registry). For each Java version, only one such JRE can be installed. Shared JREs are never uninstalled.

  Your application will also use the JRE selected by the installer.

- **Look for a suitable JRE in the configured search sequence**. The installer uses the same search sequence and Java version contraints as your launchers which are configured for the entire project [p. 52] . The most important search sequence element in this respect is the "Search Windows registry and standard locations" entry. On Microsoft Windows the registry contains information on installed JREs, on Unix platforms there is a number of standard locations which are checked, on Mac OS X the location of installed JREs is always the same.
- If no JRE has been found, the installer notifies the user

  and offers the following options:

  - Download a dynamically bundled JRE



as configured in the Bundled JRE [p. 186] step of the media wizard [p. 180] .

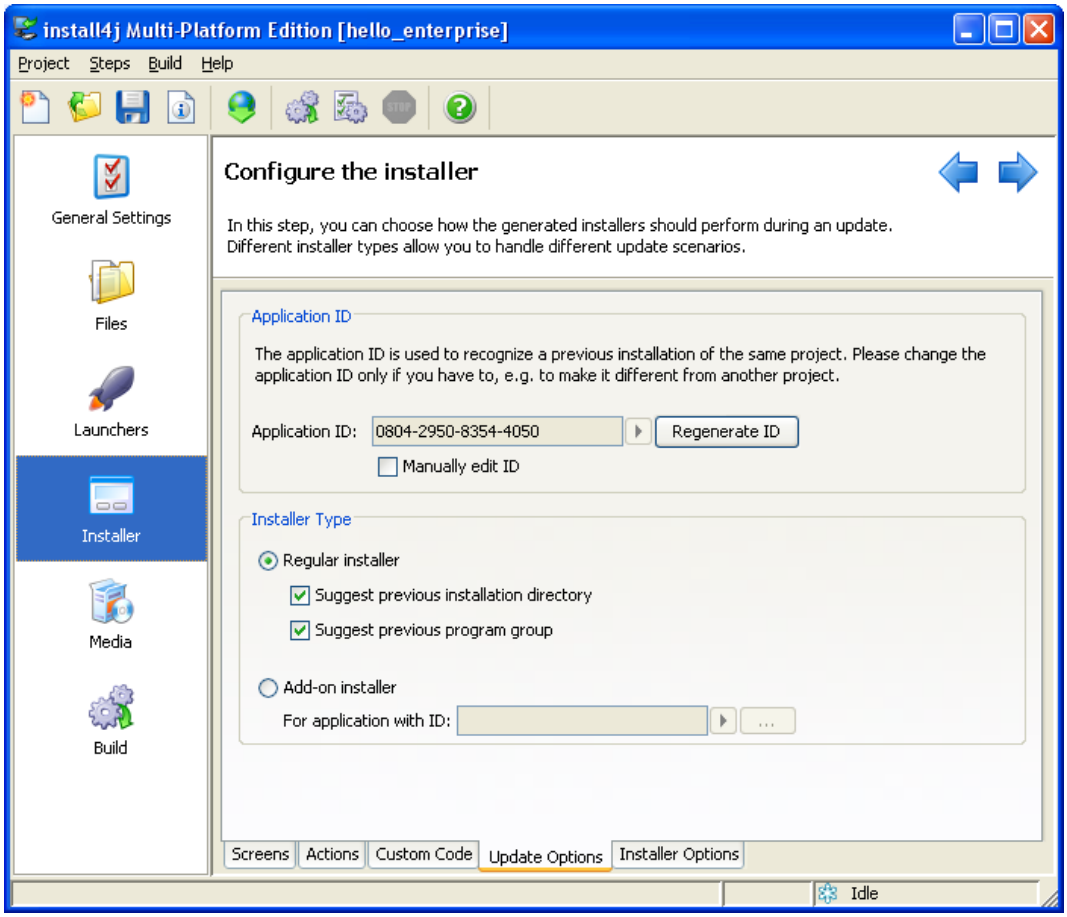
- Manually locate a JRE
- Cancel the installation

You can force the installer to skip the first two steps and show this dialog immediately with the `-manual` command line parameter [p. 34] .

A.2.5 Updates

Introduction

installers generated by install4j actively handle updates. On the Update Options [p. 169] tab in the installer section, you can configure how an installer should behave in the event of an update. An update occurs when the user installs an application into a directory where an installation with the same application id already exists.



Typically, minor upgrades of an application should be installed into the same directory as earlier installations. The default behavior of install4j is to suggest the previous installation directory and program group, so that the user is guided into installing the application into the same directory. If this behavior is not desired, you can switch off these suggestions or change the application id [p. 169] .

Updates into the same installation directory

The following points are of interest with respect to updates into the same installation directory:

- Generated installers will refuse to install on top of installations with a different application ID by default. You can change this behavior on the "Installation location" screen.

  Note: installers generated with install4j <= 3.0.x do no have an application ID, it is always possible to install on top of such an installation.
- Generated installers will detect if any of the previously installed launchers are still running and will ask the user to shutdown these applications. This happens when the "Install files" action is executed.

- Deployed services will be stopped and uninstalled before the installation. This happens when the "Install files" action is executed. You can optionally stop your services earlier with the "Stop a service" action if your update process requires it.
- During an update, the installation databases will be merged, so that files, menu entries, file associations, etc. of old installations can still be uninstalled when the uninstaller is executed.
- After an update, only the (optional) uninstall actions of the newer installation will be executed when the uninstaller is executed. However, the auto-uninstall actions from previous installations will be executed, too (for example the uninstallation of a service that is automatically registered by an "Install service" action during installation).

If you would like to uninstall the previous installation before installing any new files, you can add the "Uninstall previous installation" action before the "Install files" action. In this context, the uninstallation policies [p. 70] that exclude updates are of interest. With these uninstallation policies you can preserve certain files for updates, but uninstall them when the user manually invokes the uninstaller. The uninstaller invoked by the "Uninstall previous installation" action is running in unattended mode. You can use `context.isUninstallForUpgrade()` to exclude certain actions for an update uninstaller.

Add-on installers

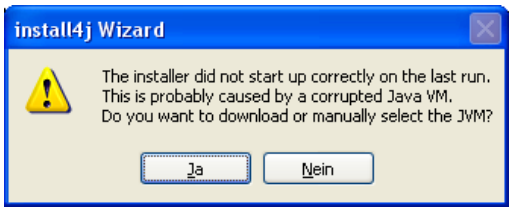
For distributing enhancements and patches, install4j offers the add-on installer type that can be configured on the Update Options [p. 169] tab in the installer section.

An add-on installer will only install on top of an installation of a specified application id. It does not have a separate uninstaller.

A.2.6 Error Handling

Damaged JREs

If the JRE search sequence [p. 38] of the installer selects a damaged JRE, the installer might fail to start up correctly. The next time the installer is executed, the installer will ask the user whether the automatic search should be performed again or if a JRE should be manually located or downloaded.



If the user chooses manual location or download, the same dialog will be displayed as for in the failure to find a JRE [p. 38] .

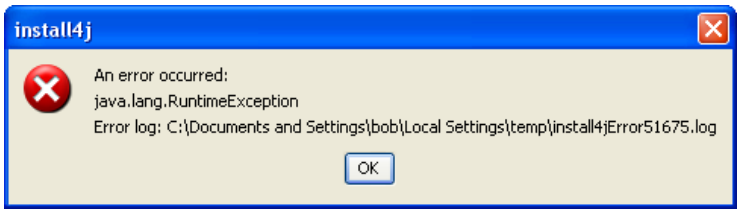
The download option is only available if a JRE has been dynamically bundled in the Bundled JRE [p. 186] step of the media file wizard [p. 180] . A JRE that has been located or downloaded in this way will also be used by your installed application.

Error logs

On Windows, when an installer is executed it always generates a log file in the temp directory that contains information about the JRE search sequence and can be used for debugging purposes. The name of the log file starts with `i4j_nlog_`. If you have a problem with JRE detection or the installer startup, please send this log file along with your support request.

It is also possible to generate this log for the JRE detection of the generated Windows launchers. In order to switch on logging, please define the environment variable `INSTALL4J_LOG=yes` and look for the newest text file whose name starts with `i4j_nlog_` in the temp directory.

If an exception is thrown in the installer, it prepares an error log and informs the user about its location



You can force the installer to print exceptions to stderr for debugging purposes with the `-Dinstall4j.debug=true` command line option [p. 34] .

Installation log

Additionally, all installers and uninstaller generate an installation log that can be used for debugging purposes. After a successful installation it is located in *`[installation dir]/.install4j/installation.log`*. For uninstallation or if the end of the installation cannot be reached, you can find it in your temp directory if you pass `-Dinstall4j.keepLog=true` to the

installer or uninstaller. The file is prefixed *i4j_log*. If you would like the installer to log to stderr as well, you can pass `-Dinstall4j.logToStderr=true` to the installer. Both arguments can also be useful for debug installers and uninstallers, where they have to be passed as VM parameters.

Error handling of Actions

You can define the error handling for every installation or uninstallation action separately. Please find more information in the Screens and Actions help topic [p. 11] .

Return values

The process of an installer returns `0` if the installation was completed successfully, `1` if the installation fails and `83` if the installer could not find a suitable JVM to run. These exit codes are especially useful to check the result of an unattended installer run [p. 32] .

A.3 Extending install4j

A.3.1 Developing with the install4j API

Introduction

There are two different circumstances where you might want to use the install4j API: Within expression/script properties [p. 172] in the configuration GUI and for the development of custom elements in install4j. The development of custom elements in install4j is rarely necessary for typical installers, most simple custom actions can be performed with a "Run script" action and most custom forms can be realized with a "Customizable form" screen.

If you would not like to miss your IDE while writing more complex custom code, you can put a single call to custom code into expression/script properties. The location of your custom code classes must configured [p. 168] , so install4j will package it with the installer and put in into the class path. In this way you can completely avoid the use of the interfaces required to extend install4j.

When you want to use install4j classes within your IDE, you can add *`$INSTALL4J_HOME/resource/i4jruntime.jar`* to your classpath (in your IDE). Do not distribute this jar file with your application, install4j will handle this for you.

Expression/script properties

Using expression/script properties in install4j is required for wiring together screens and actions [p. 11] as well as for the conditional execution of screens and actions. The most important element in this respect is the **context** which is an instance of

- **com.api.install4j.context.InstallerContext**

  in an installer
- **com.api.install4j.context.UninstallerContext**

  in an uninstaller

The context allows you to query the environment and the configuration of the installer as well as to perform some common tasks.

Please see the documentation of the com.install4j.api.context package for the complete documentation of all methods in the context. Some common applications include:

- **Setting the installation directory**

  By using `context.setInstallationDirectory(File installationDirectory)` in the installer context, you can change the default installation directory for the installer. Typically, this call is placed into a "Run script" action on the "Startup" screen.
- **Getting and setting installer variables**

  The `getVariable(String variableName)` and `setVariable(String variableName, Object value)` methods allow you to query and modify installer variables. Note that besides the "Run script" action, there is also a "Set a variable action" where you don't have to call `setVariable` yourself.
- **Conditionally executing screens or actions**

  Often, condition expressions for screens and actions check the values of variables. In addition, the context provides a number of boolean getters that you can use for conditionally executing screens and actions depending on the installer mode and environment. These methods include `isConsole()`, `isUnattended()` and others.
- **Navigating between screens**

Depending on the user selection on a screen, you might want to skip a number of screens. The `goForward(...)`, `goBack(...)` and `goBackInHistory(...)` methods provide the easiest way to achieve this.

Many other context methods are only useful if you develop custom elements for install4j.

Also have a look at the `com.install4j.api.Util` class which offers a number of utility methods that are useful in expression/script properties.

Developing custom elements for install4j

For a general overview on how to start developing with the install4j API, how to set up your IDE and how to debug your custom elements, please see the API overview in the javadoc.

install4j provides three extension points:

- Actions [p. 117]

  Please see the documentation of the com.install4j.api.actions package for the complete documentation on how to develop actions.
- Screens [p. 105]

  Please see the documentation of the com.install4j.api.screens package for the complete documentation on how to develop screens.
- Form components [p. 143]

  Please see the documentation of the com.install4j.api.formcomponents package for the complete documentation on how to develop form components.

All actions, screens and form components in install4j use this API themselves. To make your custom elements selectable in the install4j IDE, you first have to configure the custom code locations [p. 168] . When you add an action, screen or form component, the first popup gives you the choice on whether to add a standard element or search for suitable elements in your custom code [p. 170] .

If you use your custom code in multiple projects, consider packaging an install4j extension [p. 47] , which displays your custom elements alongside the standard elements that are provided by install4j and allows you to ship them easily to third parties.

Serialization

install4j serializes all instances of screens, actions and form components with the default serialization mechanism for JavaBeans that is present in Java since version 1.4. The install4j runtime includes its own implementation of `java.beans.XMLDecoder`, so the generated installers can work with Java 1.3 JREs as well if your Java version constraints [p. 52] are configured to allow this.

To learn more about JavaBeans serialization, please visit

- http://java.sun.com/j2se/1.4.2/docs/api/java/beans/XMLEncoder.html for API documentation on the long-term persistence mechanism for JavaBeans.
- http://java.sun.com/products/jfc/tsc/articles/persistence3/ for information on the format of the XML serialization.
- http://java.sun.com/products/jfc/tsc/articles/persistence4/ for information on how to write your own persistence delegates. In your bean infos for screens, actions and form components you can specify a list of additional persistence delegates for non-default types.

Compiler variables are replaced in the serialized representation of a bean. In this way, compiler variable replacement is automatically available for all properties of type `java.lang.String`. The values of installer variables and localization keys are determined at runtime, so you have to call the

utility methods in `com.install4j.api.beans.AbstractBean` before you use the values in the installer or uninstaller. For more information on variables, please see the separate help topic [p. 17] .

A.3.2 Extensions

Introduction

All standard actions [p. 117] , screens [p. 105] and form components [p. 143] in install4j use the installer API [p. 44] themselves. With this API you can create new elements that are displayed in the standard registries [p. 170] by packaging a JAR file with a few special manifest entries and putting that JAR file into the *extensions* directory of your install4j installation.

Configurability

An extension to install4j will likely need to be configurable by the user. install4j uses the JavaBean specification to control the user presentation of properties in the install4j IDE. Screens, actions, and form components correspond to beans in this context.

Optionally, you can add BeanInfo classes. In essence, a BeanInfo class next to the bean itself describes which properties are editable and optionally gives details on how they should be presented. Please see the documentation of the com.install4j.api.beaninfo package for the complete documentation on how to develop BeanInfo classes. Also, the *$INSTALL4J_HOME/samples/customCode/src* directory contains a sample action with the associated BeanInfo class.

JAR manifest

In order to tell install4j which classes are screens, actions or form components, you have to use the following manifest keys:

- **Install-Action**

  for actions implementing `com.install4j.api.actions.InstallAction`
- **Uninstall-Action**

  for actions implementing `com.install4j.api.actions.UninstallAction`
- **Installer-Screen**

  for screens implementing `com.install4j.api.screens.InstallerScreen`
- **Uninstaller-Screen**

  for screens implementing `com.install4j.api.screens.UninstallerScreen`
- **Form-Component**

  for screens implementing `com.install4j.api.formcomponents.FormComponent`

Please note that usually you do not implement these interfaces yourself, but rather extend one of the abstract base classes.

A typical manifest with one action and one screen looks like this:

```
Depends-On: driver.jar common.jar

Name: com/mycorp/actions/MyAction.class
Install-Action: true

Name: com/mycorp/screens/MyScreen.class
Installer-Screen: true
Uninstaller-Screen: true
```

Note: If you only have named sections and no global section in your manifest file, the first line must be an empty line since it separates the global keys from the named sections.

The `Depends-On` manifest key can specify a number of relative JAR files separated by spaces that must be included when the extension is deployed. That key can also occur separately for each named section.

As you see in the example for the screen, each class can have multiple keys if the appropriate interfaces are implemented.

Extension deployment

On startup, install4j will scan the manifests of all JAR files that it finds in the *`$INSTALL4J_HOME/extensions`* directory. Any screens, actions or form components that are found in the manifests are added to the standard registries [p. 170] . If a bean cannot be instantiated, the exception is printed to stderr which is captured in *`$INSTALL4J_HOME/bin/error.log`* and no further error is displayed.

If any of those screens, actions or form components are selected by the user, the required JAR files are deployed with the generated installers. This means that installing extensions does not create an overhead for installers that do not use them.

B Reference

B.1 Steps for Configuring an install4j Project

To learn more about install4j projects, please see the corresponding help topic [p. 7] or other help topics about concepts in install4j [p. 7] .

install4j's main window is organized into 6 steps that are required to build a set of media files. The side bar on the left as well as the forward and back buttons in the top right corner let you navigate between these steps:

- Step 1: General Settings [p. 50]

  (`CTRL-1`) In the **General Settings step**, you provide important information about your application and the build preferences, such as the name of your application, the JRE search sequence and the directory where the media files should be placed.

- Step 2: Files [p. 63]

  (`CTRL-2`) In the **Files step**, you define your **distribution tree**, that means you collect files from different places to be distributed in the generated media files. You can optionally define installation components.

- Step 3: Launchers [p. 76]

  (`CTRL-3`) In the **Launchers step**, you define the properties of the **native launchers** that will enable your users to start your application.

- Step 4: Installer [p. 98]

  (`CTRL-4`) In the **Installer step**, you configure the installer screens and actions.

- Step 5: Media [p. 177]

  (`CTRL-5`) In the **Media step**, you define the **media files** that will be created to distribute your application to your end users.

- Step 6: Build [p. 193]

  (`CTRL-6`) In the **Build step**, you start the actual generation of the media files.

B.2 Step 1: General Settings

B.2.1 Step 1: Enter General Project Settings

In the **General Settings step**, you provide important information about your application and specify project-wide build settings.

There are several tabs in this section:

- Application Info [p. 51]

  On this tab you enter information about your application, such as name and version.
- Java Version [p. 52]

  On this tab define the version requirements for the JRE that your application launchers should use as well as the detailed JRE search sequence.
- Languages [p. 54]

  On this tab define the principal language as well as other languages supported by the installer.
- Media File Options [p. 56]

  On this tab you enter general options regarding media file generation such as the output directory for media files and compression settings.
- Compiler Variables [p. 58]

  On this tab you can define compiler variables for your project.
- Project Options [p. 59]

  On this tab you you can adjust options regarding your install4j project.

B.2.2 General Settings - Application Info

On this tab of the General Settings step [p. 50] you enter general information about your application.

Only options with bold labels have to be filled in. The available options are:

- **Full name**

  (required) the long name used in situations where there is plenty of space for displaying a name.
- **Short name**

  (required) the alternative short name for situations where there is limited space for displaying a name or where a name should be as short as possible. The short name is used to create suggestions for installation directories in the Media step [p. 177] . It may not contain spaces.
- **Version**

  (required) the version number of your application. This value can be overriden from the command line [p. 197] or the ant task [p. 201] .
- **Publisher**

  the name of your company or your own name (e.g. used for the support information dialog in the Windows control panel)
- **Publisher URL**

  the web address of your company (e.g. used for the support information dialog in the Windows control panel)

A build will not be possible until all required fields have been completed. If a required field is missing when starting a build [p. 193] , this tab will be displayed with a warning message.

B.2.3 General Settings - Java Version

On this tab of the General Settings step [p. 50] you enter the version requirements and the search sequence for the JRE or JDK that apply to your installers [p. 98] and application launchers [p. 76] .

In the `Java version` section, you can constrain the version of the Java VM.

- The **minimum Java version** must be specified. For example, enter a value of `1.3`.
- The **maximum Java version** can optionally be specified. For example, enter a value of `1.4`.

The maximum Java version can be entered with less numeric detail than the minimum Java version to prevent the use of a higher major or minor release. For example, a minimum version of 1.4.1 and a maximum version of 1.4 ensures that the highest available 1.4.x >= 1.4.1 JRE is used, but not a 1.5 JRE. Similarly, a minimum version of 1.4.1_03 and a maximum version of 1.4.1 ensures that the highest available 1.4.1 >= 1.4.1_03 JRE is used, but not a 1.4.2 JRE.

By default, JREs with a beta version number or JREs from an early access release cycle will not be used by the launcher. If you would like to enable the use of these JREs, please check the option `allow JREs with a beta version number`.

The JRE search sequence determines how install4j searches for a JRE on the target system. New configurations get a pre-defined default search sequence. install4j has a special mechanism which allows you to bundle JREs with your media files. If you choose a particular JRE for bundling [p. 186] in one of the media file wizards [p. 180] , this JRE will always be used first and you do not need to adjust the search sequence yourself.

If you have problems with JRE detection at runtime, please see the help topic on error handling [p. 42] for a description on how to get diagnostic information.

The following types of search sequence entries [p. 60] are available:

- Search registry
- Directory
- Environment variable

The control buttons on the right allow you to modify the contents of the search sequence list:

- Add search sequence entry (key `INS`)

  Invokes the search sequence entry dialog [p. 60] . Upon closing the search sequence entry dialog with the **[OK]** button, a new search sequence entry will be appended to the bottom of the search sequence list.
- Remove search sequence entry (key `DEL`)

  Removes the currently selected search sequence entry without further confirmation.
- Move search sequence entry up (key `ALT-UP`)

  Moves the selected search sequence entry up one position in the class path list.
- Move search sequence entry down (key `ALT-DOWN`)

  Moves the selected search sequence entry down one position in the class path list.

The **design time JDK** determines which JDK or JRE is used for the following purposes:

- **Code compilation**

  install4j uses a bundled eclipse compiler, so it does not need this functionality from a JDK. However it needs a runtime library against which scripts entered in the installer configuration [p. 98] are compiled. The version of that JDK should correspond to the minimum Java version for the project configured above.

  By default, the `rt.jar` runtime library of the JRE that is is used to run the install4j IDE is used for code compilation. If your minimum Java version is lower than the the Java version used to run install4j, runtime errors can occur if you accidentally use newer classes and method.

- **Context-sensitive Javadoc help**

  If you configure a separate design-time JDK or JRE, you can enter a Javadoc directory to get context-sensitive Javadoc help for the runtime library in the Java code editor [p. 172] . By default, context-sensitive Javadoc help is only available for the install4j API [p. ?] .

- **Enhanced code completion functionality**

  If you configure a separate design-time JDK (and not a JRE), the Java code editor [p. 172] will show completion proposals for methods in the runtime library with parameter names. By default, parameter names for methods in the runtime library are not available.

The drop-down list next to the `JDK` option shows the name of all configured JDKs together with their Java versions. In order to configure a new design time JDK, select the `JDK` option, and click on **[Configure JDKs]**. This shows the Configure JDKs dialog [p. 61] .

The list of available design time JDK is saved globally for your entire install4j installation and not for the current project. The only information saved in your project is the name of the JDK configuration. In this way, you can bind a suitable JDK on another installation and on other platforms. If the JDK name saved in the project cannot be found in your install4j installation, the name will be displayed in red color with a "[not configured]" message attached. In that case, when clicking on **[Configure JDKs]**, you will be asked if you would like to configure this JDK.

A build will not be possible until all required fields have been completed. If a required field is missing when starting a build [p. 193] , this tab will be displayed with a warning message.

B.2.4 General Settings - Languages

On this tab of the General Settings step [p. 50] you define the principal language as well as additional languages supported by your installers [p. 98] .

The following options are available:

- **Principal language**

  The principal language is the language that your installer defaults to if no other supported languages match the locale at runtime.
- **Custom localization file**

  A custom localization file is text file with key-message pairs in the format of

  - **a Java properties file**

    a Java properties file has ISO 8859-1 encoding, all other characters must be represented as Unicode escape sequences, like `\u0823`.
  - **a properties file with UTF-8 encoding**

    A properties file with UTF-8 encoding has the advantage that you do not have to use escape sequences. However it might not be supported by i18n tools.

  A custom localization file can be used to

  - **override system messages**

    If any of the default messages in the installer is not appropriate for your use-case, you can change it by looking up the corresponding keys in the appropriate *`$INSTALL4J_HOME/resource/messages/messages_*.utf8`* file and define the same key in your custom localization file to override that message.
  - **localize your installer**

    Anywhere in the install4j IDE where you can enter text that is used at runtime, you can use custom localization keys [p. 17] , i.e. variables of the form `${i18n:myKey}`. Those keys are read from your custom localization file and offered by the variable selection dialog [p. 60] .

  If required, you can use parameters for your messages by using the usual `{n}` syntax in the value and listing the parameters in function-like manner after the key name in the variable instance. For example, if your key name is `myKey` and your message value is

  ```
  Create {0} entries of type {1}
  ```

  you can use a variable

  ```
  ${i18n:myKey("5", "foo")}
  ```

  in order to fill the parameters, so that the actual message becomes

  ```
  Create 5 entries of type foo
  ```

  However, in the context of localizing an installer this is rarely necessary.
- **Additional languages**

  With install4j, you can build multi-language installers that offer the user a choice between a number of languages. If you add languages to the additional languages table, the installer becomes a multi-language installer, otherwise is is a fixed-language installer. When you add a new language,

the language selection dialog [p. 60] is displayed. A new entry is then added to the table and you can configure the custom localization file by double-clicking on the appropriate cell.

- **Skip language selection dialog if auto-detected locale matches a supported language**

  This check box ensures that the language selection is only displayed if the installer cannot find a match between a supported language (either principal or additional language) and the auto-detected locale at runtime. By default this option is not selected and the language selection dialog is always displayed.

The principal language and the associated custom localization file can be overridden for each media file [p. 188] . In this way you can **build multiple fixed-language installers each with a different language**.

B.2.5 General Settings - Media File Options

On this tab of the General Settings step [p. 50] you enter general options regarding media file generation.

Only options with bold labels have to be filled in. The available options are:

- **Media file output directory**

  (required) the directory where the generated media files should be placed. If the project has already been saved, a relative directory will be interpreted as relative to the project file.

- **Media file name pattern**

  (required) the default rule for naming your media files. This text field should contain system compiler variables [p. 17] in order to be unique for each media file. If two media file names are equal, the build will fail. If the desired name for the media file cannot be obtained through the use of variables, you can override the media file [p. 188] name in the media wizard.

- **Convert dots to underscores**

  By default, dots ('.') will be converted to underscores ('_') when the media file name is evaluated. If you would like to keep all dots in your media file name, please de-select this option.

- **Compression level**

  The desired level of compression for your media files, chosen from a range of 1-9. "1" means least compressed and "9" means most compressed. Please note that extracting the media files will take longer for higher compression levels.

- **Use LZMA compression**

  LZMA compression achieves much better results, but is considerably slower, especially for compilation. LZMA compression is only used for installers and not for archives.

- **Use Pack200 JAR compression**

  Pack200 compression is a compression algorithm that's designed for JAR files and achieves exceptional results, especially for large JAR files. Since the Pack200 deflater is only included since JRE 1.5, this compression is only used if the minimum Java version requirement [p. 52] for your project is 1.5.

  If you have signed JAR files or JAR files that create a digest, please apply the *`$JDK_HOME/bin/pack200`* executable in your build process like

  ```
  pack200 --repack my.jar
  ```

  before signing the JAR files. Pack200 rearranges JAR files but the reordering is idempotent, so this pack/unpack sequence creates a stable JAR file.

  Pack200 compression can be quite slow, Pack200 decompression is relatively fast. Pack200 compression is only used for installers and not for archives.

  To avoid problems with external JAR files, you can check the the "Exclude signes JARs or JARs creating digests" option. If you would like to exclude selected JAR files only, you can place an empty *`*.nopack`* file next to it. For example, if the jar file is named *`app.jar`*, then a file *`app.jar.nopack`* in the same directory will disable Pack200 compression for that file.

- **Modification times**

  You can choose between two ways to set the modification times of installed files:

  - **Keep original file modification times**

    The original modification times are kept for the installed files. This is the default mode.

  - **Use build timestamp**

All installed files have the build time as the same modification time.

A build will not be possible until all required fields have been completed. If a required field is missing when starting a build [p. 193] , this tab will be displayed with a warning message.

B.2.6 General Settings - Compiler Variables

On this tab of the General Settings step [p. 50] you enter compiler variables for your project.

Defining compiler variables is optional and not required for a working project. For an explanation of compiler variables, please see the help topic on variables [p. 17] .

The control buttons on the right allow you to modify the contents of the compiler variables list:

- Add variable (key `INS`)

  Adds a new variable. The variable name must be unique and must not be the name of a system variable.

- Remove variable (key `DEL`)

  Removes the currently selected variable.

- Move variable up (key `ALT-UP`)

  Moves the selected variable up one position in the variables list.

- Move variable down (key `ALT-DOWN`)

  Moves the selected variable down one position in the variables list.

In order to edit any column, please double-click on it.

B.2.7 General Settings - Project Options

On this tab of the General Settings step [p. 50] you can adjust options regarding your install4j project.

The following options are available:

- **Make all paths relative when saving the project file**

  If this option is checked, install4j will try to convert all absolute paths to relative paths when saving the project file. Relative paths are always interpreted **relative to the project file**.

  If you save your project under a different path, all relative paths will be adjusted accordingly.

  Note: for cross platform usage of a single project file enabling this option is highly recommended, since file system roots are inherently incompatible across platforms.

- **Create backup files when saving**

  If this option is checked, install4j will create a backup copy of existing project files by appending "~" to the file name.

- **Auto save every 5 minutes**

  If this option is checked, install4j will save your project file every 5 minutes. Note that if the project has never been saved before, no auto save operation will be attempted.

B.2.8 Dialogs

B.2.8.1 Search Sequence Entry Dialog

The search sequence entry dialog is shown when clicking on the add button in the Java Version tab [p. 52] of the General settings step [p. 50] . Upon closing this dialog with the **[OK]** button, a new search sequence entry will be appended to the bottom of the search sequence list on that tab.

To define a search sequence entry, you select the entry type and fill out the `Detail` section of the dialog which is dependent on the selected entry type. The following entry types are available:

- **Search registry**

  Search the Windows registry and well-known standard locations for installed JREs and JDKs by Sun Microsystems.

- **Directory**

  Look in the specified directory. This is useful if you distribute your own JRE (not one provided by install4j) along with your application. In that case, be sure to supply a relative path. Note that relative directories will be interpreted as relative to the installation root directory.

- **Environment variable**

  Look for a JRE of JDK in a location that is defined by an environment variable like `JAVA_HOME` or `MYAPP_JAVA_HOME`.

B.2.8.2 Language Selection Dialog

The variable selection dialog is displayed when you click on the **[Add Language]** button on the Languages tab [p. 54]

Select one of the supported languages with a double-click or the **[OK]** button. Next to each language the ISO code is displayed that is required if you override the principal language from the command line with the LANGUAGE_ID variable [p. 198] .

B.2.8.3 Variables Selection Dialog

The variable selection dialog is displayed when you click on the variable selector [p. 60] which can be found next to most text fields in install4j. It allows you to insert a variables into the text field.

Please see the help topic on variables [p. 17] more information on variables.

The variable will be inserted at the current cursor position, if you close the variable selection dialog with the **[OK]** button or of you double-click on a variable in the list.

Variables are shown with an icon which indicates their type:

- Compiler variables

  Compiler variables [p. 58] are available for all text fields.

- Installer variables

  Installer variables are only available for text properties of screens [p. 105] , actions [p. 117] and form components [p. 143] .

- Launcher variables

Runtime variables are system variables that are evaluated at runtime of the launcher or installer. They are only shown in the variable selector next to the VM parameter text field in the Java invocation step [p. 82] of the launcher wizard [p. 78] .

- Custom localization keys

  Custom localization keys are available if you've registered a custom localization file on the Languages tab [p. 54] .

The combo box at the top right corner allows you to filter one of the above variable types. By default, all variable types are shown.

To add a compiler variable on the fly, you can click on **[Edit variables]** to invoke the variables edit dialog [p. 61] .

B.2.8.4 Compiler Variables Edit Dialog

The variables edit dialog is displayed when you click **[Edit variables]** in the variable selection dialog [p. 60] . It contains the same controls as the Compiler Variables tab [p. 58] in the General Settings step [p. 50] . Upon closing the dialog with the **[OK]** button, the contents of the list in the variable selection dialog will be updated.

B.2.8.5 Input Dialog

The input dialog allows you to enter a simple string value. Depending on the context, you can use compiler variables [p. 17] in the value. To select a variable, you can click on the variable selector [p. 60] .

B.2.8.6 Configure JDKs Dialog

The JDK configuration dialog is displayed when you click the **[Configure JDKs]** button on the Java version [p. 52] tab or in the Java editor settings dialog [p. 174] .

In this dialog, you can add one or more JDKs that will be available for the purposes outlined on the Java version [p. 52] help page. The JDK configurations are **not saved in the project**, they are saved globally for your install4j installation.

When you add a new JDK, you are asked for the home directory of the JDK that you want to enter. Instead of a JDK, you can also select a JRE, in which case no parameter names will be available in the code completion proposals of JDK methods. After you select the home directory, install4j will check whether the directory contains a JDK or JRE and runs `java -version` to determine the version of the selected JDK or JRE.

The table that shows the configured JDKs has several columns:

- **Name**

  This is a symbolic name that describes the JDK, like "JDK 1.6" When you select a design time JDK on the Java version [p. 52] tab or in the Java editor settings dialog [p. 174] , only this symbolic name will be saved in the project file. When users on other computers and other platforms configure a JDK with the same symbolic name in their install4j installation, it will be used automatically for code compilation and code completion.

  When you add a JDK, the name "JDK [major version].[minor version]" will be suggested by default. If the selection is a JRE, "JRE [major version].[minor version]" will be suggested instead. The name of the JDK configuration must be unique.

- **Java Home Directory**

  This is the Java home directory that you selected when you added the configuration. You can change the Java home directory by editing this column. The Java version check will be performed

again and the version displayed in the "Java Version" column will be updated. The symbolic name of the configuration will not be changed.

- **Javadoc Directory**

  In this column, you can enter the location of the Javadoc that should be associated with this JDK configuration. The Javadoc directory can remain empty in which case no context-sensitive Javadoc help will be available for classes from the runtime library.

- **Java Version**

  This uneditable column shows the version of the selected JDK configuration.

When you delete the JDK configuration that is currently used by the project, the project will still reference the same configured symbolic name for the JDK. It will then be shown in red color with a [not configured] message attached.

Changing the order of JDK configuration changes the order in the drop-down list on the Java version [p. 52] tab or in the Java editor settings dialog [p. 174] .

B.3 Step 2: Files

B.3.1 Step 2: Configure Distributed Files

In the **Files step**, you define your **distribution tree**. This means that you collect files from different places to be distributed in the generated media files. In addition, you can optionally define installation components.

There are three tabs in this section:

- Define distribution tree [p. 64]

  On the definition tab, you can **add and edit the structural elements** that make up the distribution tree. You can create your own directory structure and "mount" directories from your hard disk or add single files in arbitrary directories.

- View results [p. 72]

  On the results tab, you see the **actual file tree** as it will be collected and distributed by the generated media files [p. 177] . Go to this tab to check whether your actions on the definition tab have actually produced the desired results.

- Installation components [p. 73]

  On the components tab, you can optionally define parts of the distribution tree as installation components to **allow users to customize the installation** of your application.

B.3.2 Defining the distribution tree

B.3.2.1 Files - Defining the Distribution Tree

The distribution tree shows your file selections and the distribution directory structure created by you. The distribution tree is **drag-and drop enabled**.

To check whether your definition actually produces the desired results, please go to the View Results tab [p. 72] of the Files steps [p. 63] .

The top-level nodes in the distribution tree are called **file sets**. There is one "Default file set" that cannot be deleted or renamed. The relative paths of all files that are added to a file set must be unique. Please see the help topic on file sets and installation components [p. 9] for more information on how to use file sets.

Within a single file set, it causes an error if the installation paths for two files collide. For example, if you have added the contents of two different directories into the same folder in the distribution tree and both directories contain a file `file.txt`, building the project will fail with a corresponding error message. In this case, you have to exclude the file in one of the directory entries. This is only valid for files, sub-directory hierarchies on the other hand are merged and can overlap between multiple directory entries and explicitly added folders.

You can create new file sets with the **[New File Set]** action in the add menu on the right side. Each file set has its own "Installation directory" root. If you define custom roots that should be present in multiple file sets, you have to duplicate them.

When using the install4j API [p. ?] , you reference file sets with IDs. You can show IDs in the distribution tree by activating the **[Show IDs]** button on the lower right side of the distribution tree. The automatically generated numerical IDs are then shown in brackets. The selection will be remembered across restarts of install4j.

The child nodes of a file set are called **installation roots**. Their location is resolved when the installer runs. There are two types of roots:

- The **default root of the distribution tree** is labeled as "Installation directory" and has a special icon. This is the directory where your application will be installed on the target system. The directory is dependent on user actions at the time of installation. In regular installers a user can select an arbitrary directory where the application should be installed. For RPM media files, a user can override the default directory with command line parameters. For archives, the files are simply extracted into a commmon top-level directory.

  The installation directory will only be created if you execute an **"Install files" action** in the installer configuration [p. 99] . By default, the "Install files" action is placed on the "Installation" screen. If your installer should not create an installation directory, you can ignore this root and remove the "Install files" action.

  To learn more on the various installer modes, please see the corresponding help topic [p. 32] .
- If your application needs to install files into directories outside the main installation directory, you can add **custom roots** to the distribution tree. This is done with the **[New Root]** action in the add menu on the right side or in the context menu. The actual location of this root is defined by its name and has to resolve to a valid directory at runtime. There are several possibilities for using custom roots. The name of a custom root can be

  - **a fixed absolute path known at compile-time**

    This works for custom environments where there's a fixed policy for certain locations. For example, if you have to install some files to `D:\apps\myapp`, you can enter that path as the name for your custom root.

If you build installers for different platforms, that root is likely to be different for each platform. In that case, you can use a compiler variable [p. 58] for the name of the custom root and override its value for each media file [p. 188] .

- **an installer variable that you resolve at runtime**

  If you would like to install files into the directory of an already installed application, such as a plugin for your own application, you can use an installer variable that you resolve at runtime. Installer variables have an `installer:` prefix, such as `${installer:rootDir}`, and can be set in a variety of ways [p. 17] .

  The most common case would be to add a "Directory selection" screen to the screen sequence [p. 99] and set its variable name property to the variable that you've used as the name of the custom root. For the above example, that would be "rootDir" (without the ${installer:...} variable syntax).

  Alternatively, you could use a "Set a variable" action to determine the location programmatically.

- **a pre-defined installer variable**

  install4j offers several variables for "magic folders" that point to common directories, such as `${installer:sys.userHome}` which resolves to the user home directory or `${installer:sys.system32Dir}` which resolves to the *`system32`* directory on Windows.

If a custom installation root is not bound at runtime or if it points to an invalid directory, the contained files will not be installed. There will be no error messages, if you require error handling, you can use a "Run a script" action before the "Install files" action with the appropriate error message and failure strategy.

Note: For archive media file types [p. 178] , custom installation roots are not installed. If you require these custom roots for your installation, you cannot use archives.

An alternative way to redirect installed files to different directories is to use the "Directory resolver" property of the "Install files" actions. Also, the "File filter" property of that action can be used to conditionally install files. The use of these properties is only recommended if you require their full flexibility. Otherwise, using custom installation roots and installation components [p. 73] is a better approach.

Beneath an installation root, you can add files or create folders:

- To create a folder, use the **[New folder]** action in the add menu on the right side or in the context menu. A folder named "New Folder" will be created below the selected directory. If no directory or installation root is selected, it will be created below the "Installation directory" root node. Right after its creation, the default name is editable and you can enter the intended name of the folder. Confirm your entry with `Enter`. To configure further properties of the folder, you can edit the folder node (see below) to show the folder property dialog [p. 75] .
- To add files, use the **[Add files and directories]** action in the add menu on the right side or in the context menu. The file wizard [p. 66] will be displayed.

In the distribution tree you can

- **Move entries**

  Entries are moved by dragging them with the mouse to the desired location. Both directories, file entries and directory content entries can be moved. To select a target directory inside a closed directory while dragging, hover with the mouse over the closed directory and it will open after a short delay. While dragging, the insertion bar shows you where the entry would be dropped.

- **Delete entries**

  Entries can be deleted by hitting the `DEL` key or using the corresponding tool bar button or menu entry.

- **Rename entries**

  Some types of entries can be renamed by using the **[Rename]** action on the right side of the tree or from the context menu. The name of the entry can then be edited in-place.

  Renaming entries is possible for:

  - File sets
  - Roots
  - Folders

- **Edit the contents of entries**

  The contents of some types of entries can be edited by using the **[Edit]** action on the right side of the tree or hitting the `ENTER` key while the entry is selected.

  Editing entries is possible for:

  - **Folders**

    Editing a folder means opens the folder property dialog [p. 75] .

  - **Single file entries**

    Editing a single file entry will bring up the file wizard [p. 66] . Only the selected file will be shown in the "Select files" step, even if you initially selected multiple files with the wizard. If you add additional files in this step, they will be added below the selected file in the distribution tree. If you delete the selected file in this step, it will also be deleted in the distribution tree.

  - **Directory content entries**

    Editing a directory content entry will bring up the file wizard [p. 66] .

Using compiler variables [p. 17] in the distribution tree allows you to make **compile-time conditional includes**:

- if a directory node resolves to the empty string after variable replacement, the directory and any contained entries will not be included in the distribution.
- if the source directory of a "contents of directory" node resolves to the empty string after variable replacement, no files will be included through that entry.
- if the file name of a single file node resolves to the empty string after variable replacement, no file will be included.

For conditions that are evaluated at runtime or for adding platform dependent files, you should use files sets [p. 9] instead.

B.3.2.2 File Wizard

The file wizard is displayed when you invoke the **[Add files and directories]** action in the add menu on the right side of the file definition tree. To get more information about the distribution tree and related concepts, please see the overview [p. 64] .

In the first step of the file wizard you choose whether you want to add

- **the contents of a directory and its subdirectories**

  Choose this wizard type if you want to recursively add the contents of a directory. You will have the possibility of excluding certain files and subdirectories and exclude files based on their file suffix. If you would like to specify different settings for one or several files in the included directory, you have to exclude them and add them as single files in the appropriate directory.

  The subsequent steps in the wizard for this selection are:

  - Select directory [p. 68]

    Choose the directory that should be distributed.
  - Install options [p. 68]

    Select installation options like access rights or overwrite policy.
  - Uninstall options [p. 70]

    Select uninstallation options such as whether to unstall files.
  - Exclude files and directories [p. 71]

    Select files or directories that should not be distributed.
  - Exclude suffixes [p. 71]

    Enter a list of file suffixes that should be ignored,

- **a number of single files**

  Choose this wizard type if you collect a small number of files (possibly from different locations) into a single directory. Example: a number of support libraries from different directories are added into the top level directory `lib`.

  The subsequent steps in the wizard for this selection are:

  - Select files [p. 68]

    Choose the files that should be distributed.
  - Install options [p. 68]

    Select installation options like access rights or overwrite policy.
  - Uninstall options [p. 70]

    Select uninstallation options such as whether to unstall files.

B.3.2.3 Wizard steps

B.3.2.3.1 File Wizard: Select Directory

In this step of the file wizard [p. 66] , you select the directory whose contents should be recursively added to the distribution tree. This step is only shown if you select "Directory" in the first step.

You can either enter the directory manually or use the chooser button **[...]** to the right of the text field to select a directory from your file system.

B.3.2.3.2 File Wizard: Select Files

In this step of the file wizard [p. 66] , you select the files that should be added to the distribution tree. This step is only shown if you select "Single files" in the first step.

To edit the list of files you can

- **add a new entry** by clicking on the right side of the window. In the following file chooser select one or multiple files to add to the list.
- **copy a file list from the system clipboard** by clicking on the right side of the window. The file list must consist of
  - a single file entry
  - multiple file entries separated by the standard path separator (";" on Windows, ":" on Unix) or by line breaks.

  Each file entry can be

  - **absolute**

    The file entry is added as it is.
  - **relative**

    On the first occurrence of a relative path, install4j brings up a directory chooser and asks for the root directory against which relative paths should be interpreted. All subsequent relative paths will be interpreted against this root directory.

  Only unique file entries will be added to the list. If no new file entry could be found, a corresponding error message is displayed.
- **remove an existing file entry** by using the **[Remove]** action while the file is selected.
- **change the position of an existing entry** by using the **[Move Up]** and **[Move Down]** actions.

B.3.2.3.3 File Wizard: Install Options

In this step of the file wizard [p. 66] , you select options regarding the installation of the selected files and directories.

The following install options are available:

- **Overwrite policy**

  This setting determines what the installer will do if the file is already present. It does not apply for archives (including RPM archives). The overwrite policy can be one of:

  - **Always ask, except for update**

If the file is already present, the installer asks the user whether to overwrite it, regardless of the file modification dates. However, files that have been proviously installed by install4j will be overwritten.

- **Always ask**

  If the file is already present, the installer asks the user whether to overwrite it, regardless of the file modification dates and whether install4j has previously installed this file.

- **If newer, otherwise ask**

  If the file is already present, the installer silently overwrites the file if the installed file is newer, otherwise is asks the user.

- **If newer**

  If the file is already present, the installer silently overwrites the file if the installed file is newer, otherwise it does not install it.

- **Always**

  The installer silently overwrites the file in all cases.

- **never**

  The installer does not install the file.

- **Unix file and directory mode**

  On Unix-like platforms (including Linux and Mac OS X), the file mode governs the access rights to the installed files. The access mode is composed of three octal numbers (0-7) and each number completely expresses the access rights for a particular group of users:

  - **First number**

    The first octal number contains the access rights for the **owner of the file**.

  - **Second number**

    The first octal number contains the access rights for the **user group** that the file is attached to.

  - **Third number**

    The third octal number contains the access rights for **all other users**.

  For a desired combination of access rights, the octal number is calculated by adding:

  - **1**

    For the right to **execute** the file or to browse the directory. Only set this flag for directories, executables and shell scripts.

  - **2**

    For the right to **write** to the file or directory.

  - **4**

    For the right to **read** from the file or directory.

  For example, read/write rights are calculated as 2 (for writing) + 4 (for reading) = 6, read-only rights are just 4, and the rights to read/execute a file are calculated as 1 (for executing) + 4 (for reading) = 5.

  The **default access rights for files** are 644, i.e. the owner can read and write the file and all others can only read it. Since usually applications on Unix-like systems are installed by the administrator (usually called root), this means that users will only be able to read files but not to write to them. For launchers, the installer sets access rights for files to 755, which is equivalent to 644 only that

everyone can execute the launchers. If you have files that your users should be able to write to, you have to add these files to the distribution tree with a different access mode. For example, 666 would be appropriate in that case. You can reset the default mode with the **[Reset to default]** button.

The **default access rights for directories** are 755, i.e. the owner can read and write and browse the directory and all others can only read and browse it. Just as for files, this means that except for root, users will only be able to browse directories and read from them but they will not be able to create files in them. If you have directories that your users should be able to create files in, you have to add these directories to the distribution tree with a different access mode. For example, 777 would be appropriate in that case. You can reset the default mode with the **[Reset to default]** button.

B.3.2.3.4 File Wizard: Uninstall Options

In this step of the file wizard [p. 66] , you select options regarding the uninstallation of the selected files and directories.

The following uninstall options are available:

- **Uninstallation policy**

  This setting determines how the uninstaller decides whether an installed file should be uninstalled or not. The uninstallation policy can be one of:

  - **If created**

    If the file or directory was created by the installer, it will be deleted.
  - **Always**

    The file or directory will always be deleted regardless of whether it was created by the installer. Please be careful when choosing this option, since deleting directories that were not created by the installer can have severe unintended consequences.
  - **Never**

    The file or directory will not be deleted ny the uninstaller.
  - **If created, but not for update**

    If the file or directory was created by the installer, it will be deleted. However, if the uninstaller is running as part of the update (invoked by an "Uninstall previous installation" action), the file or directory will not be deleted.
  - **Always, but not for update**

    The file or directory will always be deleted regardless of whether it was created by the installer. However, if the uninstaller is running as part of the update (invoked by an "Uninstall previous installation" action), the file or directory will not be deleted. Please be careful when choosing this option, since deleting directories that were not created by the installer can have severe unintended consequences.

- **Shared file (Windows only)**

  Microsoft Windows has a concept of "shared files" where a usage counter is monitored for each file or directory. When the usage counter reaches zero the installer will delete the file or directory. This is especially useful if you install DLLs into the system32 directory that are shared by multiple applications.

B.3.2.3.5 File Wizard: Excluded Files and Directories

In this step of the file wizard [p. 66] , you can select files and subdirectories that should be excluded from distribution. This step is only shown if you select "Directory" in the first step.

The tree labeled "Excluded files and subdirectories" shows the tree of all files in the directory selected in the previous step [p. 68] . Each file and subdirectory has a check box attached. If you select that check box, the entry will **not** be distributed. Selections of subdirectories are recursive. If you select a subdirectory, its contents are hidden from the tree since they will be excluded anyway.

B.3.2.3.6 File Wizard: Excluded Suffixes

In this step of the file wizard [p. 66] , you can enter file name suffixes that should be excluded from distribution. This step is only shown if you select "Directory" in the first step.

In addition to the explicit selections of excluded files and subdirectories in the previous step [p. 71] , a list of file name suffixes separated by commas can be entered here to exclude them from the distribution. For example, entering `*.java, *.java~` will prevent files with these extensions from being distributed.

B.3.3 Files - Viewing the Results

On this tab you can check the results of your definition of the distribution tree [p. 64] .

The tree shows all files that will be distributed in the generated media files [p. 177] .

You cannot remove files from this tree or add them to it. If you would like to remove a file that has been added with a directory entry, you have to use the excluded files and directories step [p. 71] or the excluded suffixes step [p. 71] in the files directory wizard. To exclude files and directories on a per-media set basis, please see the customize project defaults [p. 188] step in the media file wizard [p. 180] .

On activating this tab, the file tree is re-read if the definition of the distribution tree [p. 64] has changed since the last time the file tree was shown. This background process can take a short while and is indicated by a `"Please wait ..."` entry in the result tree.

Should the contents of your hard disk have been modified in the meantime, you can use the **[Refresh]** button to re-read the displayed file tree.

B.3.4 Files - Defining Installation Components

On this tab you can optionally define installation components.

Installation components can be used to allow the user to customize the installation. GUI installers will present a step that lists all available installation components in a tree with check boxes and lets the user choose which components to install. Console installers will also present a list of installation components to the user for selection. If no installation components are defined, that step will be omitted and the entire distribution tree is installed.

On the left side you configure a tree of installation components and component folders. To every component folder you can add installation components and component folders as child nodes. The component tree is **drag-and drop enabled**.

In the component tree you can

- **Move entries**

  Components or component folders are moved by dragging them with the mouse to the desired location. To select a target folder inside a closed component folder while dragging, hover with the mouse over the closed component folder and it will open after a short delay. While dragging, the insertion bar shows you where the entry would be dropped.
- **Add installation components**

  With the **[Add Installation Component]** action, a new installation component is added to the currently selected component folder, or at the top-level if no component folder is selected. The name of the installation component can be edited in-place immediately.
- **Add component folders**

  With the **[Add Component Folder]** action, a new component folder is added to the currently selected component folder, or at the top-level if no component folder is selected. The name of the component folder can be edited in-place immediately.
- **Delete entries**

  With the **[Delete]** action or the `DEL` key, you can remove the currently selected installation component or component folder. All child nodes of component folders are removed as well.
- **Rename entries**

  With the **[Rename]** action, you can rename an installation component or a component folder.

To internationalize the name of the component for different media files, please use custom localization keys [p. 17] .

When using the install4j API [p. ?] , you reference installation components with IDs. You can show IDs in the component tree by activating the **[Show IDs]** button on the lower right side of the component tree. The automatically generated numerical IDs are then shown in brackets. The selection will be remembered across restarts of install4j.

This ID can be used in expressions, scripts and custom code when you want to check if the installation component has been selected for installation. A typical condition expression for an action would be `context.getInstallationComponentById("123").isSelected()` if the ID of the component is "123". In this way you can conditionally execute actions depending on whether a component is selected or not.

The right pane displays the properties of the selected element in the component tree. The options are organized into several tabs. There are different configuration options, depending on whether you've selected an installation component or a component folder:

- **Installation component**

  Installation components have the following specific tabs:

  - **Files**

    To choose the contents of an installation component, you first have to decide whether the component contains all files or just a selection of files or directories. For a selection of files and directories, you then choose the desired contents in the tree. Installation components are not mutually exclusive and you can include the same files in multiple installation components.
  - **Options**

    The available options are:

    - **Initially selected for installation**

      Whether the check box for the currently selected installation component is selected or not.
    - **Mandatory component**

      Whether the currently selected component must be installed or not. If the component is mandatory, the user cannot deselect it and the check box in the installer is grayed out.
    - **Downloadable component**

      Whether the currently selected component should be externalized for installers whose data file type [p. 184] is set to "Downloadable". These components can then be placed on a web server and are downloaded on demand if the users selects them.

  - **Dependencies**

    If the currently selected installation component only works if a number of other components are installed as well, you can select those components on the "Dependencies" tab. When this installation component is selected, the dependencies become automatically selected and mandatory. When this installation component is deselected again, the previous selection state of the dependencies is restored. The list of components in the "Dependencies" tab only shows components that will not lead to circular dependencies.

- **Component folder**

  Component folders have an "Options" tab where you can configure whether the component folder should be initially expanded or not.

Both installation components and component folders also have a **Description** tab. You can optionally display a description below each component in the installer. Any component or component folder with a description will have a toggle button with help icon on the right side. This toggle button controls whether the description is displayed below the element. You can also use the `F1` key to toggle the visibility of the description. The **Expand description automatically** check box allows you to show descriptions by default.

Note: The user can only select which installation components should be installed if the "Installation components" screen or the "Installation type" screen is part of the screen sequence [p. 99] .

The "Installation type" screen offers a selection between sets of installation components, such as "Full", "Standard" and "Custom", while the "Installation components" screen shows the tree of components that you define on this tab with check boxes in front of each node. The "Installation components" screen has a number of properties that let you customize the appearance of the descriptions. If both are present, the "Installation components" screen will only be shown if the selected installation type was configured to be customizable.

B.3.5 Dialogs

B.3.5.1 Distribution tree file chooser dialog

The distribution file chooser dialog shows files or directories in the distribution tree. This tree does not necessarily correspond to a portion of the filesystem of your hard disk, since a virtual folder hierarchy with arbitrarily mounted directories from your hard disk can be defined on the Definition tab [p. 64] of the Files step [p. 63] .

The shown files or directories are a subset of the result tree [p. 72] in the Files step [p. 63] . The actual filter depends on the particular context of your action and is displayed in the title bar of the dialog.

Should the contents of your hard disk have been modified in the meantime, you can use the **[Refresh]** button to re-read the displayed file tree.

B.3.5.2 Folder properties dialog

The folder properties dialog is displayed when you edit a folder in the distribution tree [p. 64] .

In the folder properties dialog you can set the **access rights** for the selected folder. On Unix-like platforms (including Linux and Mac OS X), the file mode governs the access rights to the installed directories. The access mode is composed of three octal numbers (0-7) and each number completely expresses the access rights for a particular group of users:

- **First number**

  The first octal number contains the access rights for the **owner of the file**.
- **Second number**

  The first octal number contains the access rights for the **user group** that the file is attached to.
- **Third number**

  The third octal number contains the access rights for **all other users**.

For a desired combination of access rights, the octal number is calculated by adding:

- **1**

  For the right to **browse** the directory.
- **2**

  For the right to **write** to the directory.
- **4**

  For the right to **read** from the directory.

The **default access rights for directories** are 755, i.e. the owner can read and write and browse the directory and all others can only read and browse it. Since usually applications on Unix-like systems are installed by the administrator (usually called root), this means that except for root, users will only be able to browse directories and read from them but they will not be able to create files in them. If you have directories that your users should be able to create files in, you have to set a different access mode for them. For example, 777 would allow all users to create, read, write and delete files in the directory. You can reset the default mode with the **[Reset to default]** button.

B.4 Step 3: Launchers

B.4.1 Step 3: Configure Launchers

Launchers are responsible for starting your application. There are two types of launchers:

- **Generated launchers**

  install4j can generate native launchers that start your application. For example, on Windows, a `.exe` file will be created that among other things takes care of finding a suitable JRE, displaying appropriate error messages in case of need and then starts your application. Using launchers generated by install4j has numerous advantages as compared to using home-grown batch files and shell scripts.

  Each launcher definition is compiled separately for each defined media set [p. 177] . Therefore, for the majority of all cases, a single launcher definition will be sufficient to start your application. If, for example, your distribution contains two GUI applications and a command line application, you have to define 3 launchers, regardless of how many media files [p. 177] you define.

  When your application is started with a launcher generated by install4j, you can query the system property **install4j.appDir** to get the installation directory and and **install4j.exeDir** to get the directory where the launcher resides. Use `System.getProperty("install4j.appDir")` and `System.getProperty("install4j.exeDir")` to access these values.

- **External launchers**

  If you already have an external launcher for your application, you can let install4j use that launcher instead of generating one. Since external launchers are most likely platform dependent, you will have to add external launchers for each platform that is targeted by your media files [p. 177] . Make sure to exclude the irrelevant launchers [p. 188] in your media file definitions in this case.

To define a new launcher, you double-click on the new launcher entry in the list of defined launchers or choose *Launcher->New launcher* from install4j's main menu. The launcher wizard [p. 78] will then be displayed. Once you have completed all steps of the launcher wizard, a new launcher entry will be displayed in the list of launchers. The icon of a launcher indicates if it is a

- GUI application launcher
- Console application launcher
- Service application launcher
- External launcher

In the list of launchers you you can

- **Reorder launcher definitions**

  Launcher definitions are reordered by dragging them with the mouse to the desired location. While dragging, the insertion bar shows you where the launcher definition would be dropped. The order of launchers is not relevant for install4j, reordering is provided only for the purpose of letting you arrange the launcher definitions according to your personal preferences.

- **Copy launcher definitions**

  Launcher definitions are copied by copy-dragging them (e.g. on Windows, press `CTRL` while dragging) or using the **[Copy Launcher]** action while the source launcher is selected.

  The name of the copied launcher definition will be prefixed with "Copy of". You can change this default name by renaming the launcher definition (see below).

- **Rename launcher definitions**

  Launcher definitions can be renamed by selecting *Rename Launcher* from the context menu or *Launcher->Rename launcher* from install4j's main menu.

  An input dialog will be displayed where the current name can be edited. Please note that the name of the launcher is for your own information only and is not used in the distribution.

- **Delete launcher definitions**

  Launcher definitions can be deleted by using the **[Delete Launcher]** action or by hitting the `DEL` key while the launcher definition is selected.

- **Edit a launcher definition**

  Launcher definitions can be edited by using the **[Edit Launcher]** action or by hitting the `ENTER` key while the launcher definition is selected.

  The launcher wizard [p. 78] will be displayed for the selected launcher definition. Please note that you can directly access any step in the wizard by clicking on it in the index.

B.4.2 Launcher Wizard

The launcher wizard is displayed when you add a new launcher or when you edit an exiting launcher. To learn more information about the launchers, please see the overview [p. 76] .

In the first step of the launcher wizard you choose whether you want to create

- **a generated launcher**

  The subsequent steps with the associated advanced options [p. 79] capture all information required to start your Java application.
- **an external launcher**

  The external launcher wizard queries the following data:

  - **Launcher executable**

    Enter the path to the executable in the distribution tree. You can select a file from the distribution tree by clicking the **[...]** chooser button.
  - **Menu integration**

    The menu integration options are the same as for the generated launcher [p. 91] .

B.4.3 Wizard steps

B.4.3.1 Launcher Wizard: Configure Executable

In this step of the launcher wizard [p. 78] , you enter the properties of the executable that is to be generated.

The following properties of the executable can be edited in the `Executable` section of this step:

- **Executable type**

  Executables created by install4j can be either GUI applications, console applications or service applications

  - **GUI application**

    There is no terminal window associated with a GUI application. If stdout and stderr are not redirected (see the redirection advanced step [p. 85]), both streams are inaccessible for the user. This corresponds to the behavior of *`javaw(.exe)`*.

    If your GUI application uses **SWT or QT Jambi instead of Swing**, please select the `uses SWT or QT` check box below this radio button. This is mainly important for correct behavior on Mac OS X where the application must be started differently in this case. The executables on Microsoft Windows always include the XP manifest, so that on Windows XP, SWT applications are displayed in XP style and not in Windows 2000 style.

  - **Console application**

    A console application has an associated terminal window. If a console application is opened from the Windows explorer, a new terminal window is opened. If stdout and stderr are not redirected (see the redirection advanced step [p. 85]), both streams are printed on the terminal window. This corresponds to the behavior of *`java(.exe)`*.

  - **Service application**

    A service runs independently of logged-on users and can be run even if no user is logged on. A service cannot rely on the presence of a console, nor can it open windows. On Microsoft Windows, a service executable will be compiled by install4, on Mac OS X a startup item will be created and on Unix-like platforms a start/stop script will be generated.

    When you develop a service please note the following requirement: The `main` method will be called when the service is started.

    To handle the shutdown of your service, you can use the `Runtime.addShutdownHook()` method to register a thread that will be executed before the JVM is terminated.

    For information on how services are installed or uninstalled, please see the help on service options [p. 86] .

- **Executable name**

  Enter the desired name of the executable without any trailing `.exe` or `.sh`.

- **File set**

  Choose the file set to which the launcher should be added. File sets are defined in the distribution tree [p. 64] . If you do not use different file sets, "Default file set" will be the only option which is activated by default.

- **Directory**

  Enter the directory in the distribution tree where the executable should be generated. If you leave this field empty, the executable will be generated in the installation root directory. You can select a directory from the distribution tree by clicking the **[...]** chooser button.

- **Allow only a single running instance of the application**

  If you select this check box, the generated executable can only be started once. Subsequent user invocations will bring the application to the front. In the `StartupNotification` class of the install4j launcher client API you can register a startup handler to receive the command line parameters. In this way, you can handle file associations with a single application instance. This feature is only available on Microsoft Windows, on Mac OS X, single bundle media files always behave this way.

- **Fail if an exception in the main thread is thrown**

  Executables created by install4j can monitor whether the main method throws an exception and show an error dialog in that case. This provides a generic startup error notification facility for the developer that handles a range of errors that would otherwise not be notified correctly. For example, if an uncaught exception is thrown during application startup, a GUI application might simply hang, leaving the user in the dark about the reasons for the malfunction. With the error message provided by the install4j executable, reasons for startup errors are found much more easily.

- **Working directory**

  For some applications (especially GUI applications) you might want to change the working directory to a specific directory relative to the executable, for example to read config files that are in a fixed location. To do so, please select the `Change working directory to` check box and enter a directory relative to the executable in the adjacent text field. To change the current directory to the same directory where the executable is located, please enter a single dot.

B.4.3.2 Launcher Wizard: Define Launcher Icon

In this step of the launcher wizard [p. 78] , you define the icon for the generated launcher.

If you would like to associate a custom icon with your launcher, select the "add icon to launcher" check box.

- In the `Cross platform` section you then have to choose icon files in the PNG image format (extension *`*.png`*) in the sizes 16x16 and 32x32 pixels. On Microsoft Windows, the generated executable will have an icon with these images, on other platforms, these image files will be used for desktop integration. It is recommended to use 32-bit images with an alpha channel, 8 bit-palette images will be generated where required. Generated Windows icons contain traditional 256 color images and 32-bit images with an alpha channel ("Windows XP icons"). However, it is also possible to use 8 bit-palette images with a transparency color for the input image files.
- If you have an **external icon file for Microsoft Windows**, you can select the `Use ICO file` option in the `Windows` section and choose an icon file (extension *`*.ico`*) in the text field below. With the `Generate from PNG files` option, the icon will be generated as described in the `Cross platform` section.
- If you have an **external icon file for Mac OS X**, you can select the `Use ICNS file` in the `Mac OS X` section and choose a Mac OS X icon file (extension *`*.icns`*) in the text field below. With the `Use standard icon` option, a standard icon provided by install4j will be used. No Mac OS X icon is generated by install4j, since the ICNS format is undocumented. Mac OS X icons have to be generated on Mac OS X with the Icon Composer application located in *`/Developer/Applications`*.

Note: If the project has already been saved, relative file paths will be interpreted as relative to the project file.

B.4.3.3 Launcher Wizard: Configure Java Invocation

In this step of the launcher wizard [p. 78] , you enter the information required to start your application.

The following properties of the Java invocation can be edited in the `General` section of this step:

- **Main class**

  Enter the fully qualified main class of your application. Next to the text field is a **[...]** chooser button that brings up a dialog with a list of all public main classes [p. 96] in the class path. To use this facility, you have to set up your classpath first (see below).

- **VM parameters**

  If there are any VM parameters you would like to specify for the invocation of your Java application, you can enter them here (e.g. `-Dmyapp.myproperty=true` or `-Xmx256m`).

  Note: You must quote parameters that contain spaces. Please quote the entire parameter like `"-Dapp.home=${installer:sys.launcherDirectory}"` and not just the value. Incorrect quoting will lead to failure of the launcher.

  Please read the help topic on VM parameters [p. 22] for more information on how install4j can help you with **adjusting the VM parameters at runtime**.

- **Arguments**

  If you need to specify arguments for your main class, you can enter them here. Arguments passed to the executable will be appended to these arguments.

- **Allow VM passthrough parameters**

  If you would like to allow the user to specify VM parameters with the syntax `-J[VM parameter]` (e.g. `-J-Xmx512m`), select the `Allow VM passthrough parameters` check box.

  Note: This setting applies only to Windows launchers. On Unix platforms you can use the `INSTALL4J_ADD_VM_PARAMS` environment variables to add VM parameters to the launcher. On Mac OS X, you can edit the *`Info.plist`* file to change the VM parameters.

In the `Class path` section of this step you can configure the class path and the error handling for missing class path entries. The class path list shows all class path entries that have been added so far. The following types of class path entries [p. 96] are available:

- Scan directory
- Directory
- Archive
- Environment variable

The symbol prepended to an entry indicates that an error with that entry will lead to a startup failure with an error message displayed to the user.

The control buttons on the right allow you to modify the contents of the class path list:

- Add class path entry (key `INS`)

  Invokes the class path entry dialog [p. 96] . Upon closing the class path entry dialog with the **[OK]** button, a new class path entry will be appended to the bottom of the class path list.

- Remove class path entry (key `DEL`)

Removes the currently selected class path entry.

- Move class path entry up (key `ALT-UP`)

  Moves the selected class path entry up one position in the class path list.

- Move class path entry down (key `ALT-DOWN`)

  Moves the selected class path entry down one position in the class path list.

To change the error handling mode of a class path entry [p. 96] , select the class path entry and press **[Toggle 'fail on error']** right below the class path list or choose the corresponding menu item from the context menu.

B.4.3.4 Launcher Wizard: Configure Splash Screen

In this step of the launcher wizard [p. 78] , you can configure a splash screen for your application.

install4j offers three different splash screen options

- **No splash screen**
- **install4j splash screen**

  This is a splash screen whose implementation is provided by install4j. On Windows, this splash screen will be displayed by native code an provides for extremely fast user feedback. This mode works with all supported JVMs. Further configuration settings are available on the Advanced splash screen options tab [p. 94] .
- **Java 6+ splash screen**

  From Java 6 on, Java provides a splash screen implementation. If you would like to use this option, your project must at least specify a Java minimum version of 1.6 on the Java Version tab [p. 52] of the General Settings [p. 50] .

  Please note that you cannot specify the `-splash:` VM parameter in the , since this VM parameter is parsed by the default Java launchers (*`java.exe`* and *`javaw.exe`*) which are not used by install4j for Windows launchers.

If you decide to display a splash screen, you have to enter an image file. The file format can be PNG or GIF. If the project has already been saved, a relative file path will be interpreted as relative to the project file.

B.4.3.5 Advanced options

B.4.3.5.1 Launcher Wizard: Configure Redirection

In this step of the launcher wizard [p. 78] , you can configure the redirection settings for stderr and stdout.

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 79] and choosing "Redirection" from the **[Advanced options]** popup menu or by clicking directly on the index.

The following redirection settings can be edited:

- **Redirection of stderr**

  To redirect stderr to a file, select the `Redirect stderr` check box and enter a file name in the adjacent text field. The file name is interpreted relative to the executable. Enter `/dev/null` if you want to suppress output completely for all platforms.

- **Redirection of stdout**

  To redirect stdout to a file, select the `Redirect stdout` check box and enter a file name in the adjacent text field. The file name is interpreted relative to the executable. Enter `/dev/null` if you want to suppress output completely for all platforms.

B.4.3.5.2 Launcher Wizard: Configure Service Options

In this step of the launcher wizard [p. 78], you define further options for service executables. All options on this screen will only be enabled if the selected executable type in the "Executable" step [p. 79] is "Service".

Note: this advanced option screen is reachable by selecting the executable step [p. 79] and choosing "Service options" from the **[Advanced options]** popup menu or by clicking directly on the index.

The `Startup options` section is **only relevant for Microsoft Windows**.

Windows services are registered by the installer. It is also possible to install services from the command line by passing `/install` to the generated service executable. The default start mode of the service can be determined in this section:

- **Default start type**
  - **Start on demand**

    In start on demand mode, your service must be manually started by the user in the Windows service manager. Use this option, if you're not sure if your users will actually want to run your application as a service, but you want to give them an easy way to do so. This installation mode can be forced on the command line if the user passes `/install-demand` to the generated executable instead of `/install`.
  - **Auto start**

    In auto start mode, your service is always started when Windows is booted. This installation mode can be forced on the command line if the user passes `/install-auto` to the generated executable instead of `/install`.

Windows services are always uninstalled by passing `/uninstall` to the generated service executable. All command line switches also work with a prefixed dash instead of a slash (like `-uninstall`) or two prefixed dashes (like `--uninstall`).

To start or stop the service, the `/start` and `/stop` options are available. In addition, a `/status` argument shows if the service is already running. The exit code of the status command is 0 when the service is running, 3 when it is not running and 1 when the state cannot be determined (for example when it is not installed on Windows).

As a second parameter after the `/install` parameter, you can optionally pass a **service name**. In that way you can

- install a service with a different service name than the default name.
- Use the same service executable to start multiple services with different names. To distinguish several running service instances at runtime, you can query the system property `exe4j.launchName` for the service name. Note that you also have to pass the same service name as the second parameter if you use the `/uninstall`, `/start` and `/stop` parameters.

For Unix service executables, the `start`, `stop` and `status` arguments are available for the generated start script. The stop command waits for the service to shut down. The exit code of the status command is 0 when the service is running and 3 when it is not running.

If your service depends on another service, say a database, you can enter the service name (the name of the startup item on Mac OS X) of the other service in the platform specific `Dependencies` section.

You do not have to enter core OS services such as filesystem or network, these services will always be initialized before your service is launched. If you have dependencies on multiple services, you can enter a list of these service names separated by commas. Text fields for specifying dependencies are available for Windows and Mac OS X. On Unix-like platforms, the start/stop script has to be integrated into the boot sequence by the administrator.

In most cases, you can leave the dependencies empty.

B.4.3.5.3 Launcher Wizard: Configure Windows Version Info Resource

In this step of the launcher wizard [p. 78] , you can configure whether a version info resource should be generated for the Microsoft Windows executable and what values the version info fields should take. **This step is only relevant for Microsoft Windows** and is important if your application wants to obtain the "Designed for Windows" logo.

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 79] and choosing "Windows version info" from the **[Advanced options]** popup menu or by clicking directly on the index.

A version info resource will enable the Windows operating system to determine meta information about your executable. This information is displayed in various locations. For example, when opening the property dialog for the executable in the Windows explorer, a "Version" tab will be present in the property dialog if you have chosen to generate the version info resource.

The version info resource consists of several pieces of information. If you check `Generate version info resource`, there are several fields whose values must be entered in the text fields on this step. Note that the "original file name", the "company name", the "product name" and the "product version" fields in the version info resource are filled in automatically by install4j.

- **File version**

  If you want to specify a version for the file which is a different from the product version, you can do it here. If this field is left empty, the product version entered on the Application Info tab [p. 51] of the General Settings step [p. 50] will be used for the file version.
- **Internal name**

  Choose a short internal name for identifying your application.
- **File description**

  Enter a description of the application.
- **Legal copyright**

  Enter a copyright statement for your application.

B.4.3.5.4 Launcher Wizard: Vista Execution Level

In this step of the launcher wizard [p. 78] , you can configure the execution level for the launcher executable on Windows Vista.

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 79] and choosing "Vista execution level" from the **[Advanced options]** popup menu or by clicking directly on the index.

The execution level can be one of

- **As invoker**

  This is the default setting. The executable will be executed with the rights of the current token. If the user is an Administrator, this will be a filtered token so the executable will not have all administration rights.

- **Highest available**

  This level will raise the rights of the executable to the maximum extend available for the current user. This applies to Administrators that usually run with a filtered token. Windows Vista will show a question to the user if he wants to elevate the rights of this application. For a standard user this is the same as "As invoker".

- **Require administrator**

  This is the same as "Highest available" when the user is an Administrator running with a filtered token. If the user is a standard user, Windows Vista will ask for the credentials of an Administrator account.

B.4.3.5.5 Launcher Wizard: Custom Unix Launcher Script

In this step of the launcher wizard [p. 78] , you can configure an optional custom script for Unix launchers.

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 79] and choosing "Unix launcher script" from the **[Advanced options]** popup menu or by clicking directly on the index.

If you specify a bourne shell custom script, the entered script fragement will be inserted into the launcher script immediately before the Java invocation of your launcher takes place. This is a hook for experienced users to make custom changes in the environment.

You can select one of:

- **No custom script**

  No custom script will be inserted.
- **Custom script from file**

  Specify a file from which the custom script will be read. If you enter a relative file, the file will be interpreted relative to the project file.
- **Direct entry of custom script**

  Enter your custom script in the text area below.

B.4.3.5.6 Launcher Wizard: Configure Menu Integration

In this step of the launcher wizard [p. 78] you customize the start menu integration of the launcher.

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 79] and choosing "Menu name" from the **[Advanced options]** popup menu or by clicking directly on the index.

The "Create standard program group" action [p. 117] optionally adds menu entries for launchers on Microsoft Windows and creates links for launchers in a suitable directory on Unix. Please choose one of three possibilities:

- **Integrate into menus with standard name**

  By default, the name of the launcher configuration in the list of launchers [p. 76] will be used for any desktop integration of the launcher, such as the start menu entry in Windows.

- **Integrate into menus with custom name**

  To use a different name for the menu integration, choose this option and enter the desired name in the text field below. To put the launcher in a sub-folder in the Windows program group, just enter a path (like `Client\Launcher`) here or use a compiler variables [p. 17] to make this change for the Windows media file definitions only.

- **Exclude from menu integration**

  To entirely exclude this launcher from any menu integration, choose this option. If this option is chosen, no links will be generated for this launcher on Unix by the "Create standard program group" action.

B.4.3.5.7 Launcher Wizard: Configure Native Library Directories

In this step of the launcher wizard [p. 78] , you can configure directories that contain native libraries.

Note: this advanced option screen is reachable by selecting the "Java invocation" step [p. 82] and choosing "Native libraries" from the **[Advanced options]** popup menu or by clicking directly on the index.

If your application uses native libraries that you would lke to load with a `System.loadLibrary()` call, the directory where the native library is located must be included in a system-dependent environment variable. You can add such directories in the path list of this step.

- Add native library directory (key `INS`)

  Lets you add a new directory to the end of the list. The native libraries entry dialog [p. 97] will be displayed. You can use compiler variables [p. 17] to change native library directories for different media files. For this purpose, you can define one variable and override it in each media file definition.

- Remove native library directory (key `DEL`)

  Removes the currently selected native library directory entry.

- Move entry up (key `ALT-UP`)

  Moves the selected native library directory entry up one position in the path list.

- Move entry down (key `ALT-DOWN`)

  Moves the selected native library directory entry down one position in the path list.

B.4.3.5.8 Launcher Wizard: Choose Preferred VM

In this step of the launcher wizard [p. 78] , you can configure the preferred VM that install4j will choose to invoke your application. This setting only influences the choice of the VM type after a JRE has been selected according to the search sequence. The search sequence for the JRE is specified on the Java Version tab [p. 52] of the General Settings step [p. 50] .

Note: this advanced option screen is reachable by selecting the "Java invocation" step [p. 82] and choosing "Preferred VM" from the **[Advanced options]** popup menu or by clicking directly on the index.

After install4j finds a suitable JRE or JDK, it tries to honor the setting you make in this step. You can select one of the following:

- **Default VM**

  install4j will use the default VM for the found JRE.
- **Client hotspot VM**

  install4j will try to use the client hotspot VM for the found JRE. This is equivalent to using the `-client` switch when invoking `java` from the command line.
- **Server hotspot VM**

  install4j will try to use the server hotspot VM for the found JRE. This is equivalent to using the `-server` switch when invoking `java` from the command line.

Please note that it is not an error if the selected JVM is not present for the found JRE. install4j will simply use another JVM to launch your application in that case.

B.4.3.5.9 Launcher Wizard: Splash Screen Options

In this step of the launcher wizard [p. 78] , you can configure splash screen options.

Note: this advanced option screen is reachable by selecting the "Splash screen" step [p. 84] and choosing "Splash screen options" from the **[Advanced options]** popup menu or by clicking directly on the index.

The behavior of the splash screen can be defined in the `General` section of this step:

- **Hide splash screen when first application window is shown**

  If this option is checked, the executable generated by install4j will monitor the state of your application and hide the splash screen as soon as a window is opened. If you want to hide the splash screen programmatically, you can use install4j's splash screen client API.
- **Splash screen is always on top**

  If this option is checked, the splash screen remains always on top of other windows opened by your application. On some platforms, this option has no effect.

The `Status line` and `Version line` sections allow you to position the text lines on the splash screen and configure their font. The status line is dynamically updatable with install4j's splash screen client API while the text of the version line may be overridden with a command line option [p. 198] of the install4j compiler.

You can configure the following properties of a text line

- **Text**

  The (initial) text displayed in the text line.
- **Position**

  The x and y-coordinates of the text line on the splash screen. The origin of the coordinate system is the top left corner of the splash screen window.
- **Font**

  The font used for drawing the text line:

  - **Name**

    The name of the font. Please choose a common font name that is likely to be available on all target platforms. If unavailable at runtime, a platform dependent standard dialog font will be used as a fallback.
  - **Weight**

    The weight of the font. 8 fine-grained font weights are offered as a choice.
  - **Size**

    The size of the font in points.
  - **Color**

    The color of the font. By clicking on **[...]**, a color chooser dialog is brought up.

In both text lines, you can use the `%VERSION%` variable to substitute the version entered on the Application Info tab [p. 51] of the General Settings step [p. 50] .

To **visually position the text lines** with mouse and keyboard on the actual splash screen image, please click on the **[Position text lines visually]** button. The visual positioning dialog [p. 97] will then

be displayed. On exiting the dialog with the **[OK]** button, the X/Y coordinate text fields (see above) will be updated for both text lines.

B.4.3.6 Dialogs

B.4.3.6.1 Main Class Selection Dialog

The main class selection dialog is shown when clicking on the **[...]** chooser button next to the main class text field in the Java invocation step [p. 82] . It shows all classes with a public main method.

Please choose a main class from the list and confirm with **[OK]** or double-click on the selected class.

B.4.3.6.2 Classpath Entry Dialog

The class path entry dialog is shown when clicking on the add button in the "Configure Java Invocation" step [p. 82] of the launcher wizard [p. 78] . Upon closing this dialog with the **[OK]** button, a new class path entry will be appended to the bottom of the class path list of that step.

To define a class path entry, you first select the entry type, then check the `fail if an error occurs with this class path entry` check box in case you want the startup to be terminated if this class path entry is faulty and finally fill out the `Detail` section of the dialog which is dependent on the selected entry type. The following entry types are available:

- **Scan directory**

  Scan a directory for archives with the extensions *`*.jar`* and *`*.zip`* to be added to the class path. In the `Detail` section of the dialog you must choose a directory either by entering the path in the text field or by clicking **[...]** and choosing it with a file chooser.

  Error handling:

  If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this directory does not exist.

- **Directory**

  Add a directory to the class path. In the `Detail` section of the dialog you must choose a directory either by entering the path in the text field or by clicking **[...]** and choosing it with a file chooser.

  Error handling:

  If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this directory does not exist.

- **Archive**

  Add an archive with the extension *`*.jar`* or *`*.zip`* to the class path. In the `Detail` section of the dialog you must choose an archive either by entering the path in the text field or by clicking **[...]** and choosing it with a file chooser.

  Error handling:

  If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this archive does not exist.

- **Environment variable**

  Add the contents of an environment variable to the class path. In the `Detail` section of the dialog you must enter the name of an environment variable.

  Error handling:

  If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this environment variable is not defined.

Except for the "Environment variable" classpath type, you can use environment variables in the text field with the following syntax: `${VARIABLE_NAME}` where you replace VARIABLE_NAME with the desired environment variable.

Note that for path selections by means of a file chooser (**[...]** buttons), install4j will try to convert the path to be relative to the distribution source directory.

B.4.3.6.3 Native Libraries Entry Dialog

The native libraries entry dialog is shown when clicking on the Add button in the Native libraries [p. 92] advanced options step below Java invocation step [p. 82] .

Please enter a directory that contains native libraries by entering the relative path to the distribution tree root directly or choosing it with the **[...]** chooser button next to the text field. You can use compiler variables [p. 17] to change native library directories for different media files. For this purpose, you can define one variable and override it in each media file definition [p. 188] .

B.4.3.6.4 Visual Positioning of Text Lines

The visual positioning dialog is shown when clicking on the **[Position text lines visually]** button in the "configure splash screen" step [p. 84] of the launcher wizard [p. 78] . Upon closing this dialog with the **[OK]** button, the X/Y coordinate text fields will be updated for status and version text lines in that step.

The visual positioning dialog displays the selected image with overlaid status and text line placeholders that are surrounded on the left and bottom by lines. These lines flash for the selected text line. You can position the selected text line on the image by dragging it with the mouse or using the cursor keys. Pressing `CTRL` with the cursor keys moves the text line in larger steps.

Please note that only the font color is reflected in the font of the text line placeholders. Font weight, font size and font name are only used in the runtime version of the splash screen.

B.5 Step 4: Installer

B.5.1 Step 4: Configure the Installer

In the **Installer step**, you configure all aspects of your installer, most importantly the screens and actions representing the user input and the actual installation.
The Installer step is divided into several tabs which are located at the bottom of install4j's main window:

- Screens & actions [p. 99]

  On this tab you configure the screens and actions in your installer and uninstaller as well as custom installer applications.

- Custom Code [p. 168]

  On this tab you configure the location of your custom code for additional libraries to be used in scripts as well as your own implementations of actions, screens and form components.

- Update Options [p. 169]

  On this tab you configure how your installers handle installations when an earlier version has already been installed.

B.5.2 Installer - Screens and Actions

For more information on screens and related concepts, please see the corresponding help topic [p. 11] .

The screens and actions tab shows a tree representation of the installer, the uninstaller and other installer applications, such as updaters. The nodes in the tree are of the following types:

- Applications [p. 102]

  An application consist of a series of screens.
- Screens [p. 105]

  A screens displays information to the user, optionally gathers user input and optionally executes a series of actions when the user moves to the next screen.
- Actions [p. 117]

  An action usually makes a modification to the installation.

The **[Add]** button shows a popup window where you can select whether to add

- an action [p. 117] , a screen [p. 105] or an application [p. 102] . Actions and screens are made available by install4j or are contributed by an installed extension [p. 47] . A registry dialog [p. 170] will be shown where you can select the desired screen or action. When adding an application, the application template dialog [p. 171] is displayed.
- an action or a screen that is contained in your custom code. New types of reusable actions or screens can be developed with the install4j API [p. 44] . In your custom code configuration [p. 168] you can specify code locations that are scanned for suitable classes. A class selector [p. 170] will be shown where you can select the desired class.
- an action group or a screen group [p. 142] . The new group is initially empty. Note that you can also create groups directly from a selection in the tree of installer elements (see below).

Installer elements can only be added to appropriate parent elements. If no appropriate parent element is selected, install4j tries to find one by moving in the ancestor hierarchy from the current selection. If no appropriate parent element can be found, an error message is displayed.

- **Applications**

  are added at the top level.
- **Screens and screen groups**

  can be added to applications or screen groups.
- **Actions and action groups**

  can be added to screens or action groups.

If you select a single installer element in the tree of installer elements, you can edit its properties on the right side. Selecting multiple installer elements is possible on the same tree level, i.e. all selected elements have to be siblings in the tree.

When the configuration area is focused, you can transfer the focus back to the tree of installer elements with the keyboard by pressing `ALT-F1`.

The tree of installer elements provides the following actions in the toolbar on the right that operate on the current selection. You can also access these actions from the context menu or use the associated keyboard shortcuts.

- **Delete**

  All selected installer elements will be deleted after a confirmation dialog when invoking the **[Delete]** action. The deleted installer elements cannot be restored.

- **Rename**

  After you add a installer element, the tree of installer elements shows it with its default name. This is often enough, however, if you have multiple instances of the same installer element alongside, a custom name makes it easier to distinguish these instances. You can assign a custom name to each installer element with the **[Rename]** action. The default name is still displayed in brackets after the custom name. To revert to the default, just enter an empty custom name in the rename dialog.

- **Comment**

  By default, installer elements have no comments associated with them. You can add comments to selected installer elements with the **[Add Comments]** action. When a comment is added, the affected installer elements will receive a "Comments" tab. After adding a comment to a single installer element, the comment area is focused automatically. Likewise, you can remove comments from one or more installer elements with the **[Remove Comments]** action.

  In order to visit all comments, you can use the **[Show next comment]** and **[Show previous comment]** actions. These actions will focus the comment area automatically and wrap around if no further comments can be found.

- **Disable**

  In order to "comment out" installer elements, you can use the **[Disable]** action. The configuration of the disabled installer elements will not be displayed, their entries in the tree of installer elements will be shown in gray and they will not be checked for errors when the project is built.

- **Copy and paste**

  install4j offers an inter-process clipboard for installer elements. You can **[Cut]** or **[Copy]** installer elements to the clipboard and **[Paste]** them in the same or a different instance of install4j. Note that references to launchers or references to files in the distribution tree might not be valid after pasting to a different project.

  Pasted installer elements are appended to the end of the same level that would be chosen if you added installer elements of that type. Sequence restrictions with respect to the the already present installer elements may force a different order.

- **Reorder**

  If your selection is a single contiguous interval, you can move the entire block up or down in the list. The selection can only be moved on the same level with the reorder actions. To move the selection to a different parent, you can cut and paste it (see above).

- **Group**

  You can create a screen group or an action group [p. 142] from the selected installer elements with the **[Create Group]** action. This action is only enabled if the selection consists of multiple screens or multiple actions. The new group will be inserted in place of the selected installer elements.

  You can dissolve a group with the **[Dissolve Group]** action. This action is only enabled if the selection consists of a single screen group or action group. The elements contained in the group will be inserted in place of the group. Nested groups will not be dissolved.

- **Link**

  You can reuse screens and actions by linking to a single definition. This is particularly useful if you define a installer maintenance application [p. 102] that should repeat parts of the installer, such as a number of forms that query the user for initial values to set up your application.

  In order to link to a screen, action, screen group or action group, you first select the installer element and invoke the **[Copy Link]** action. Then you navigate to the installer element where the link should be inserted and invoke the **[Paste Link]** action. The configuration area of a link will only contain a button that selects the original definition in the tree of installer elements.

  install4j ensures that there are no broken links in the tree of installer elements. When you delete an installer element, all links to it will be deleted as well. If that is the case, the deletion message will tell you how many links are about to be deleted.

When using the install4j API [p. ?] , you reference installer elements with IDs. You can show IDs in the tree of installer elements by activating the **[Show IDs]** button on the lower right side of the tree of installer elements. The automatically generated numerical IDs are then shown in brackets. The selection will be remembered across restarts of install4j.

In order to adjust the information density in the tree of installer elements, you can change the **icon size** by choosing large or small icons in the *Icon Size* sub-menu in the context menu. The default setting is to show large icons. The selection will be remembered across restarts of install4j.

B.5.3 Installer - Configuring Applications

Applications are configured on the screens & and actions tab [p. 99] .

The top-level nodes represent the different applications that can be configured for the project. There are 3 types of applications:

- **Installer**

  The installer is the application that is executed when the media file is invoked by the user, for example, when the user double-clicks on the installer executable in the Windows explorer. The installer cannot be deleted from the tree of installer elements.

- **Uninstaller**

  The uninstaller is a special application for uninstalling an installation. It is used in various contexts:

  - Directly invoked by the user
  - Invoked from the Windows software registry
  - Invoked by the "Uninstall previous installation" action

  The uninstaller cannot be deleted from the tree of installer elements. If you do not wish to generate an uninstaller, you can disable it [p. 99] .

- **Custom installer application**

  You can add any number of custom installer applications that can be invoked after the installation. install4j comes with several templates for auto-updaters [p. 28] . Custom applications can also be used for writing maintenance applications for your installation.

  You can add new custom installer application by clicking on the **[Add]** button on the right side of the list and choosing `Add Application` from the popup. The application templates dialog [p. 171] will be displayed and lets you choose a starting point for your custom installer application. Application templates are entirely made up of existing screens, actions and form components. You can modify the selected application template after adding it.

  Unlike the installer and uninstaller above, custom applications are also created for archive media files [p. 178] . Please see the help topic on screens and actions [p. 11] for more information on how to create first-run installers for archives.

Each installer application has a **startup sequence** of actions [p. 117] . Those actions are executed before the installer application presents a user interface. If any of these actions fails and has a "Quit on failure" failure strategy, the installer application will not be shown.

Common properties for installer applications are:

- **VM Parameters**

  If you need to pass special VM parameters to the installer application, you can enter them here. A common case would be to raise the maximum heap size with a different -Xmx parameter if your installers require a lot of memory.

- **Custom image for title bar**

  You can optionally choose a different image for the top right corner of the installer wizard. This image will be visible for all screens. The recommended size is 60x60 pixels.

- **Window width**

Here you can set the initial width of the installer application window. A width of at least 500 is recommended.

- **Window height**

  Here you can set the initial height of the installer application window. A width of at least 500 is recommended.

- **Resizeable**

  By default, the installer application window is resizeable, if you would like a fixed window size you can uncheck this option.

- **Add install4j watermark to installer screens**

  By default, a watermark with the "install4j" product name is added to the divider that separates the navigation buttons on each screen of the installer application. You can optionally disable this watermark.

- **Look and Feel init script**

  An optional script to initialize the look and feel. If empty, the system look and feel will be set. Please note that the look and feel must be available added to the custom code if it is not already part of the JRE.

- **Allow unattended mode**

  By default, installers allow unattended installations. Please see the corresponding help topic on installer modes [p. 32] for more information. All standard actions and standard screens support unattended installations. If your policy forbids unattended installations or if you include custom code that cannot handle unattended installations, you can disable them by deselecting this property.

- **Allow console installations**

  By default, installers allow console installations. Please see the corresponding help topic on installer modes [p. 32] for more information. All standard actions and standard screens support console installations, form screens [p. 14] are also fully mapped to console installers. If your policy forbids console installations or if you include custom code that cannot handle console installations, you can disable them by deselecting this property.

Special properties for the installer are:

- **Suppress initial progress dialog**

  On Windows, the installer displays a native progress dialog that informs the user that the installer is being prepared. You can suppress this dialog by deselecting this property.

- **Create log file for stderr output**

  If selected, and output on stderr is detected, an file named error.log will be created next to the installer and all output to stderr will be redirected to that file.

Special properties for the uninstaller are:

- **Executable name**

  The name of the executable for the uninstaller. Please enter a name without any path components and without a file extension. By default, this property is set to "uninstall".

- **Executable directory**

  The directory to which the executable of the uninstaller will be written. If empty, it will be placed in the .install4j directory. By default, this property is set to ".".

Special properties for custom installer applications are:

- **Executable name**

  The name of the executable for the custom installer application. Please enter a name without any path components and without a file extension.
- **Executable directory**

  The directory to which the executable of the custom installer application will be written. If empty, it will be placed in the .install4j directory.
- **Change working directory**

  If selected the working directory will be changed to the value in 'Working directory' at startup.
- **Working directory**

  The working directory to be used when 'Change working directory' is selected.
- **Vista execution level**

  The execution level for this application. If you want to modify files in the instalation direction, you most likely need administrator rights.

If you would like to associate a **custom icon** with your installer application, select the "Custom Icon" tab in the configuration of the installer application.

- In the `Cross platform` section you then have to choose icon files in the PNG image format (extension *`*.png`*) in the sizes 16x16 and 32x32 pixels. On Microsoft Windows, the generated executable will have an icon with these images, on other platforms, these image files will be used for desktop integration. It is recommended to use 32-bit images with an alpha channel, 8 bit-palette images will be generated where required. Generated Windows icons contain traditional 256 color images and 32-bit images with an alpha channel ("Windows XP icons"). However, it is also possible to use 8 bit-palette images with a transparency color for the input image files.
- If you have an **external icon file for Microsoft Windows**, you can select the `Use ICO file` option in the `Windows` section and choose an icon file (extension *`*.ico`*) in the text field below. With the `Generate from PNG files` option, the icon will be generated as described in the `Cross platform` section.
- If you have an **external icon file for Mac OS X**, you can select the `Use ICNS file` in the `Mac OS X` section and choose a Mac OS X icon file (extension *`*.icns`*) in the text field below. With the `Use standard icon` option, a standard icon provided by install4j will be used. No Mac OS X icon is generated by install4j, since the ICNS format is undocumented. Mac OS X icons have to be generated on Mac OS X with the Icon Composer application located in *`/Developer/Applications`*.

Note: If the project has already been saved, relative file paths will be interpreted as relative to the project file.

B.5.4 Installer - Configuring Screens

Screens are configured on the screens & and actions tab [p. 99] .
Please see the list of available screens [p. 106] that come with install4j.

A screen is a **single step in an installer application**. It displays information to the user or gathers user input.

If a screen has attached actions [p. 117] , there will be an expand control to the left of the screen icon that allows you to show the associated actions.

Common properties of screens are:

- **Condition expression**

  This expression is evaluated just before the screen is shown. If the expression or script returns false, the screen will be skipped.
- **Validation expression**

  This expression or script is called when the user clicks the next button. If it returns false, the current screen will be displayed again. You can use this to validate user input. Error messages are not displayed automatically, you can use the `Util.showErrorMessage(String errorMessage)` method in your script.
- **Rollback barrier**

  If the screen should be a rollback barrier. When a rollback barrier is completed, none of the preceding actions will be rolled back. You can use this property to prevent an incomplete rollback of complex changes or to protect actions from rollback when the user hits "Cancel" in the post-install phase. The installation screen is a rollback barrier by default.
- **Quit after screen**

  If the screen should have a "Finish" button instead of a "Next" button. The installer or uninstaller will quit after this screen. The "Cancel" button will not be visible if this option is checked.
- **Back button**

  Allowing the user to go back to previous screens can be problematic if the previous screen has actions attached, since by default every action is just executed once. The default behavior is the "Safe back button", where the back button is hidden if the previous screen has actions attached. If you configure your actions [p. 117] to be executed multiple times, you might want to choose the "Always visible" setting. With the "Always hidden" setting you can prevent the user from going back to the previous screen.
- **Pre-activation script**

  This script is executed just before the screen is displayed. If the screen relies on variables that are not yet initialized (for example, a form screen), you can perform the initialization here. Also you can use the API to change certain properties of the screen. Please see the examples given by the code gallery [p. 175] in the install4j IDE.
- **Post-activation script**

  This script is executed just after the screen is displayed. Here, you could display a message box if required.

Some screens only make sense when corresponding actions are used later on in the installer or uninstaller. For example, the "Services" screen will only be displayed at runtime if there are "Install a service" actions present on a subsequent screen. If such a dependency is not fulfilled after adding a screen, a corresponding notification is displayed.

B.5.5 Installer - Available Screens

Category: Customizable screens

Banner screen

A screen that has a banner on the left side and some text on white background on the right side. Banner screens are suitable for start and finish screens.

Applies to: Installer, Uninstaller

Properties:

- **Background color for banner**

  If you specify a custom banner, you might want to adjust the background color of the banner panel, the default value is suitable for the standard banner. Set to "None" in order to reset to the default value.
- **Image for banner**

  Specify a PNG or GIF image file for your custom banner. Clear to reset to the default banner.
- **Info text**

  A paragraph that explains to the user what this screen is about. This message is shown in the body of the screen.
- **Screen title**

  The title of the screen, shown in a bold and larger font. Should be a concise subject. This question is also used by the console installer for presenting the screen.

Directory selection

A screen that asks the user to select a directory. All displayed messages are configurable.

Applies to: Installer, Uninstaller

Properties:

- **Allow spaces in directory name**

  If selected, spaces are valid characters in the directory name for Unix/Linux installers, otherwise an error message is displayed if the user chooses a directory with spaces in it.
- **Directory description**

  The description of the kind of directory that use should select in a few words, e.g. "ABC directory".
- **Info text**

  A paragraph that explains to the user what this screen is about. This message is shown in the body of the screen.
- **Initial directory**

  The initially selected directory. Can be empty if no directory should be initially selected.
- **Only accept writable directories**

If selected, non-writable directories will be rejected.

- **Standard directory**

  A directory name that should be appended to the user selection in the directory browser. Should be empty if an existing directory has to be selected.

- **Screen subtitle**

  The subtitle of the screen, shown below the title in a normal font. Should be a short question. This question is also used by the console installer for presenting the screen.

- **Screen title**

  The title of the screen, shown in a bold and larger font. Should be a concise subject.

- **Variable name for selection**

  The name of the variable to which the selected directory is saved when the user advances to the next screen.

Display progress

A screen that displays a progress bar with a status line capturing the progress information of associated actions. The associated actions are executed immediately when the screen is activated. All displayed messages are configurable.

Applies to: Installer, Uninstaller

Properties:

- **Cancel enabled**

  If the cancel button should be enabled.

- **Initial status message**

  The initial status message displayed by the progress screen. You can change this message with "Set messages" actions or by invoking Context.getProgressInterface().setStatusMessage("...").

- **Screen subtitle**

  The subtitle of the screen, shown below the title in a normal font. Should be a short question. This question is also used by the console installer for presenting the screen.

- **Screen title**

  The title of the screen, shown in a bold and larger font. Should be a concise subject.

Display text

A screen that displays text to the user, either plain text or HTML. All displayed messages are configurable.

Applies to: Installer, Uninstaller

Properties:

- **Displayed text**

The text that is displayed in the screen, either plain text or HTML. For HTML, the value should start with <html>, otherwise the plain text will be displayed. The text is displayed in a scrollable text area.

- **Load displayed text from file**

  Same as the "Displayed text" property, only that the text is loaded from a file. This property is only used if the "Displayed text" property is empty. You can also specify a zip file containing files named after the ISO language code (i.e. en.txt, de.txt or en.html). The text will then be chosen automatically depending on the installer language.

- **Info text**

  A paragraph that explains to the user what this screen is about. This message is shown in the body of the screen.

- **Screen subtitle**

  The subtitle of the screen, shown below the title in a normal font. Should be a short question. This question is also used by the console installer for presenting the screen.

- **Screen title**

  The title of the screen, shown in a bold and larger font. Should be a concise subject.

Program group selection

A screen that allows the user to select a program group on Microsoft Windows. All displayed messages are configurable.

Applies to: Installer, Uninstaller

Properties:

- **Program groups for all users**

  If selected, the program groups for all users are shown, otherwise the program groups for the current user are shown.

- **Info text**

  A paragraph that explains to the user what this screen is about. This message is shown in the body of the screen.

- **Initial program group**

  The initially selected program group. Can be empty if no program group should be initially selected.

- **Screen subtitle**

  The subtitle of the screen, shown below the title in a normal font. Should be a short question. This question is also used by the console installer for presenting the screen.

- **Screen title**

  The title of the screen, shown in a bold and larger font. Should be a concise subject.

- **Variable name for selection**

  The name of the variable to which the selected program group is saved when the user advances to the next screen.

Category: Free forms

Configurable banner form

A screen where form elements can be configured along the vertical axis. Most types of information that you would like to query from a user during the installation can be easily expressed with this screen. The screen has a banner on the left side and a white background on the right side. Banner screens are suitable for start and finish screens.

Applies to: Installer, Uninstaller

Properties:

- **Background color for banner**

  If you specify a custom banner, you might want to adjust the background color of the banner panel, the default value is suitable for the standard banner. Set to "None" in order to reset to the default value.
- **Image for banner**

  Specify a PNG or GIF image file for your custom banner. Clear to reset to the default banner.
- **Fill horizontally**

  If set, the form will fill the entire horizontal extent of the screen. Otherwise, it will be centered horizontally and all form components will not be wider than their preferred widths.
- **Fill vertically**

  If set, the form will fill the entire vertical extent of the screen. Otherwise, it will be centered vertically. Note that form components always have their preferred heights when the "Scrollable" property is selected. If "Fill vertically" is selected, the form starts at the top and any remaining space is empty.
- **Info text**

  A paragraph that explains to the user what this screen is about. This message is shown in the body of the screen.
- **Scrollable**

  If set, the form will be wrapped in a scroll pane. If not set, certain form components which can grow in the vertical direction (like the text area form component) can claim remaining vertical space. Please note that those components have to be configured accordingly.
- **Screen title**

  The title of the screen, shown in a bold and larger font. Should be a concise subject. This question is also used by the console installer for presenting the screen.

Configurable form

A screen where form elements can be configured along the vertical axis. Most types of information that you would like to query from a user during the installation can be easily expressed with this screen.

Applies to: Installer, Uninstaller

Properties:

- **Fill horizontally**

  If set, the form will fill the entire horizontal extent of the screen. Otherwise, it will be centered horizontally and all form components will not be wider than their preferred widths.
- **Fill vertically**

  If set, the form will fill the entire vertical extent of the screen. Otherwise, it will be centered vertically. Note that form components always have their preferred heights when the "Scrollable" property is selected. If "Fill vertically" is selected, the form starts at the top and any remaining space is empty.
- **Scrollable**

  If set, the form will be wrapped in a scroll pane. If not set, certain form components which can grow in the vertical direction (like the text area form component) can claim remaining vertical space. Please note that those components have to be configured accordingly.
- **Screen subtitle**

  The subtitle of the screen, shown below the title in a normal font. Should be a short question. This question is also used by the console installer for presenting the screen.
- **Screen title**

  The title of the screen, shown in a bold and larger font. Should be a concise subject.

Category: Standard screens

Welcome

A screen that welcomes the user to the installation of your application. This screen should be placed at the beginning of the installation

Applies to: Installer

Properties:

- **Background color for banner**

  If you specify a custom banner, you might want to adjust the background color of the banner panel, the default value is suitable for the standard banner. Set to "None" in order to reset to the default value.
- **Image for banner**

  Specify a PNG or GIF image file for your custom banner. Clear to reset to the default banner.

Display license agreement

A screen that displays a license agreement to the user, either plain text or HTML. The license agreement must be accepted before the installation continues.

Applies to: Installer

Properties:

- **Initially accepted**

  If selected, the "Accept" radio button is initially selected.
- **Displayed license**

  The license that is displayed in the screen, either plain text or HTML. For HTML, the value should start with <html>, otherwise the plain text will be displayed. The text is displayed in a scrollable text area.
- **Load displayed license from file**

  Same as the "Displayed license" property, only that the text is loaded from a file. This property is only used if the "Displayed license" property is empty. You can also specify a zip file containing files named after the ISO language code (i.e. en.txt, de.txt or en.html). The text will then be chosen automatically depending on the installer language.
- **User must scroll to bottom**

  If selected, the user can only accept the license if the text area with the license text has been previously scrolled to the bottom. Has no effect if the "Initially selected" property is selected.

Installation location

The screen that asks the user where to install the application. This determines the principal installation directory.

Applies to: Installer

Properties:

- **Allow spaces in directory name**

  If selected, spaces are valid characters in the installation directory name for Unix/Linux installers, otherwise an error message is displayed if the user chooses a directory with spaces in it. Some JREs do no work on Unix if installed to a path that contains spaces, so spaces are disallowed by default.
- **Insufficient disk space warning**

  Show a warning message if there is not sufficent disk space for the installation on the selected target drive.
- **Existing directory warning**

  Ask the user whether to install the application in the selected directory if it already exists and the installation is not an update.
- **Show free disk space**

  Show the disk space that is available on the selected drive or partition. This setting is only effective for Windows, Mac OS X and Linux.
- **Show required disk space**

  Show the disk space that is required for the installation. You should switch this off if your installation includes other data sources.
- **Suggest application directory**

When the user chooses a directory, always append the default application directory configured in the media file wizard. You should only switch this off if you substitute a different installation directory in the screen validation.

- **Validate application id**

  Check if another application is installed in the selected directory or if the application is not the correct target for an add-on installer.

Installation type

A screen that displays a list of installation types that correspond to configurable component sets. The default types "Full","Standard" and "Customize" are provided by default. The "Installation components" screen may be hidden by this screen, depending on the installation type selected by the user. This screen will not be shown if no installation components are defined.

Applies to: Installer

Properties:

- **Bold font**

  Use a bold font for the descriptions

- **Installation types**

  Installation types are principally defined by a configurable set of components. The first installation type is selected by default in the installer.

  Each installation type has the following configurable properties:

  - A name for the installation type. This name is presented to the user
  - An optional description of the installation type. This description is displayed below the name and can be shown or hidden by the user
  - If the description is displayed by default or not
  - If the installation type is customizable or not. If the user-selected installation type is customizable, the "Installation components" screen will be shown if present, otherwise that screen will be skipped.
  - A set of installation components. Installation components are configured in the install4j IDE on the Files->Installation Components tab.You can choose between the options of installing all defined components, the default selected components as configured on the Files->Installation Components tab, or directly select a number of installation components in a check tree.

  By default, 3 universally useable installation types are added whose names and descriptions are internationalized. You can change or delete the default installation types as well as add new ones.

- **Italic font**

  Use an italic font for the descriptions

- **Smaller font**

  Use a smaller font for the descriptions

Installation components

A screen that displays all installation components and asks the user which components should be installed. This screen will not be shown if no installation components are defined.

Applies to: Installer

Properties:

- **Allow spaces in directory name**

  If selected, spaces are valid characters in the installation directory name for Unix/Linux installers, otherwise an error message is displayed if the user chooses a directory with spaces in it. Some JREs do no work on Unix if installed to a path that contains spaces, so spaces are disallowed by default.
- **Bold font**

  Use a bold font for the descriptions
- **Insufficient disk space warning**

  Show a warning message if there is not sufficent disk space for the installation on the selected target drive.
- **Existing directory warning**

  Ask the user whether to install the application in the selected directory if it already exists and the installation is not an update.
- **Italic font**

  Use an italic font for the descriptions
- **Show free disk space**

  Show the disk space that is available on the selected drive or partition. This setting is only effective for Windows, Mac OS X and Linux.
- **Show installation directory chooser**

  Show the installation directory chooser below the component selector.
- **Show required disk space**

  Show the disk space that is required for the installation. You should switch this off if your installation includes other data sources.
- **Smaller font**

  Use a smaller font for the descriptions
- **Suggest application directory**

  When the user chooses a directory, always append the default application directory configured in the media file wizard. You should only switch this off if you substitute a different installation directory in the screen validation.
- **Validate application id**

  Check if another application is installed in the selected directory or if the application is not the correct target for an add-on installer.

Create program group

A screen that allows the user to select the default program group. The "Create standard program group" action is bound to this selection, other program group actions are bound to this selection if their program group property is empty. This screen will not be shown if the "Create standard program group" is not present.

Applies to: Installer

Properties:

- **User can change "all users"**

  If the user can change the default value of the "All users" property in the "Create standard program group action". This change affects all program group actions that rely on a default program group.
- **User can disable creation**

  If the user can disable all program group actions that rely on a default program group, such as the "Create standard program group action".

File associations

A screen that displays a list of all subsequent file association actions and asks the user which associations should be made. This screen will not be shown if there are no corresponding file association actions after this screen.

Applies to: Installer

Properties:

- **Show selection buttons**

  If selected, the screen will show buttons for selecting and unselecting all file associations.

Services

A screen that allows the user to select what services should be installed. This screen will not be shown if no service executables are installed after this screen.

Applies to: Installer

Additional confirmations

A screen that displays a list of confirmations as check boxes whose results can be used in condition expressions for actions. While other types of form components can be added to this screen, only check boxes and other simple elements are consistent with the displayed text. For arbitrary forms, use the "Configurable form" screen instead.

Applies to: Installer, Uninstaller

Installation

The screen that displays displays the installation progress. Where possible, installation actions should be added to this screen.

Applies to: Installer

Properties:

- **Cancel enabled**

  If the cancel button should be enabled.

Display information

A screen that displays text to the user, either plain text or HTML. In contrast to the "Display text" screen, all messages on this screen are pre-defined and localized.

Applies to: Installer, Uninstaller

Properties:

- **Displayed text**

  The text that is displayed in the screen, either plain text or HTML. For HTML, the value should start with <html>, otherwise the plain text will be displayed. The text is displayed in a scrollable text area.
- **Load displayed text from file**

  Same as the "Displayed text" property, only that the text is loaded from a file. This property is only used if the "Displayed text" property is empty. You can also specify a zip file containing files named after the ISO language code (i.e. en.txt, de.txt or en.html). The text will then be chosen automatically depending on the installer language.

Finish

A screen that tells the user that the installation is finished. This screen should be placed at the end of the installation.

Applies to: Installer

Properties:

- **Background color for banner**

  If you specify a custom banner, you might want to adjust the background color of the banner panel, the default value is suitable for the standard banner. Set to "None" in order to reset to the default value.
- **Image for banner**

  Specify a PNG or GIF image file for your custom banner. Clear to reset to the default banner.

Uninstall Welcome

A screen that welcomes the user to the uninstallation of your application. This screen should be placed at the beginning of the uninstallation.

Applies to: Uninstaller

Properties:

- **Background color for banner**

  If you specify a custom banner, you might want to adjust the background color of the banner panel, the default value is suitable for the standard banner. Set to "None" in order to reset to the default value.
- **Image for banner**

  Specify a PNG or GIF image file for your custom banner. Clear to reset to the default banner.

Uninstallation

The screen that displays displays the uninstallation progress. Where possible, uninstallation actions should be added to this screen.

Applies to: Uninstaller

Uninstallation failure

The screen that is displayed if the uninstallation was not completed successfully. Further information regarding the uninstallation problems is displayed to the user. This screen is not shown if the uninstallation was completed successfully or if it is placed before the uninstallation screen. The uninstaller will terminate after showing this screen in case of failure.

Applies to: Uninstaller

Uninstallation success

The screen that is displayed if the uninstallation was completed successfully.

Applies to: Uninstaller

Properties:

- **Background color for banner**

  If you specify a custom banner, you might want to adjust the background color of the banner panel, the default value is suitable for the standard banner. Set to "None" in order to reset to the default value.
- **Image for banner**

  Specify a PNG or GIF image file for your custom banner. Clear to reset to the default banner.

B.5.6 Installer - Configuring Actions

Actions are configured on the screens & and actions tab [p. 99] .

Please see the list of available actions [p. 118] that come with install4j.

An action performs a **configurable unit of work** of the installer application.

Actions are attached to screens [p. 105] or they are part of the **startup sequence** that allows you to perform actions before the installer or uninstaller is displayed. If any of these actions fails and has a "Quit on failure" failure strategy, the installer application will not be shown.

Common properties of actions are:

- **Condition expression**

  This expression is evaluated just before the action is executed. If the expression or script returns false, the action will be skipped.
- **Rollback barrier**

  If the action should be a rollback barrier. When a rollback barrier is completed, none of the preceding actions will be rolled back. You can use this property to prevent an incomplete rollback of complex changes or to protect actions from rollback when the user hits "Cancel" in the post-install phase.
- **Can be executed multiple times**

  If the action can be executed multiple times. If unselected, the action will only be executed once and do nothing for subsequent invocations of the containing screen. The default settings for screens ensure that a screen with actions is only shown once. However, if the "Back button" property of a screen [p. 105] is changed of if you skip screens programatically, a screen with actions might be shown multiple times.
- **Failure strategy**

  If an action fails (i.e. returns 'false'), the installer or uninstaller can continue, quit, or ask the user what to do. If you select something other than 'Continue on failure', you should enter an error message in the "Error message" property unless the action displays the error itself.
- **Error message**

  If the action fails, this error message is displayed to the user, otherwise the action fails silently.

Most often, actions are added to the "Installation" or "Uninstallation" screens. The advantage of those screens is that they have a progress and status bar that is utilized by actions. If a screen does not expose a progress interface, the status and progress messages of attached actions are lost. This is no problem for near-instantaneous actions such as setting an environment variable, but for time-consuming operations the user should be informed about progress, even if it is only an indeterminate progress bar. As an alternative to the "Installation" or "Uninstallation" screens, you can use "Display progress" screens to create additional installation phases.

Some actions have an **"affinity" to a particular screen** and will suggest to add themselves to that screen, such as the actions in the "Final options" category which would like to go to the "Finish" screen. However, this is only a suggestion to guide you for the most common use case.

Some actions have an **associated screen** that allows the user to modify the behavior of the action. For example, the "Install a service" action has a corresponding "Services" screen that allows the user to decide whether the service should be installed and started on bootup. If such a relationship exists, a corresponding notification is displayed after adding an action.

B.5.7 Installer - Available Actions

Category: Control

Run script

Runs a custom script. The script must return a boolean value. If it returns false, the installation will be canceled.

Applies to: Installation, Uninstallation

Properties:

- **Optional Rollback Script**

  The script that will be executed in case of a rollback. The return type is void.
- **Script**

  The script that will be executed. The script must return a boolean value. If it returns false, the installation will be canceled.

Set a variable

Sets a variable by running a custom script. The script must return a String.

Applies to: Installation, Uninstallation

Properties:

- **Fail if value is null**

  If selected, the action will fail if a null value is returned from the script.
- **Only if undefined**

  The variable will only be set if it was previously undefined. This is useful for variables that your user can pass via -V or -varfile at the command line.
- **Script**

  The script that will be executed. The script must return a String.
- **Variable name**

  The name of the variable that will be set. Enter the variable without the installer prefix and the percent signs.

Set messages

Sets the messages in the progress interface.

Applies to: Installation, Uninstallation

Properties:

- **Detail message**

The detail message.

- **Status message**

  The status message.

- **Use detail**

  If the detail message should be set.

- **Use status**

  If the status message should be set.

Set the progress bar

Change the value of the progress bar or set it to indeterminate mode.

Applies to: Installation, Uninstallation

Properties:

- **Percent value**

  The progress value from 0 to 100. This property is only used when a percentage value is set or added.

- **Type of change**

  Change the progress bar either to a percentage value, add progress, set it to indeterminate mode, start a timer, or return from indeterminate mode and show the last percentage value.

- **Timer maximum value**

  The maximum progress value to be set by the timer. This property is only used when the timer is started.

- **Timer period**

  The time in milliseconds for one percent. This property is only used when the timer is started.

Sleep

Sleep a specified number of milliseconds. This is useful to ensure that a progess screen is displayed for at least a certain period of time.

Applies to: Installation, Uninstallation

Properties:

- **Sleep time**

  The sleep time in milliseconds.

Category: Desktop integration

Add a desktop link

Create a link on the desktop to an installed executable or file. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Create for all users**

  If the desktop link should be created for all users. If unselected, the link will be created for the current user only. If a "Create program group" screen is present, the "Create shortcuts for all users" check box will override this property.
- **Arguments**

  Optional arguments to the executable for Windows and Unix.
- **Tooltip description**

  An optional description for Windows that will be displayed in the tooltip.
- **Target file**

  The installed file or executable for which a link will be created on the desktop
- **Target is Single Bundle**

  If selected and the media set is a single bundle installer, the desktop icon will point to the bundle instead.
- **Name**

  The name of the desktop icon
- **Icon file**

  An optional different icon (*.ico) for the link on Windows.

Add an executable to the startup folder on Windows

Add an installed executable to the startup folder on Windows so that it will be started automatically when the user logs in. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Create for all users**

  If the startup item should be created for all users. If unselected, the link will be created for the current user only.
- **Startup executable**

  The executable that should be started when the user logs in
- **Entry name**

  The name of the entry in the startup folder

Create a file association

Create an association between a file extension and a launcher, so that the launcher is invoked when the user double-clicks a file with the selected extension. On Windows, if the application has not yet been started, the arguments to the main method will contain the file name. Subsequent invocations and all invocations on Mac OS X can be intercepted with the com.install4j.api.launcher.StartupNotification class. Only effective on Windows and Mac OS X. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Description**

  A description that is presented to the user as the text next to the corresponding checkbox in the "File associations" screen.
- **File extension**

  The file extension for which the file association should be created. Must not include the leading dot.
- **Launcher**

  The launcher that will be invoked when the file association is invoked by the user.
- **Execute on Mac OS X**

  If the file association should be performed on Mac OS X.
- **Icon file for Mac OS X**

  An optional icon file (*.icns) for the file association on Mac OS X. If empty, a default icon will be used.
- **Restart Finder**

  If true the Finder should be restarted at the end of the installation. This might be necessary for the icon (and sometimes the association itself) to be picked up immediately. Note that users might find this restart disruptive. Additionally, it you launch an application at the end of the installation it can be hidden by Finder windows.
- **Selected**

  If the file association is selected in the "File associations" screen.
- **Execute on Windows**

  If the file association should be performed on Windows.
- **Icon file for Windows**

  An optional icon file (*.ico) for the file association on Windows. If empty, a default icon will be used.

Create a quick launch icon

Create a link in the quick launch section of the Windows task bar to an installed executable or file. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Arguments**

  Optional arguments that should be passed to the executable when started with the quick launch link.
- **Description**

  The description that will be displayed in the tool tip
- **Target file**

  The installed file or executable for which a link will be created on the quick launch bar
- **Icon file**

  An optional icon file (*.ico) for the quick launch link. If empty, the default icon will be used.

Create program group

Create standard program group entries on Windows and freedesktop.org compatible UNIX desktops. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Add default launcher links**

  If generated launchers are placed into the program group automatically with their default menu untegration properties. You can rename and move the default menu integrations in the program group entries tree. If you delete them, the default menu integration can be enabled again on the "Executable info->Menu integration" step of the launcher wizard.
- **Add uninstaller**

  If the uninstaller should be added to the program group, too.
- **Create for all users**

  If the program group is created for all users or only for the current user. If a "Create program group" screen is present, the "Create shortcuts for all users" check box will override this property.
- **Application categories**

  The freedesktop.org (KDE, GNOME) application categories used to determine the best place in the applications menu. Multiple categories can be separated with a semicolon.
- **Enabled**

  If the program group creation should be performed by default. Note that if you set this property to false and the "Create program group" screen is not present, the program group will never be created.
- **Directory for links**

  The default value for the directory in which links for all relevant launchers (those with "menu integration" enabled) will be created on UNIX.
- **Program group entries**

On Windows, the entries in the program group tree will be created in the start menu by the installer.

The control buttons allow you to modify the contents of the list of program group entries. You can add new sub-folders and new file links.In the edit dialog, you have to fill in the **display name** of the program group entry.as wellas the **target file** for the of the program group link. This has to be a file or directory relative to the distribution root directory. Please note that if you select a directory as the target, it will not "fly out" in the program group, but a separate explorer window will be opened if the user clicks on it. To display all files in a directory, please add all of them as separate program group entries.

Optionally, you can specify an **icon** that is used for this program group entry. The icon file must point to an `*.ico` file. If the file name is relative, it is interpreted as relative to the project file. If you do not specify an icon, the default icon is determined by the system.

- **Program group name**

  The default value for the program group where entries for all relevant launchers (those with "menu integration" enabled) will be created. If the "Create program group" screen is present, the user can change this selection. If you leave this property empty your links will be created at the top level.

Create start menu entry

Create a start menu entry on Windows. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Create for all users**

  If the program group is created for all users or only for the current user.
- **Arguments**

  Optional arguments that should be passed to the executable when started with this entry.
- **Entry name**

  The entry name in the start menu. The name can contain sub-folders with backslashes.
- **Target file**

  The installed file or executable for which a start menu entry will be created
- **Icon file**

  An optional icon file (*.ico) for the entry. If empty, the default icon will be used.

Register Add/Remove item

Register an Add/Remove item in the Windows software registry. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Icon**
  An optional icon file (*.ico).
- **Item name**
  The name of the item that is displayed in the Windows software registry.

Category: File operations

Add Windows file rights

Adds access rights to a file or directory on Windows.

Applies to: Installation, Uninstallation

Properties:

- **All**
  All available rights.
- **Execute**
  The right to execute the object.
- **File or Directory**
  The file or directory which rights should be modified. The rights for a directory will be inherited by all subdirectories and included files.
- **Group**
  The group for which the access right should be granted.
- **Read**
  The right to read the object.
- **Write**
  The right to write to the object.

Copy a file

Copy a file. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation, Uninstallation

Properties:

- **Delay if necessary**
  If selected and the destination file cannot be replaced, the operation will be scheduled for the next reboot. The variable sys.rebootRequired will be set to Boolean.TRUE in this case.
- **Destination file**
  The destination file.

- **Access mode**

  The UNIX access mode.
- **Overwrite mode**

  How to handle an existing destination file.
- **Shared file**

  If the file should be registered as a shared file.
- **Source file**

  The file to be copied.
- **Trigger reboot if required**

  If selected and the operation is delayed until reboot, the user will be asked for a reboot automatically at the end of installation.
- **Uninstall mode**

  The mode how the uninstaller should handle the file created with this action.

Create a symbolic link

Creates a symbolic link. This action has no effect on Windows.

Applies to: Installation, Uninstallation

Properties:

- **File**

  The file or directory that the symbolic link should point to.
- **Link file**

  The link file that should be created.
- **Remove on uninstall**

  If the link should be deleted by the 'Uninstall files' action in the uninstaller.

Delete a file or directory

Deletes a file or directory. The directory can be deleted recursively.

Applies to: Installation, Uninstallation

Properties:

- **Backup for rollback**

  If checked a backup of the files to be deleted will be made and restored in case of rollback.
- **File**

  The file to be deleted
- **Recurse into directories**

  If checked and the file is a directory the action will delete all content of this directory.

Install content of a zip file

Installs the content of an external zip file to an arbitrary location. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Delay if necessary**

  If selected and a destination file cannot be replaced, the operation will be scheduled for the next reboot. The variable sys.rebootRequired will be set to Boolean.TRUE in this case.
- **Destination directory**

  The destination directory. Relative directory information in the zip file will be added to this value
- **Dir access mode**

  The UNIX access mode of installed directories.
- **File access mode**

  The UNIX access mode of installed files.
- **Overwrite mode**

  How to handle an existing destination file.
- **Shared file**

  If the file should be registered as a shared file.
- **Show progress**

  If the action should show its progress with the progress bar and the detail message.
- **Trigger reboot if required**

  If selected and an operation is delayed until reboot, the user will be asked for a reboot automatically at the end of installation.
- **Uninstall mode**

  The mode how the uninstaller should handle files created with this action.
- **Zip file**

  The zip file that contains the content to be installed.

Move a file

Moves a file. The newly created file is subject to removal by the 'Uninstall files' action.

Applies to: Installation, Uninstallation

Properties:

- **Delay if necessary**

If selected and the destination file cannot be replaced, the operation will be scheduled for the next reboot. The variable sys.rebootRequired will be set to Boolean.TRUE in this case.

- **Destination file**

  The destination file.
- **Access mode**

  The UNIX access mode.
- **Overwrite mode**

  How to handle an existing destination file.
- **Shared file**

  If the file should be registered as a shared file.
- **Source file**

  The file to be moved.
- **Trigger reboot if required**

  If selected and the operation is delayed until reboot, the user will be asked for a reboot automatically at the end of installation.
- **Uninstall mode**

  The mode how the uninstaller should handle the file created with this action.

Set the UNIX access mode of a file

Sets the UNIX access mode of a file. This action has no effect on Windows.

Applies to: Installation, Uninstallation

Properties:

- **File**

  The file or directory that the access mode should be set for.
- **Mode**

  The mode to be set. This can be any string that can be used with chmod.
- **Recursive**

  If true and file is a directory the operation will be performed recursively.

Set the modification time of a file

Sets the modification time of a file.

Applies to: Installation, Uninstallation

Properties:

- **File**

  The file or directory that the modification time should be set for.

- **Time**

  The new modification time.

Set the owner of a file

Sets the owner and optionally the group of a file. This action has no effect on Windows.

Applies to: Installation, Uninstallation

Properties:

- **File**

  The file or directory that the owner should be set for.
- **Owner**

  The owner to be set. If you want to set the group, too, please add it with a colon (example: user:group).
- **Recursive**

  If true and file is a directory the operation will be performed recursively.

Category: Final options

Create a response file

Create a response file at an arbitrary location to save user input for subsequent installations. This file can be used with the -varfile command line option.

Applies to: Installation, Uninstallation

Properties:

- **Excluded variables**

  The variable that should be excluded from the response file. If empty all variables will be used. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. Entries that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate entries, this allows you to build a variable length list of entries at runtime.
- **File**

  The file that the text should be appended to. If it doesn't exist it will be created.

Execute launcher

Execute an installed launcher and return immediately. This action is intended to be placed on the "Finish" screen. A confirmation can be added automatically to the "Finish" screen.

Applies to: Installation

Properties:

- **Launcher**
  The launcher that will be executed. Service launchers are not shown.

Reboot computer

Reboot the computer on Microsoft Windows. This action will trigger a reboot that takes place at the end of installation or uninstallation. By default, the user will be asked whether to reboot or not.

Applies to: Installation, Uninstallation

Properties:

- **Ask user**
  Ask the user whether the reboot should be performed or not.

Show URL

Show a URL in the default browser. This action is intended to be placed on the "Finish" or the "Uninstallation success" screen.

Applies to: Installation, Uninstallation

Properties:

- **URL**
  The URL that will be shown.

Show file

Show a file with the associated application. Usually, a text file or an HTML file is appropriate. This action is intended to be placed on the "Finish" screen. A confirmation can be added automatically to the "Finish" screen.

Applies to: Installation

Properties:

- **File**
  The file that will be shown.

Category: Miscellaneous

Add VM options

Adds VM options for a launcher by modifying or creating a *.vmoptions file or by changing the Info.plist file. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Launcher**

  The launcher that the VM options should be added for.
- **VM options**

  The unquoted options that should be added. Note that system property defnitions must be prefixed with -D just as on the command line, e.g. -Dkey=value. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. VM options that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate VM options, this allows you to build a variable length list of VM options at runtime.

Modify an environment variable on Windows

Sets, appends to, or prepends to an environment variable on Windows.

Applies to: Installation, Uninstallation

Properties:

- **Modification type**

  Modification type
- **User specific**

  If the variable is user specific or system wide. This has no effect on Windows 9x.
- **Value**

  The value to be set, appended or prepended.
- **Variable name**

  The name of the variable that should be modified.

Modify classpath

Changes the classpath of a launcher by modifying or creating a *.vmoptions file or by changing the Info.plist file. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Classpath entries**

The classpath entries. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. Entries that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate entries, this allows you to build a variable length list of entries at runtime.

- **Launcher**

  The launcher that the classpath should be changed for.

- **Modification type**

  Modification type

Require admin user

Requires that an admin user executes the installer otherwise fails. On Mac OS X, you can optionally restart the installer as the root user.

Applies to: Installation, Uninstallation

Properties:

- **Restart as root user**

  If selected and the user is an administrator, he will be asked for his password and the installer will be restarted with root privileges.

Run executable or batch file

Runs an executable or a Windows batch file. The action can optionally wait for termination of the executable.

Applies to: Installation, Uninstallation

Properties:

- **Arguments**

  The arguments passed to the executable. Please note that in the property sheet, arguments have to be separated by semicolons (;) and in the edit dialog each argument starts on a new line. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. Arguments that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate arguments, this allows you to build a variable length list of arguments at runtime.

- **Executable**

  The file that should be executed. Please do not add arguments here, there is a separate "Arguments" property.

- **Fail for redirection errors**

  If selected, the action fails if the redirection files cannot be accessed. Otherwise, those errors are silently ignored.

- **Log arguments**

If the arguments should be written into the log file or not. Disabled by default due to security reasons.

- **Redirection file for stderr**

  A file to which the stderr output of the executed process is saved. If empty, the stderr output will be discarded.

- **Redirection file for stdin**

  A file which should be fed to the input stream of the executed process. If empty, the stdin input will be empty.

- **Redirection file for stdout**

  A file to which the stdout output of the executed process is saved. If empty, the stdout output will be discarded.

- **Wait for termination**

  If the action should wait for termination of the process and check if the return value is 0.

- **Working directory**

  The working directory for the execution.

Category: Registry modifications

Add access rights for a key in the Windows registry

Add access rights for a key in the Windows registry.

Applies to: Installation, Uninstallation

Properties:

- **All**

  All available rights.

- **Execute**

  The right to execute the object.

- **Group**

  The group for which the access right should be granted.

- **Key name**

  The name of the registry key without a leading backslash.

- **Read**

  The right to read the object.

- **Registry root**

  The Windows registry root where the key is located.

- **Write**

  The right to write to the object.

Delete a key or value in the Windows registry

Delete a key or value in the Windows registry.

Applies to: Installation, Uninstallation

Properties:

- **Key name**

  The name of the registry key that should be deleted or contains the value to be deleted without a leading backslash.
- **Only if empty**

  If a key should only be deleted when it contains no sub-keys or values.
- **Registry root**

  The Windows registry root where the key or value should be deleted.
- **Value name**

  The name of the registry value that should be deleted. If you leave this empty, the key will be deleted instead.

Delete a node or key in the Java preference store

Delete an entire package node or a key-value pair in the Java preference store.

Applies to: Installation, Uninstallation

Minimum Java requirement: 1.4

Properties:

- **Key**

  The key that should be deleted. If you leave this empty, the entire package node will be deleted instead. The action does not return an error if this key does not exist.
- **Only if empty**

  If a node should only be deleted when it contains no sub-nodes or keys.
- **Package name**

  The name of the package node in the preference store that should be deleted or contains the key-value pair to be deleted. The action does not return an error if this package node does not exist.
- **Preference root**

  If you want to delete the node or key-value pair for the current user, all users, or both.

Set a key in the Java preference store

Set a key-value pair in the Java preference store. The package node is created if necessary. This is the most convenient way to communicate settings to your launchers.

Applies to: Installation, Uninstallation

Minimum Java requirement: 1.4

Properties:

- **Key**

  The key for which a value should be set.
- **Package name**

  The name of the package node in the preference store where the key-value pair should be set.
- **Preference root**

  If you want to set the key for the current user only or for all users. Due to access rights it can happen that the system preference registry is not writable, in that case a fallback to the user specific registry can be tried.
- **Value**

  The string value that should be set for the key.

Set a value in the Windows registry

Set a value in the Windows registry. This action can also create the appropriate key if necessary.

Applies to: Installation, Uninstallation

Properties:

- **Create key**

  If set the key will be created if it doesn't exist.
- **Key name**

  The name of the registry key that contains the value or that should be created without a leading backslash.
- **Registry root**

  The Windows registry root where the key should be created
- **Value**

  The value that should be written into the registry.
- **Value name**

  The name of the registry value.

Category: Services

Install a service

Installs a service. On Windows, this is done by executing the service launcher with the appropriate arguments. On Unix, a link will be placed in /etc/init.d. On Mac OS, a StartupItem will be created. This action will be automatically reverted by the 'Uninstall files' action.

Applies to: Installation

Properties:

- **Add to "Services" screen**
  Ask the user in the "Services" screen whether this action should be performed or not.
- **Allow user to change start type**
  allow user to change the start type on the "Services screen"
- **Service**
  The service launcher that will be invoked.
- **Initially selected**
  Initially selected for installation on the "Services screen". Note that if this property is set to "false" and the "Services screen" is not present, this action will never be performed.

Start a service

Starts a service by executing the service launcher with the appropriate arguments.

Applies to: Installation

Properties:

- **Autostart only**
  If selected the service will only be started when it is installed as autostart service.
- **Service**
  The service launcher that will be started.

Stop a service

Stops a service by executing the service launcher with the appropriate arguments.

Applies to: Installation, Uninstallation

Properties:

- **Service**
  The service launcher that will be stopped.

Category: Text files

Append text to a file

Append text to a file or write text to a new file.

Applies to: Installation, Uninstallation

Properties:

- **Encoding**

  The encoding of the file. If you leave this empty the system default will be used.
- **Escaped text**

  If selected escape sequences like \n,\t or \u1234 in the text property will be replaced.
- **File**

  The file that the text should be appended to. If it doesn't exist it will be created.
- **Text**

  The text that should be appended.

Fix line feeds

Changes the line feeds of a text file to the platform specific type.

Applies to: Installation, Uninstallation

Properties:

- **File**

  The file or directory that the line feeds should be fixed for.
- **Recursive**

  If true and file is a directory the operation will be performed recursively. If false and file is a directory the files in this directory will be fixed, but no subdirectories will be entered.
- **Suffixes**

  The suffixes with a leading dot of the files to be fixed if the "File" property is a directory. If empty, all files will be used. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. Suffixes that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate suffixes, this allows you to build a variable length list of suffixes at runtime.

Modify a text file

Modify an installed text file by replacing a search value in the file.

Applies to: Installation, Uninstallation

Properties:

- **Encoding**

  The encoding of the file. If you leave this empty the system default will be used.
- **Escape for property file**

If set, the replaced value will be escaped for use in a Java property file.

- **Text file**

  The text file that should be modified
- **Replace value**

  The value that should be used instead
- **Search value**

  The value that should be searched

Modify a text file with regular expressions

Modify an installed text file by applying a regular expression.

Applies to: Installation, Uninstallation

Minimum Java requirement: 1.4

Properties:

- **Encoding**

  The encoding of the file. If you leave this empty the system default will be used.
- **Escape for property file**

  If set, the replaced value will be escaped for use in a Java property file.
- **Text file**

  The text file that should be modified.
- **Quote variables**

  If values of installer variables in the match and replacement expressions should be quoted. This means that the characters of replaced installer variables will be treated literally instead of modifying the search or replace expressions with special charaters such as \ or $.
- **Match expression**

  The match expression.
- **Replace all**

  If all occurrences should be replaced or only the first one.
- **Replacement**

  The replacement.

Category: Update

Check for update

Load the update descriptor from the a URL and save it to the a variable. If sucessful, the variable will contain an instance of com.install4j.api.UpdateDescriptor

Applies to: Installation, Uninstallation

Properties:

- **Show error message**

  Show a default error message if the download fails.
- **Update descriptor URL**

  The URL from which the update descriptor for this project can be downloaded. The update descriptor file is automatically created when compiling the project and can be found in the media output directory. The URL must start with http://.
- **Variable**

  The installer variable to which an instance of class com.install4j.api.UpdateDescriptor will be saved if the action is sucessful.

Download file

Download a URL and save it to a file

Applies to: Installation, Uninstallation

Properties:

- **Delete downloaded file on exit**

  If selected, the downloaded file will be deleted when the installer application terminates.
- **Show error message**

  Show a default error message if the download fails.
- **Show file name**

  If selected, the name of the downloaded file and the target directory will be displayed. This setting has no effect if "Show progress" is not selected.
- **Show progress**

  If the action should show its progress with the progress bar.
- **Target file**

  The file to which the downloaded URL will be saved.
- **URL**

  The URL from which the file should be downloaded. The URL must start with http://.

Shut down calling launcher

Shut down the launcher that called this application if it was started with the com.install4j.api.launcher.ApplicationLauncher API.

Applies to: Installation, Uninstallation

Properties:

- **Timeout**

The timeout in seconds this action will wait if the 'Wait' property is true. If set to 0 there will be no timeout.

- **Wait**

  If selected the action will wait for the calling launcher to exit.

Category: XML files

Apply an XSLT transform

Transform an installed file by applying an XSLT stylesheet.

Applies to: Installation, Uninstallation

Minimum Java requirement: 1.4

Properties:

- **Download external entities**

  If selected, a DTD referenced with an HTTP system ID will be downloaded as the document is parsed. The success of the action requires a direct internet connection in that case.
- **Destination file**

  The output of the transformation. This can be the same file as the source.
- **Source file**

  The source for the transformation. This can be the same file as the destination.
- **Validate XML file**

  If selected, the XML parser will validate the document according to a associated DTD or XML schema. If the validation is unsuccessful, the action will fail.
- **Stylesheet**

  The XSLT stylesheet to apply.

Replace text in XML files

Modify an installed XML file by selecting nodes with an XPath expression and applying a regular expression on the selected values.

Applies to: Installation, Uninstallation

Minimum Java requirement: 1.4

Properties:

- **Download external entities**

  If selected, a DTD referenced with an HTTP system ID will be downloaded as the document is parsed. The success of the action requires a direct internet connection in that case.
- **XML file**

The XML file that should be modified.

- **Quote variables**

  If values of installer variables in the match and replacement expressions should be quoted. This means that the characters of replaced installer variables will be treated literally instead of modifying the search or replace expressions with special charaters such as \ or $.

- **Match expression**

  The match expression.

- **Replace all**

  If all occurrences should be replaced or only the first one.

- **Replacement**

  The replacement.

- **Validate XML file**

  If selected, the XML parser will validate the document according to a associated DTD or XML schema. If the validation is unsuccessful, the action will fail.

- **XPath expression**

  The XPath expression to selected DOM nodes that have a value (this includes attributes and text). Example for replacing text in an attribute: '/myRootNode/myChildNode/@myAttribute'. Example for replacing text in an element: '/myRootNode/myChildNode/text()'

Install files

Install all files in the distribution tree that are contained in the selected installation components.

Applies to: Installation

Properties:

- **Insufficient disk space warning**

  Show a warning message if there is not sufficent disk space for the installation on the selected target drive.

- **Delay if necessary**

  If selected and a destination file cannot be replaced, the operation will be scheduled for the next reboot. The variable sys.rebootRequired will be set to Boolean.TRUE in this case.

- **Directory resolver**

  Expression or script that resolves the actual installation directory separately for each installed file. Return null, if you would like to choose the standard installation directory for a file.

- **File filter**

  Expression or script that is invoked for each file to decide whether to install the file or not.

- **Install runtime**

  Create the installation directory and install the install4j runtime. If your installer just modifies some folders and does not need launchers, an uninstaller or custom installer applications,

you can deselect this option and use other installation roots in the distribution tree to install files.

- **Show file names**

  If set, the names of the files that are installed will be shown during the installation.

- **Installation size calculator**

  Expression or script that calculates a custom installation size in bytes. The default size in bytes is passed as a parameter.

- **Trigger reboot if required**

  If selected and an operation is delayed until reboot, the user will be asked for a reboot automatically at the end of installation.

- **Validate application id**

  Check if another application is installed in the selected directory or if the application is not the correct target for an add-on installer. If you have an "installation location" screen, you don't have to select this option.

Uninstall files

Uninstall all installed files.

Applies to: Uninstallation

Uninstall previous installation

Uninstalls the previous installation of this application in the selected installation directory.

Applies to: Installation

B.5.8 Installer - Screens and Actions Groups

Screen and action groups can be configured on the screens & and actions tab [p. 99] .

Actions and screens can be grouped in the tree of installer elements. Groups of the same type can be nested, meaning that you can put a screen group into a screen group or an action group into an action group.

You can nest as many levels of groups as you wish. Next to the label of the screen or action group in the tree of installer elements, the number of all contained screens or actions is shown in bold font. Elements in nested groups are counted as well.

Grouping has the following common benefits for screens and actions:

- **Organization**

  If you have many screens or actions, groups emphasize which elements belong together. You can add a common comment to the group.
- **Common condition**

  Groups have a "Condition expression" property that allows you to skip the group with a common condition instead of having to repeat the condition for each contained element.
- **Single link target**

  If you want to reuse a set of adjacent screens or actions in a different part of your project, you can put them in a group and add a single link to that group instead of linking to each element separately.
- **Looping**

  A group has a "Loop expression" property that allows you to execute the group repeatedly until the loop expression returns `false`.

The following benefits are exclusive to screen groups:

- **Jump targets**

  When you jump to a screen programmatically (with `context.gotoScreen(...)`), it is more maintainable to jump to a group instead of to a single screen. You can think of the group as taking the function of a label in this case.

The following benefits are exclusive to action groups:

- **Break group on error**

  Action groups have a boolean "On error break group" property. If selected, and one of the contained actions returns with an error, the control flow will step out of the action group and continue with the next element after the group. This behavior only takes effect if the problematic action has its failure strategy set to "Continue on failure".
- **Default error message**

  You can define a "Default error message" for the group that is used for all actions that do not have their own error message configured.

B.5.9 Installer - Configuring Form Components

For more information on form screens and related concepts, please see the corresponding help topic [p. 14] .

Please see the list of available form components [p. 145] that come with install4j.

If you select a single form component in the list of form components, you can edit its properties on the right side.

When the configuration area is focused, you can transfer the focus back to the list of form components with the keyboard by pressing `ALT-F1`.

The list of form components provides the following actions in the toolbar on the right that operate on the current selection. You can also access these actions from the context menu or use the associated keyboard shortcuts.

- **Delete**

  All selected form components will be deleted after a confirmation dialog when invoking the **[Delete]** action. The deleted form components cannot be restored.

- **Rename**

  After you add a form component, the list of form components shows it with its default name. This is often enough, however, if you have multiple instances of the same form component alongside, a custom name makes it easier to distinguish these instances. You can assign a custom name to each form component with the **[Rename]** action. The default name is still displayed in brackets after the custom name. To revert to the default, just enter an empty custom name in the rename dialog.

- **Comment**

  By default, form components have no comments associated with them. You can add comments to selected form components with the **[Add Comments]** action. When a comment is added, the affected form components will receive a "Comments" tab. After adding a comment to a single form component, the comment area is focused automatically. Likewise, you can remove comments from one or more form components with the **[Remove Comments]** action.

  In order to visit all comments, you can use the **[Show next comment]** and **[Show previous comment]** actions. These actions will focus the comment area automatically and wrap around if no further comments can be found.

- **Disable**

  In order to "comment out" form components, you can use the **[Disable]** action. The configuration of the disabled form components will not be displayed, their entries in the list of form components will be shown in gray and they will not be checked for errors when the project is built.

- **Copy and paste**

  install4j offers an inter-process clipboard for form components. You can **[Cut]** or **[Copy]** form components to the clipboard and **[Paste]** them in the same or a different instance of install4j. Note that references to launchers or references to files in the distribution tree might not be valid after pasting to a different project.

  Pasted form components are appended to the end of the list of form components.

- **Reorder**

  If your selection is a single contiguous interval, you can move the entire block up or down in the list.

Common properties of form components are:

- **Initialization script**

  A script that initializes the form component. To configure the contained principal component, such as a JCheckBox, use the configurationObject parameter (if available). This script will run after the internal initialization of the form component, just before the component appears on the screen.

- **Reset initialization on previous**

  If set, the component will be initialized each time the user enters in the forward direction. Otherwise, the initialization will be performed only once. This setting affects both the internal initialization as well as the initialization script.

- **Insets**

  The "Top", "Left", "Bottom" and "Right" properties allow you to specify insets around the form component.

You can preview a form screen with the **[Preview]** button which is also available on the property page of a screen. The preview **does not show the actual screen**, it shows an installer window with typical elements and a form that fills the entire content area of the screen. The actual screen might have a different visual appearance and the form might be smaller. However, the layout of the form itself will be the same at runtime.

B.5.10 Installer - Available Form Components

Category: Action components

Button

A standard button with an optional leading label. When the user clicks on the button, an action script is executed.

Properties:

- **Action script**

  The script that is executed when the button is clicked by the user. The return type is void.
- **Button icon**

  The icon displayed on the button. Can be empty if the button text is set.
- **Button text**

  The text that is displayed on the button. Can be empty if the button icon is set.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.

Hyperlink URL label

A label that displays a hyperlink. When the user clicks on the hyperlink, the appropriate action is performed, depending on the protocol of the URL.

Properties:

- **Hyperlink text**

  The text that is displayed on the hyperlink label.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **URL**

  The URL for the hyperlink. For example http://www.ej-technologies.com

Hyperlink action label

A label that displays a hyperlink. When the user clicks on the hyperlink, an action script is executed

Properties:

- **Action script**

  The script that is executed when the hyperlink is clicked by the user. The return type is void.
- **Hyperlink text**

  The text that is displayed on the hyperlink label.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.

Category: Labels and spacers

Horizontal separator

A horizontal separator with an optional label.

Properties:

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Font color**

The color of the label font. If empty, the default color will be used.

- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the label. Can be empty.

- **Text**

  The text of the label. Can be empty.

- **Use special title font**

  If selected, a special font and color are used for the separator label. The actual font depends on the look and feel. This setting overrides all other font settings for the label.

Key value pair label

A pair of labels. The first ('key') label aligns with other leading labels on the form, the second ('value') label consumes the remaining horizontal space,

Properties:

- **Icon-text gap**

  The gap between the key label icon and the key label text in pixels.

- **Font color**

  The color of the key label font. If empty, the default color will be used.

- **Font**

  The font of the key label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the key label. Can be empty.

- **Text**

  The text of the key label. Can be empty.

- **Icon-text gap**

  The gap between the value label icon and the value label text in pixels.

- **Font color**

  The color of the value label font. If empty, the default color will be used.

- **Font**

  The font of the value label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the value label. Can be empty.

- **Text**

  The text of the value label. Can be empty.

Label

A single label.

Properties:

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.

Multi-line HTML label

A multi-line label that wraps text as needed and displays simple HTML. In particular you can include HTML links that open a browser.

Properties:

- **HTML**

  The HTML for the label. The value should start with <html>, otherwise the plain text will be displayed. You can include HTML links that open a browser when clicked by the user. URLs in <a href="..."> links should start with http:// or file://.

Multi-line label

A multi-line label that wraps text as needed.

Properties:

- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Text**

  The text of the label.

Vertical spacer

An invisible vertical spacer of configurable height.

Properties:

- **Spacer height**

  The height of the spacer in pixels. The spacer itself is invisible.

Category: Option selectors

Check box

A check box with an optional leading label. The user selection (Boolean.TRUE or Boolean.FALSE) is saved to a variable.

Properties:

- **Text**

  The text of the check box. Can be empty.
- **Coupled components**

  You can select other components on the same form screen which are enabled only if the check box is selected.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initially selected**

  If set, the check box is initially selected
- **Inverse coupling**

  If set, the coupling of other form components will be inverted with respect to the selection state of the check box.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Selection script**

  The script that is executed when the selection state of the check box is changed by the user. The return type is void.
- **Variable name**

The name of the variable to which the user input is assigned. The variable value will be one of java.lang.Boolean.TRUE or java.lang.Boolean.FALSE, depending on the user selection.

Combo box

A combo box with an optional leading label. The user can enter arbitrary text into the combo box. The user selection (the selected item as a string) is saved to a variable.

Properties:

- **Fill horizontal space**

  If set, the combo box will fill all the available horizontal space, otherwise it will be as wide as required for the the widest item.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initially selected index**

  The zero-based index of the initially selected item in the combo box.
- **Input validation expression**

  An expression or script that validates the user input when the combo box loses the focus. If the expression returns false, the focus remains in the combo box. In that case you should display an error message.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Combo box entries**

  The items in the combo box as a list separated by semicolons. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. Items that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate items, this allows you to build a variable length list of items at runtime.
- **Selection change script**

  A script that is executed when the selection is changed by the user. This script is only required for advanced customizations of the form screen. The return type is void.
- **Variable name**

  The name of the variable to which the user input is assigned. The variable value will be the selected item string.

Drop-down list

A drop-down list with an optional leading label. The user selection (the selected index as a java.lang.Integer) is saved to a variable.

Properties:

- **Fill horizontal space**

  If set, the drop-down list will fill all the available horizontal space, otherwise it will be as wide as required for the the widest item.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initially selected index**

  The zero-based index of the initially selected item in the drop-down list.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Drop-down list entries**

  The items in the drop-down list as a list separated by semicolons. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. Items that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate items, this allows you to build a variable length list of items at runtime.
- **Selection change script**

  A script that is executed when the selection is changed by the user. This script is only required for advanced customizations of the form screen. The return type is void.
- **Variable name**

  The name of the variable to which the user input is assigned. The variable value will be the index of the selected item as a java.lang.Integer.

List

A list with an optional leading label. The user selection (the selected indices) is saved to a variable.

Properties:

- **Fill horizontal space**

If set, the list will fill all the available horizontal space, otherwise it will be as wide as required for the the widest item.

- **Fill extra vertical space**

  If set, the component will expand to fill remaining vertical space. Extra vertical space is only available, if the form is not scrollable. Custom form screens have a "Scrollable" property, which must be set to false.

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.

- **Initially selected index**

  The zero-based index of the initially selected item in the list.

- **Font color**

  The color of the label font. If empty, the default color will be used.

- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the label. Can be empty.

- **Text**

  The text of the label. Can be empty.

- **List entries**

  The items in the list as a list separated by semicolons. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. Items that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate items, this allows you to build a variable length list of items at runtime.

- **Multi-selection**

  If set, the user can select multiple entries at the same time.

- **Scrollable**

  If set, the list will be wrapped in a scroll pane.

- **Selection change script**

  A script that is executed when the selection is changed by the user. This script is only required for advanced customizations of the form screen. The return type is void.

- **Variable name**

  The name of the variable to which the user input is assigned. If multiple items can be selected, the variable value will be an int array with the selected indices, otherwise the variable value will be the index of the selected item as a java.lang.Integer

- **Visible rows**

  If the list is scrollable, this property determines the height of the list.

Radio button group

A number of radio buttons in a common button group with an optional leading label. The user selection (the selected index as a java.lang.Integer) is saved to a variable.

Properties:

- **Axis**

  The direction along which the radio buttons will be laid out.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initially selected index**

  The zero-based index of the initially selected radio button.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Radio button labels**

  The labels of all the radio buttons as a list separated by semicolons.
- **Variable name**

  The name of the variable to which the user input is assigned. The variable value will be the index of the selected radio button as a java.lang.Integer.

Single radio button

A single radio button with an optional leading label. If selected, a specified string is saved to a variable. If you place multiple instances of this form component on a form screen and give them the same variable name, they will form a radio button group.

Properties:

- **Coupled components**

  You can select other components on the same form screen which are enabled only if the radio button is selected.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initially selected**

  If selected, the radio button will be initially selected.
- **Inverse coupling**

If set, the coupling of other form components will be inverted with respect to the selection state of the radio button.

- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Radio button label**

  The text of the radio button.
- **Variable name**

  The name of the variable to which the user input is assigned. The variable value will be the string defined below.
- **Variable value**

  The value that should be witten to the variable if this radio button is selected.

Category: Sliders and spinners

Slider

A slider with an optional leading label. The user input (a java.lang.Integer) is saved to a variable.

Properties:

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial value**

  The initial value of the slider.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Major tick spacing**

The spacing between major ticks expressed as a value.

- **Maximum value**

  The maximum value on the right side of the slider.

- **Minimum value**

  The minimum value on the left side of the slider.

- **Minor tick spacing**

  The spacing between minor ticks expressed as a value.

- **Snap to ticks**

  If set, the user selection is snapped to the closest tick value.

- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.lang.Integer.

Spinner of dates

A spinner with date and time values with an optional leading label. The user input is saved to a variable.

Minimum Java requirement: 1.4

Properties:

- **Date format pattern**

  A pattern to format dates for display and input as described in the javadoc of java.text.SimpleDateFormat. An example is "yyyy.MM.dd 'at' HH:mm:ss z". If empty, a locale-dependent default pattern with date and time components will be used.

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.

- **Initial value**

  The initial value of the spinner.

- **Font color**

  The color of the label font. If empty, the default color will be used.

- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the label. Can be empty.

- **Text**

  The text of the label. Can be empty.

- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.util.Date.

Spinner of enumerated values

A spinner with enumerated values with an optional leading label. The user input is saved to a variable.

Minimum Java requirement: 1.4

Properties:

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initially selected index**

  The zero-based index of the initially selected item in the spinner.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **List entries**

  The items in the spinner. The items in the list must be separated by semicolons. If you click on the edit button in the property editor, you can enter one item per line in a separate dialog. List items that are installer variables with array values (e.g. String[] or Object[]) are expanded as separate list items, this allows you to build a variable length list of list items at runtime.
- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.lang.String.

Spinner of integer values

A spinner with integer values with an optional leading label. The user input is saved to a variable.

Minimum Java requirement: 1.4

Properties:

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial value**

  The initial value of the spinner.
- **Font color**

The color of the label font. If empty, the default color will be used.

- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the label. Can be empty.

- **Text**

  The text of the label. Can be empty.

- **Maximum value**

  The maximum value on the right side of the spinner.

- **Minimum value**

  The minimum value on the left side of the spinner.

- **Step size**

  The step size for the spinner.

- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.lang.Integer.

Category: Special selectors and displays

Directory chooser

A directory chooser with an optional leading label. The user selection is saved to a variable.

Properties:

- **Allow spaces in directory name**

  If selected, spaces are valid characters in the selected directory name for Unix/Linux installers, otherwise an error message is displayed if the user chooses a directory with spaces in it.

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.

- **Initial directory**

  The initially selected directory. Can be empty.

- **Font color**

  The color of the label font. If empty, the default color will be used.

- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the label. Can be empty.

- **Text**

The text of the label. Can be empty.

- **Manual entry allowed**

  If selected, the user can enter the directory manually in the text field. Otherwise, the text field is disabled.

- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.lang.String.

File chooser

A file chooser with an optional leading label. The user selection is saved to a variable.

Properties:

- **File filter name**

  The name of the file filter. If empty, all files are selectable.

- **Filtered file extension**

  The list of the filtered file extension. If empty, all files are selectable.

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.

- **Initial file**

  The initially selected file. Can be empty.

- **Font color**

  The color of the label font. If empty, the default color will be used.

- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Icon**

  An image file with an icon for the label. Can be empty.

- **Text**

  The text of the label. Can be empty.

- **Manual entry allowed**

  If selected, the user can enter the file manually in the text field. Otherwise, the text field is disabled.

- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.lang.String.

Installation directory chooser

An installation directory chooser with an optional display of required and free space. The user selection is set as the installatoin directory.

Properties:

- **Allow spaces in directory name**

  If selected, spaces are valid characters in the installation directory name for Unix/Linux installers, otherwise an error message is displayed if the user chooses a directory with spaces in it. Some JREs do no work on Unix if installed to a path that contains spaces, so spaces are disallowed by default.
- **Insufficient disk space warning**

  Show a warning message if there is not sufficent disk space for the installation on the selected target drive.
- **Existing directory warning**

  Ask the user whether to install the application in the selected directory if it already exists and the installation is not an update.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Show free disk space**

  Show the disk space that is available on the selected drive or partition. This setting is only effective for Windows, Mac OS X and Linux.
- **Show required disk space**

  Show the disk space that is required for the installation. You should switch this off if your installation includes other data sources.
- **Suggest application directory**

  When the user chooses a directory, always append the default application directory configured in the media file wizard. You should only switch this off if you substitute a different installation directory in the screen validation.
- **Validate application id**

  Check if another application is installed in the selected directory or if the application is not the correct target for an add-on installer.

Progress display

An progress display that can show the progress of the actions attached to the containing screen.

Properties:

- **Detail line visible**

  If selected, the detail line is visible.
- **Hide initially**

  If selected, the progress bar is hidden when the form is shown. When the actions attached to the screen are executed, the progrss bar is made visible.
- **Status line visible**

  If selected, the status line is visible.

Update schedule selector

Drop-down box that lets the user select an update schedule for your application. You can use the com.install4j.api.update.UpdateScheduleRegistry class in your application to check if you should launch an updater. Please see the Javadoc for more information. Please note that simply adding this form component does not automatically launch an updater at regular intervals.

Properties:

- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial update schedule**

  The initially selected update schedule. If the user has already installed the application before, the currently active selection by the user will be selected and this value will be ignored.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.

Category: Text fields

Password field

A password text field with an optional leading label. The user input is displayed with '*' characters. The user input is saved to a variable.

Properties:

- **Text field columns**

  The width of the text field, expressed as a multiple of the width of the character 'm'. If zero, the text field will fill the entire horizontal space.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial text**

  The initial text in the text field. Can be empty.
- **Input validation expression**

  An expression or script that validates the user input when the password field loses the focus. If the expression returns false, the focus remains in the password field. In that case you should display an error message.
- **Key validation expression**

  An expression or script that validates each key that is pressed in this component by the user.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Font**

  The font of the text field. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Variable name**

  The name of the variable to which the user input is assigned.
- **Write encoded value to response file**

  Write an encoded value of the entered password to the response file. Note that the encoding only prevents casual observation of the password. Do not enable if you require strict security for the password.

Text area

A text area with an optional leading label. The user input is saved to a variable.

Properties:

- **Text area columns**

  The width of the text area, expressed as a multiple of the width of the character 'm'. If zero, the text area will fill the entire horizontal space.
- **Fill extra vertical space**

  If set, the component will expand to fill remaining vertical space. Extra vertical space is only available, if the form is not scrollable. Custom form screens have a "Scrollable" property, which must be set to false.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial text**

  The initial text in the text field. Can be empty.
- **Input validation expression**

  An expression or script that validates the user input when the text area loses the focus. If the expression returns false, the focus remains in the text area. In that case you should display an error message.
- **Key validation expression**

  An expression or script that validates each key that is pressed in this component by the user.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Wrap lines**

  If set, lines will wrap at the end of the text area. Otherwise a horizontal scroll bar will be show when needed.
- **Text area rows**

  The height of the text area, expressed as a multiple of line heights. Must be greater than zero.
- **Font**

  The font of the text field. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Use label font**

  If set, the default label font is used and other font settings are ignored.
- **Variable name**

  The name of the variable to which the user input is assigned.
- **Wrap entire words**

This property is only relevant when lines are wrapped. If set, lines will wrap on word boundaries if possible.

Text field

A text field with an optional leading label. The user input is saved to a variable.

Properties:

- **Text field columns**

  The width of the text field, expressed as a multiple of the width of the character 'm'. If zero, the text field will fill the entire horizontal space.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial text**

  The initial text in the text field. Can be empty.
- **Input validation expression**

  An expression or script that validates the user input when the text field loses the focus. If the expression returns false, the focus remains in the text field. In that case you should display an error message.
- **Key validation expression**

  An expression or script that validates each key that is pressed in this component by the user.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Font**

  The font of the text field. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Variable name**

  The name of the variable to which the user input is assigned.

Text field with date format

A text field with an optional leading label and a date format. The user input (a java.util.Date) is saved to a variable.

Minimum Java requirement: 1.4

Properties:

- **Text field columns**

  The width of the text field, expressed as a multiple of the width of the character 'm'. If zero, the text field will fill the entire horizontal space.
- **Date display style**

  The date display style specifies the verbosity of the date component.
- **Date format**

  The date format specifies the components of the date that should be editable, i.e. date, time or date and time.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial value**

  The initial date in the text field.
- **Input validation expression**

  An expression or script that validates the user input when the text field loses the focus. If the expression returns false, the focus remains in the text field. In that case you should display an error message.
- **Key validation expression**

  An expression or script that validates each key that is pressed in this component by the user.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Font**

  The font of the text field. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Time display style**

  The time display style specifies the verbosity of the time component.
- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.util.Date.

Text field with format mask

A text field with an optional leading label and an arbitrary format mask. The user input is saved to a variable. The default mask is that of an SSN. For more information, please see the javadoc of javax.swing.text.MaskFormatter.

Minimum Java requirement: 1.4

Properties:

- **Allow invalid input**

  If set, invalid input will be allowed during editing. When the text field loses focus, invalid input will not be accepted, so the final value is guaranteed to be valid in any case.
- **Text field columns**

  The width of the text field, expressed as a multiple of the width of the character 'm'. If zero, the text field will fill the entire horizontal space.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial text**

  The initial text in the text field. Can be empty.
- **Input mask**

  The input mask as defined be the javadoc of javax.swing.text.MaskFormatter.
- **Input validation expression**

  An expression or script that validates the user input when the text field loses the focus. If the expression returns false, the focus remains in the text field. In that case you should display an error message.
- **Invalid characters**

  If not empty, this string defines the characters that are invalid for user input.
- **Key validation expression**

  An expression or script that validates each key that is pressed in this component by the user.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

  The text of the label. Can be empty.
- **Placeholder character**

  The character that is displayed for empty characters of the input mask that still have to be filled out by the user.
- **Font**

  The font of the text field. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Valid characters**

  If not empty, this string defines the characters that are valid for user input.
- **Return literal characters**

  If set, the value that is saved to the variable contains literal characters defined in the input mask.
- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.lang.String.

Text field with integer format

A text field with an optional leading label and an integer format. The user input is saved to a variable with type java.lang.Long.

Minimum Java requirement: 1.4

Properties:

- **Allow invalid input**

  If set, invalid input will be allowed during editing. When the text field loses focus, invalid input will not be accepted, so the final value is guaranteed to be valid in any case.
- **Text field columns**

  The width of the text field, expressed as a multiple of the width of the character 'm'. If zero, the text field will fill the entire horizontal space.
- **Icon-text gap**

  The gap between the label icon and the label text in pixels.
- **Initial text**

  The initial text in the text field. Can be empty.
- **Input validation expression**

  An expression or script that validates the user input when the text field loses the focus. If the expression returns false, the focus remains in the text field. In that case you should display an error message.
- **Key validation expression**

  An expression or script that validates each key that is pressed in this component by the user.
- **Font color**

  The color of the label font. If empty, the default color will be used.
- **Font**

  The font of the label. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.
- **Icon**

  An image file with an icon for the label. Can be empty.
- **Text**

The text of the label. Can be empty.

- **Maximum number of digits**

  The maximum number of digits that are acceptable for user input. If zero, there is on limit.

- **Minimum number of digits**

  The minimum number of digits that are acceptable for user input.

- **Font**

  The font of the text field. If empty, the default font is used. Use the font name "dialog" for the default label font and a "0" size value for the default size.

- **Use grouping separator**

  If set, a locale-dependent grouping separator is displayed.

- **Variable name**

  The name of the variable to which the user input is assigned. The type of the variable value is java.lang.Long.

B.5.11 Installer - Custom Code

Custom code is used for

- specifying additional libraries that can be used in scripts and expressions [p. 172] of screens [p. 105] , actions [p. 117] and form components [p. 143] .
- developing new types of actions, screens or form components with the install4j API. Please see the for more information.

  Before you start to develop a new action, please have a look at the available actions [p. 118] and screens [p. 106] . If it's just a few lines of code, you can use the "Run script" action to enter them directly into install4j. If you would like to collect user input, most use cases can be solved with a "Configurable form" screen.

In the **[Custom code]** section you can specify the location of your custom code. The following custom code location types [p. 170] are available:

- Class or resource files
- Directories
- Archives

All classes used by your custom code have to be included in these locations (except for Java runtime classes and install4j framework classes).

The control buttons allow you to modify the contents of the list of custom code locations, the **[Add]** button displays the custom code entry dialog [p. 170] .

After you have chosen your custom code locations, you will be able to select your own screens, actions and form components.

By default, all custom code is packaged separately into the installer. If your custom code uses the same libraries as your installed launchers (such as a database driver), these libraries are duplicated, increasing the size of your installer. If you would like to avoid this duplication you can select the "use installed JAR files where possible" check box. In that case, the install4j compiler checks if a custom code JAR file location is equal to a JAR file that was packaged in the distribution tree and doesn't package it again.

Note: The installed JAR files will be available only after the "Install files" action has run. All your custom code that depends on these JAR files must be placed after the "Install files" action, otherwise a runtime error will occur. If you use installation components, these libraries should be part of a mandatory component.

The "use installed JAR files where possible" option changes the initialization strategy of the installer. If selected, the installer initialized screens and actions up to the "Install files" action at startup and the rest after the "Install files" action has run. If not selected, all screens and actions are initialized at startup.

B.5.12 Installer - Update Options

Please see the help topic on updates [p. 40] for a general discussion on how generated installers handle installations when an earlier version has already been installed.

Every install4j project has an application ID. When you create a new project, the application ID is calculated. The ID is displayed on this tab. If you have to change the ID, you can use the **[Regenerate ID]** button. You can also change the ID manually if the `manually edit ID` check box is checked. You should only change the ID if you want to change the **identity** if your project. The application ID ensures that later versions of your application will be able to find and recognize earlier installations.

install4j offers two types of installers:

- **Regular installer**

  This generates standalone installer. The following options related to updates are available for regular installers:

  - **Suggest previous installation directory**

    If a previous installation can be detected on the computer, the installer will suggest the directory of that previous installation.
  - **Suggest previous program group**

    If a previous installation can be detected on the computer, the installer will suggest the program group of that previous installation.

- **Add-on installer**

  This generates an installer that can **only** be installed on top of an installation of a certain installation. An add-on installer doesn't have a separate uninstaller. This is useful to distribute patches and enhancements.

  If the add-on installer type is selected, you have to enter an application ID for the base application in the text field below. With the **[...]** chooser button, you can select an install4j project file from your file system, and extract its application ID.

B.5.13 Dialogs

B.5.13.1 Custom Code Entry Dialog

The custom code entry dialog is shown when clicking on the add button in the Custom Code tab [p. 168] .
The following entry types are available:

- **Class or resource files**

  For simple actions, screens or form components that do not depend on other classes, it is easiest to insert their class files directly, especially if you build your installer extensions together with your application. Anonymous inner classes will be included automatically. If you select a resource file, e.g. an image, it will be added to the top-level directory of the custom JAR file and will be available via `Class.getResourceAsStream()`.

- **Directories**

  With this type of entry you can add an entire directory. Please make sure to select a classpath root directory, otherwise your classes cannot be loaded.

- **Archives**

  With this type of entry you can add a JAR file. JAR files can optionally ne mapped to installed JAR files, so that they are not duplicated if they are used by both custom code and launchers. Please see the help on the Custom Code tab [p. 168] for more information.

Use the **[...]** chooser button to select files and directories from your file system. A relative path will be interpreted relative to the project file.

B.5.13.2 Class Selector Dialog

The custom class selection dialog is shown when you add an action [p. 117] , a screen [p. 105] or a form component [p. 143] from your custom code.

The custom class selector shows all classes that implement the appropriate interface, depending on the context:

- `com.install4j.api.actions.InstallAction` for actions in the installation mode.
- `com.install4j.api.actions.UninstallAction` for actions in the uninstallation mode.
- `com.install4j.api.screens.InstallerScreen` for screens in the installation mode.
- `com.install4j.api.screens.UninstallerScreen` for screens in the uninstallation mode.
- `com.install4j.api.formcomponents.FormComponent` for form components.

Note that you usually do not implement these interface directly but rather extend one of the abstract classes in the respective packages.

Please see the API description for a detailed explanation of these base classes.

B.5.13.3 Registry Dialog

The registry dialog is displayed when you add a standard action [p. 117] , screen [p. 105] or form component [p. 143] . It shows all built-in elements as well as any elements contributed by installed extensions [p. 47] .

The registry dialog is **quick-search enabled**, you can start typing your query when the tree is focused. The search term will be displayed in a yellow dialog at the top of the tree. If no match is found, the

search term is displayed in red. If a match is found, the search term is displayed in black and the match is made visible. The matched portion is drawn inverted with a green background.

To navigate between matches, you can use the arrow keys or `F3` and `SHIFT-F3`.

You can use wildcards in your search term, for example: `Font*Handle`.

B.5.13.4 Application Templates Dialog

The application templates dialog is displayed when you add an application [p. 102] on the screens & actions [p. 172] tab.

Available application templates are grouped into categories. If you select the top-level `Empty custom application`, a new application will be added that initially does not contain any screens and actions.

install4j comes with several updater templates. Please see the help topic on auto-update functionality [p. 28] for more help on creating updaters.

The application templates dialog is **quick-search enabled**, you can start typing your query when the tree is focused. The search term will be displayed in a yellow dialog at the top of the tree. If no match is found, the search term is displayed in red. If a match is found, the search term is displayed in black and the match is made visible. The matched portion is drawn inverted with a green background.

To navigate between matches, you can use the arrow keys or `F3` and `SHIFT-F3`.

You can use wildcards in your search term, for example: `Font*Handle`.

B.5.13.5 String Edit Dialog

The string edit dialog is shown from the action [p. 117] , screen [p. 105] or form component [p. 143] editors when you click on the **[...]** for a

- a "multi-line string" property. Multi-line strings cannot be edited inline in the property sheet.
- a "list of strings" property. While the inline editor in the property sheet accepts items separated by semicolons (';'), this dialog separates item by line breaks. When you wish to enter a new item, you have to put it on a new line.

All key bindings in the editor are configurable. Choose *Settings->Key Map* to display the Key map editor [p. 175] .

The editing functionality in the *Edit* menu includes:

- **Undo/Redo**
- **Copy/Cut/Paste**

  The "Paste with dialog" action shows previous selections.
- **Rectangular selections**
- **Extended selection and deletion**

  This included actions like "Select word" and "Delete line".
- **Join lines**
- **Duplicate lines**
- **Indent/Unindent selection**
- **Toggle case**

By choosing *Edit->Insert Variable* from the menu, you can add a compiler variable or custom localization key at the current cursor position. It will be added with the text field variable syntax, like ${i18n:myKey} for a custom localization key or ${compiler:myVariable} for a compiler variable.

The search functionality in the *Search* menu includes:

- **Find**

  Find simple or regular expressions in the selected or the entire text with options of case sensitivity and word matching. With "Find next occurrence" and "Find previous occurrence" you can quickly move among the search results.
- **Replace**

  Same as "Find" with an option to replace the found items.
- **Quick search**

  Search text by typing directly in the editor and highlighting the search results as you type.

B.5.13.6 Java Code Editor

The Java code editor is shown from the screens & actions [p. 99] tab or the form component [p. 143] editor when you click on the **[...]** for a Java code property.

Please see the help on the string editor dialog [p. 171] for common editing functionality.

The box above the edit area show the available parameters for the Java code property as well as the return type. If parameters or return type are classes (and not primitive types), they will be shown as hyperlinks. Clicking on such a hyperlink opens the Javadoc in the external browser. If you would not like the default browser to be opened, you can configure your own browser in the preferences dialog [p. 196] .

To get more information on classes from the `com.install4j.*` packages, please choose *Help->Show Javadoc Overview* from the menu and read the the help topic for the install4j API [p. 44] . In addition, the Java code editor offers a code gallery [p. 175] that contains useful snippets that show you how to get started with using the install4j API. The code gallery is invoked from the tool bar or by choosing *Code->Insert from Code gallery* from the menu.

A number of packages can be used with using fully-qualified class names. Those packages are:

- java.util.*
- java.io.*
- javax.swing.*
- com.install4j.api.*
- com.install4j.api.beans.*
- com.install4j.api.context.*
- com.install4j.api.events.*
- com.install4j.api.screens.*
- com.install4j.api.actions.*
- com.install4j.api.formcomponents.*
- com.install4j.api.update.*
- com.install4j.api.windows.*
- com.install4j.api.unix.*

You can put a number of import statements as the first lines in the text area in order to avoid using fully qualified class names.

Java code properties can be

- **expressions**

  An expression doesn't have a trailing semicolon and evaluates to the required return type.

  Example: `!context.isUnattended() && !context.isConsole()`

  The above example would work as the condition expression of an action and skip the action for unattended or console installations.

- **scripts**

  A script consists of a series of Java statements with a return statement of the required return type as the last statement.

  Example: `if (!context.getBooleanVariable("enterDetails")) context.goForward(2, true, true); return true;`

  The above example would work as the validation expression of a screen and skip two screens forward (checking the conditions of the target screen as well as executing the actions of the current screen) if the variable with name "enterDetails" is not set to "true".

install4j detects automatically whether you have entered an expression or a script.

The primary interface to interact with the installer or uninstaller is the **context** which is always among the available parameters. The context provides information about the current installation and gives access to variables, screens, actions and other elements of the installation or uninstallation. The parameter is of type

- `com.install4j.api.context.InstallerContext` for screens and actions in the installation mode
- `com.install4j.api.context.UninstallerContext` for screens and actions in the uninstallation mode
- `com.install4j.api.context.Context` for form components.

Apart from the context, the action, screen or form component to which the Java code property belongs is among the available parameters. If you know the actual class, you can cast to it and modify the object as needed.

The Java editor offers the following code assistance powered by the eclipse platform:

- **Code completion**

  Pressing `CTRL-Space` brings up a popup with code completion proposals. Also, typing a dot (".") shows this popup after a delay if no other character is typed. While the popup is displayed, you can continue to type or delete characters with `Backspace` and the popup will be updated accordingly. "Camel-hump completion" is supported, i.e. typing `NPE` and hitting `CTRL-Space` will propose `NullPointerException` among other classes. If you accept a class that is not automatically imported, the fully qualified name will be inserted.

  The completion popup can suggest:

  - variables and default parameters. Default parameters are displayed in bold font.
  - packages (when typing an import statement)
  - classes
  - fields (when the context is a class)
  - methods (when the context is a class or the parameter list of a method)

You can configure code completion behavior in the Java editor settings [p. 174] .

- **Problem analysis**

  The code that you enter is analyzed on the fly and checked for errors and warning conditions. Errors are shown as red underlines in the editor and red stripes in the right gutter. Warnings (such as an unused variable declaration) are shown as a yellow backgrounds in the editor and yellow stripes in the right gutter. Hovering the mouse over an error or warning in the editor as well as hovering the mouse over a stripe in the gutter area displays the error or warning message.

  The status indicator at the top of the right gutter is

  - **green**

    if there are no warnings or errors in the code.
  - **yellow**

    if there are warnings but no errors in the code.
  - **red**

    if there are errors in the code. In this case the code will not compile and the installer cannot be generated.

  You can configure the threshold for problem analysis in the Java editor settings [p. 174] .

- **Context-sensitive Javadoc**

  Pressing `SHIFT-F1` opens the browser at the Javadoc page that describes the element at the cursor position. If no corresponding Javadoc can be found, a warning message is displayed. Javadoc for the Java runtime library can only be displayed if a design time JDK is configured and a valid Javadoc location is specified in the design time JDK configuration [p. 61] .

  You can set the design time JDK in the Java editor settings [p. 174]

All key bindings in the Java code editor are configurable. Choose *Settings->Key Map* to display the Key map editor [p. 175] .

Screens, actions and form components are wired together with **installer variables**, please see the help topic on screens and actions [p. 11] for more information. Setting and getting installer variables is done through the context parameter with the `context.getVariable(String variableName)` and `context.setVariable(String variableName, Object value)` methods. The convenience method `context.getBooleanVariable(String variableName)` makes it easier to check conditions. Any object can be used as the value for a variable. To use installer variables in text properties of actions, screens and form components, write them as `${installer:myVariableName}`.

If the gutter icon in the top right corner of the dialog is green, your script is going to compile unless you have disabled error analysis in the Java editor settings [p. 174] . In some situations, you might want to try the actual compilation. Choosing *Code->Test Compile* from the menu will compile the script and display any errors in a separate dialog. Saving your script with the **[OK]** button will not test the syntactic correctness of the script. When your install4j project is compiled, the script will also be compiled and errors will be reported.

B.5.13.7 Java Editor Settings

The Java editor settings dialog is shown when you select *Settings->Java Editor Settings* from the menu in the Java code editor dialog [p. 172] .

In the **code completion popup settings** section, you can configure the following options:

- **Auto-popup code completion after dot**

If selected, each time you type a dot (".") in the Java code editor, the code completion popup will be displayed after a certain delay unless you type another character in the meantime.

- **Delay**

  The "Auto-popup code completion after dot" feature above uses a configurable delay. By default, the delay is set to 1000 ms.

- **Popup height**

  The height of the completion popup in number of entries is configurable.

In the **display code problems** section, you can configure the threshold for which code problems are shown in the editor.

- **None**

  No code problems are displayed at all.

- **Errors only**

  Only problems that prevent code compilation are displayed. Errors show as red underlines in the editor and red stripes in the right gutter.

- **Errors and warnings**

  In addition to errors, warnings are displayed. Warnings cover all kinds of suspicious conditions that could be sources of bugs such as an unused local variable. Warnings are displayed as yellow backgrounds in the editor and yellow stripes in the right gutter.

The **design time JDK** section mirrors the design time JDK configuration on the Java version [p. 52] tab of the general settings [p. 50] .

B.5.13.8 Code Gallery

The code gallery dialog is displayed by choosing *Code->Insert from Code Gallery* from the menu in the Java code editor dialog [p. 172] .

Available code snippets are grouped into categories. They show how to use the install4j API in common use cases. The script is shown in a preview on the right side. You can either copy a portion of the script with `CTRL-C` or click **[OK]** to insert the entire script at the current cursor position.

Please note that not all code snippets might be directly usable in the script that you are editing.

Some java script properties have **special code snippets** that are only shown for this property. If such code snippets exist, they are displayed in a category with the name of the java script property in bold font.

The code gallery dialog is **quick-search enabled**, you can start typing your query when the tree is focused. The search term will be displayed in a yellow dialog at the top of the tree. If no match is found, the search term is displayed in red. If a match is found, the search term is displayed in black and the match is made visible. The matched portion is drawn inverted with a green background.

To navigate between matches, you can use the arrow keys or `F3` and `SHIFT-F3`.

You can use wildcards in your search term, for example: `Font*Handle`.

B.5.13.9 Key Map Editor

The key map editor is displayed by choosing *Settings->Key map* from the menu in the Java code editor dialog [p. 172] or the string edit dialog [p. 171] .

The active key map controls all key bindings in the editor. By default, the **[Default]** key map is active. The default key map cannot be edited directly. To customize key bindings, you first have to copy the

default key map. Except for the default key map, the name of a key map can be edited by double-clicking on it.

When assigning new keystrokes or removing existing key strokes from a copied map, the changes to the base key map will be shown as "overridden" in the list of bindings.

The key map editor also features search functionality for locating bindings as well a conflict resolution mechanism.

Key bindings are saved in the file *$HOME/.install4j4/editor_keymap.xml*. This file only exists if the default key map has been copied. When migrating an install4j installation to a different computer, you can copy this file.

B.6 Step 5: Media

B.6.1 Step 5: Configure Media

Media files are the final output of install4j: single installer files that are used to distribute your application to your users. The creation of a media file has platform dependent options, so for each platform, you have to define a media file. It also makes sense to define several media files for one platform in case you wish to distribute different subsets of your distribution tree, or if you distribute your application with and without a bundled JRE.

To define a new media file, you double-click on the new media file entry in the list of defined media files or choose *Media->New media file* from install4j's main menu. The first step of the media wizard will then be displayed. The subsequent steps [p. 180] depend on your choice of the media file type [p. 178] in this first step.

Once you have completed all steps of the media wizard and clicked **[OK]** in the final step, a new media file entry will be displayed in the list of media files.

In the list of media files, you can

- **Reorder media file definitions**

  Media file definitions are reordered by dragging them with the mouse to the desired location. While dragging, the insertion bar shows you where the media file definition would be dropped. The order of media files determines the order in which the media files are generated. Reordering is mainly provided for the purpose of letting you arrange the media file definitions according to your personal preferences.
- **Copy media file definitions**

  Media file definitions can be copied by copy-dragging a media file definition or using the corresponding tool bar button or menu entry. The name of the copied media file definition will be prefixed with "Copy of".
- **Rename media file definitions**

  Media file definitions can be renamed by using the corresponding tool bar button or menu entry. An input dialog will be displayed where the current name can be edited. Please note that except for the `%SETNAME%` variable used in the media file options [p. 56] , the name of the media file is not used in the distribution but is for your own information only.
- **Delete media files definitions**

  Media file definitions can be deleted by hitting the `DEL` key or using the corresponding tool bar button or menu entry.
- **Edit a media file definition**

  Media file definitions can be edited by hitting the `ENTER` key or using the corresponding tool bar button or menu entry.

  The appropriate media wizard [p. 177] will be displayed for the selected media file definition. Note that you can directly access any step in the wizard by clicking on it in the index.

Each media file definitions has an ID which can be used to select certain media files when building the project from the command line [p. 198] . To show all IDs, choose *Media->Show media file IDs* from the main menu. The IDs will then be shown in square brackets next to the names of the media file definitions.

B.6.2 Available Media File Types

There are two fundamentally different types of media files:

- **Installers**

  The media file is an executable that invokes the installer. Optionally, the installer can be executed as an unattended installer or as a console installer. Please see the corresponding help topic [p. 32] for more information.

  - **Windows media file**

    a media file for Windows is a native setup executable that installs your application with an installer wizard.

    The installer can download a JRE if no suitable JRE is found on the target system.
  - **Mac OS X single bundle media file**

    a single bundle media file for Mac OS X is a .dmg file that extracts an installer wizard (by double clicking on it). The wizard installs your application as a **single application bundle**. If you wish to support multiple launchers, please choose the "Mac OS X folder media wizard" (see below).

    The default JRE (which is always present on Mac OS X) is used during the installation phase.

    If you would like to create a separate directory next to the generated application bundle that contains user files, you cannot add it directory to the "Installation directory" root of the distribution tree, since all files under that node will end up in the application bundle. The solution to this problem is to use the single bundle installer and to add another installation root to the distribution directory, that root should be set to

    ```
    ${installer:sys.installationDir}/My Application Documents
    ```

    if you want to call the additional folder "My Application Documents". That folder will be created next to the installed application bundle.
  - **Mac OS X folder media file**

    a folder media file for Mac OS X is a .dmg file that extracts an installer wizard (by double clicking on it). The wizard installs your application as a folder that contains the entire distribution tree and **multiple application bundles** for each included launcher.

    The default JRE (which is always present on Mac OS X) is used during the installation phase.
  - **Unix/Linux GUI installer media file**

    a Unix/Linux GUI installer media file is an executable shell script that extracts an installer and installs your application with an installer wizard.

    The installer can download a JRE if no suitable JRE is found on the target system.

- **Archives**

  The media file is an archive that the user can extract to an arbitrary location. No screens are shown and no actions are executed. If you define additional installation roots, the files in them are not installed. No components can be downloaded.

  Archives are intended as a fallback or as additional packages such as documentation bundles. If your installer heavily relies on actions, screens and additional installation roots, you should not use archives to distribute your application. The main advantages of archives such as the ability to

install them at the command line is also available from installers by using their unattended or console installation modes [p. 32] .

- **Windows archive media file**

  an archive media file for Windows is a ZIP-file that contains your application.

  Note: This media file type **does not have a GUI installer**. If you wish to create a GUI installer for Windows, please choose the "Windows media wizard" (see above).

- **Mac OS X single bundle archive media file**

  a single bundle media file for Mac OS X is a .tgz archive that extracts a **single bundle** for your application. If you wish to support multiple launchers, please choose the "Mac OS X folder archive media wizard" (see below).

  Note: This media file type **does not have a GUI installer**. If you wish to create a GUI installer for Mac OS X, please choose the "Mac OS X single bundle media wizard" (see above).

- **Mac OS X folder archive media file**

  a folder media file for Mac OS X is a .tgz archive that contains the entire distribution tree and **multiple application bundles** for each included launcher.

  Note: This media file type **does not have a GUI installer**. If you wish to create a GUI installer for Mac OS X, please choose the "Mac OS X folder media wizard" (see above).

- **Linux RPM media file**

  an RPM archive for Linux can be installed and uninstalled with the *rpm* command on the most popular Linux distributions. There are also a large number of graphical package management tools that Linux users can use to install an RPM archive.

  If you do not wish to use RPM for your Linux media file or if you wish to provide an alternative installation package which does not use RPM, please use the Unix/Linux archive media wizard (see below).

  Note: This media file type **does not have a GUI installer**. If you wish to create a GUI installer for Linux, please choose the "Unix/Linux GUI installer media wizard" (see below).

- **Unix/Linux archive media file**

  a Unix/Linux archive media file is a gzipped TAR archive that contains your application.

  Note: This media file type **does not have a GUI installer**. If you wish to create a GUI installer for Unix or Linux, please choose the "Unix/Linux GUI installer media wizard" (see above).

Note: GUI launchers on Mac OS X only start a single instance of your application. Subsequent launches will not start additional JVMs. You can use the `com.install4j.api.launcher.StartupNotification` from the install4j API to be informed about those invocations.

B.6.3 Media File Wizards

The media file wizard is displayed when you add a new media file or when you edit an exiting media file. To learn more information about the various media file types, please see the overview [p. 178] .

The media file wizards show a number of steps which depend on the media file type. Common steps are:

- Platform [p. 181]

  Choose the media file type.
- Installer options [p. 182]

  Define options for the installer.
- Data files [p. 184]

  Specify where the installer data should be placed. Not displayed for archives.
- Bundled JRE [p. 186]

  Decide if and how a JRE should be bundled with the installer. Not displayed for Mac OS X media file types.
- Customize project defaults [p. 188]

  A number of project settings can be customized on a per-media file basis.

In addition, there are a number of steps that depend on the media file type:

- 32-bit or 64-bit [p. 190]

  For Windows media files only.
- Code signing [p. 191]

  For Windows media files only.
- Launcher [p. 192]

  For Mac OS X single bundle media files only.

B.6.4 Wizard steps

B.6.4.1 Media File Wizard: Platform

In this step of the media file wizard [p. 180] you select the media file type [p. 178]. If you are creating a new media file definition, the subsequent steps are undefined at this point.

If you are editing an existing media file definition, changing the media file type away from from the current selection will change the wizard into a different one and data that has been entered in the subsequent steps will be lost after a warning message has been confirmed.

B.6.4.2 Media File Wizard: Installer Options

In this step of the media file wizard [p. 180] you define options for the installer, most importantly the default installation directory.

This step is different for installers and archives:

- **Installers:**
  - **Installation directory**

    Enter a simple directory name (without backslashes). The standard location for applications will we prepended to this directory name. In other words: do not enter `C:\Program Files\MyApplication` but only `MyApplication` The installer will find out the correct equivalent for `C:\Program Files`, `/opt` or similar standard locations at runtime. By default, install4j will suggest the short name you have entered in the general application options [p. 51] . It is also possible to enter a composite relative directory like `My Corp\MyApplication`.
  - **Use custom installation base directory**

    If you do not want to install your application to the standard application directory, you can enter a custom base directory here. This is useful for internal deployments with non-standard directory policies. The installation directory entered above will be appended to the custom base directory. For example, if the application should be installed in `D:\apps\MyApplication`, check the custom installation base option, enter `D:\apps` in the text field below it and enter `MyApplication` in the installation directory text field above.

    On Unix, if you want to suggest a directory below the user home directory, you can use ~ as the custom installation base directory.

- **Archives:**

  With the installation directory you determine the top level directory for the archive. All files will be contained in the top level directory. Enter a simple name without slashes, such as `myapp`. By default, install4j will suggest the short name you have entered in the general application options [p. 51] .

  For Mac OS X single bundle archives, it is not possible to set an installation directory since all files in a single bundle are in contained in a single directory whose name is determined by the name of the main launcher. The user can move the entire bundle somewhere else by dragging the displayed icon.

For **Windows installer media files**, this step includes an option `Request admin privileges on Windows Vista` that allows you to compile the installer and uninstaller executables in such a way that Windows Vista will know up-front that the installer and uninstaller require admin privileges. If these privileges are not available, Windows Vista will present a dialog that allows the user to specify the credentials of an admin user.

Select this option if your installer always requires admin rights on Windows Vista. This option does **not** replace the "Require admin user" action [p. 117] . Contrary to the cross-platform "Require admin user" action, this option is static and influences the way the installer executable is launched. When the "Require admin user" action is executed and the installer doesn't run with admin rights, the user is informed with a message box and the installer quits.

If you do not select this option the execution level "Highest available" is used on Windows Vista. For a standard user the installer will run with the rights of this user. If the user is an Administrator with filtered rights, Windows Vista will ask him if the installer should be allowed to gain full administration rights.

For **Unix/Linux GUI installer media files**, this step has a **Installer custom script** sub-step.

If you specify a bourne shell custom script, the entered script fragement will be inserted into the launcher script immediately before the Java invocation of your installer takes place. This is a hook for experienced users to make custom changes in the environment.

You can select one of:

- **No custom script**

  No custom script will be inserted.
- **Custom script from file**

  Specify a file from which the custom script will be read. If you enter a relative file, the file will be interpreted relative to the project file.
- **Direct entry of custom script**

  Enter your custom script in the text area below.

For **Linux RPM media files**, this step has sub-steps where you can define scripts to run before and after installation or uninstallation by the rpm executable. The available hooks are:

- Pre-install script
- Post-install script
- Pre-uninstall script
- Post-uninstall script

Please see http://www.rpm.org for more information.

You can select one of:

- **No custom script**

  No custom script will be inserted.
- **Custom script from file**

  Specify a file from which the custom script will be read. If you enter a relative file, the file will be interpreted relative to the project file.
- **Direct entry of custom script**

  Enter your custom script in the text area below.

B.6.4.3 Media File Wizard: Data Files

In this step of the media file wizard [p. 180] you define where the installer data should be placed.

Typically, installers are single files that contain all data that they install on the user's request. There are three common use cases where this is not the case:

- **CD/DVD installers with large data files**

  If your application relies on large amounts of data, it is often distributed on a CD or DVD. In that case, you typically ship a number of external data files that you do not wish to package inside the installer. The installer should start up quickly and the data files should not be extracted from the installer in order to save time. The user might decide to install only certain components, so some data files might not be needed at all. If they are included in the installer executable, all this data would have to be read from disk.
- **Installers with large optional components**

  Some applications have large optional components that are not relevant for the typical user. To reduce download size for the majority, the optional component should be downloadable on demand.
- **Net installers**

  Some application are highly modular, so it is not feasible to build a set of installers for typical use cases. A net installer lets the user select the desired components and downloads them on demand. The download size of the net installer is small since no parts of the application are contained in the installer itself.

To accommodate the above use cases, install4j offers three different ways to handle the installer data files:

- **Included in media file**

  All data files are included in the installer so you can ship it as a single download.
- **External**

  This mode covers the "CD/DVD installers with large data files" use case.

  All data files are placed in a directory next to your installer that has the name of your installer with the extension **.dat**. For example, if your media file name is *`hello_4_0`* (resulting in a Windows installer executable *`hello_4_0.exe`*), the directory containing the external data files is named *`hello_4_0.dat`*. You have to ship this directory in the same relative location on your CD or DVD.

  The number of data files depends the definition of your installation components. The data files are generated in such a way that

  - the files for an installation component are contained in one or more data files
  - there are no files in those data files that do not belong to this installation component

  If components do not overlap, there's a one-to-one correspondence between data files and installation components.
- **Downloadable**

  This mode covers the "Installers with large optional components" and "Net installers" use cases. It can only be used if you define installation components [p. 73] .

  Data files are generated just like for the "External" mode, but only for installation components that have been **marked as downloadable** in the installation component definition [p. 73] . If no installation components are marked as "downloadable", this mode will behave like the "Included in media file" mode. For a "net installer", all installation components are "downloadable".

For this mode, you have to enter a **HTTP download URL**, so the installer knows from where it should download the data files at runtime if the user requests downloadable components. The URL must begin with http:// and point to a directory where you place the data files. For example, if the data file *hello_windows_4_0.000* is downloadable and the download URL is *http://www.test.com/components*, the data file must be uploaded to the web server, so that the installer can download the data file from the URL *http://www.test.com/components/hello_windows_4_0.000*.

Any data files that you leave in the data file directory next to the installer will not be downloaded. This means that if you test your installer directory from the location where it was generated, the installer finds all data files in the data file directory and does not try to download them.

B.6.4.4 Media File Wizard: Bundled JRE

In this step of the media file wizard [p. 180] you define if an how a JRE should be bundled with the installer.

If you choose to bundle a JRE by selecting the "`bundle the following JRE`" radio button, you have to choose a JRE from the drop down list below it. If you select "manual entry" in the drop down list, a text field will appear where you can enter a file name with compiler variables. Please note that you cannot enter a path here, but only a bundle filename that will be searched in *`$INSTALL4J_HOME/jres`* and *`{user home directory}/.install4j4/jres`*. The existence of this bundle file will be checked only at compile time, not by the wizard.

You can download additional JREs with the JRE download wizard [p. 194] . Click **[JRE download wizard]** to download the JREs you need. The drop down list will be updated after the download has finished.

If you wish to bundle a JRE that is not available from ej-technologies' download server or that has custom modifications (like an installation of the `javax.comm` API), please see the JRE bundle creation wizard [p. 195] on how to create your own JRE bundle.

Further options and runtime behavior are different for installers and archives:

- **Installers:**

  The `Bundle type` section allows you to choose between the two JRE bundling modes offered by install4j:

  - **Static bundle**

    The selected JRE will be **distributed in your media file**. install4j will automatically adjust the JRE search sequence [p. 52] of all generated launchers and include the bundled JRE as the first choice. A statically bundled JRE will always be distributed in the directory *`jre`* right below the installation root directory [p. 63] .

    When you update your application and include a static JRE bundle again, the old JRE bundle will be deleted prior to installation, so that any files left over from the old JRE cannot interfere with the new JRE.

  - **Dynamic bundle**

    A dynamic bundle is **downloaded on demand**. If the user already has a suitable JRE installed, that JRE will be used. If there is no such JRE available on the target machine, the installer will download the dynamically bundled JRE from the URL that you specify in the text fields below.

    To enable the download on demand, you have to locate the corresponding *`.tar.gz`* bundle archive in the *`jres`* subdirectory of your install4j installation and place it on a server so that the `HTTP download URL` will point to the bundle archive. The URL has to be of the form *`http://www.myserver.com/somewhere/windows-x86-1.3.1_08.tar.gz`*.

    In addition, an optional FTP fallback download URL can be specified in the second text field. This URL must be of the form: *`ftp://ftp.myserver.com/somewhere/windows-x86-1.3.1_08.tar.gz`*. The FTP server must support anonymous access.

    If the installer determines that there is no suitable JRE present, it will ask the user whether the JRE should be downloaded.

    - On Windows, a progress bar with download speed and estimated duration will be displayed during the download.

- On Unix-like systems, the progress will be shown in the terminal. Adding an FTP download URL will increase the chance that the download will work on Unix-like systems behind restrictive firewalls.

If the download fails or is aborted by the user, the download URL will be displayed together with instructions on where to place the downloaded bundle archive.

You can **override the default JRE search** in a Microsoft Windows installer executable by passing the argument `-manual` to the installer executable. The installer will then report that no JRE could be found and offer you to locate one in your file system. If you have set up a dynamic JRE bundle, it will also offer you to download one. This is a good way to test if your download URL is correct.

The check box `install as a shared JRE` determines whether the bundled JRE should be private for the application or whether other applications distributed with install4j can share this JRE. The following scenarios are covered by this approach:

- If you distribute several applications with dynamically bundled JREs, installing as a shared JRE is advisable, since the user will have to wait for the download only once.
- If you have a main application and several add-on applications, it makes sense to statically bundle the JRE with the main application and install it as a shared JRE, so the add-on applications can be distributed without JRE.

Note: installers generated by install4j will never install a JRE on the system path or make Windows registry changes. The term "shared installation" only applies to applications distributed with install4j. Other applications will not be able to use such a JRE.

- **Archives:**

  install4j will automatically adjust the JRE search sequence [p. 52] of all generated launchers and include the bundled JRE as the first choice. The JRE will always be distributed in the directory *`jre`* right below the installation root directory [p. 63] .

B.6.4.5 Media File Wizard: Customize Project Defaults

In this step of the media file wizard [p. 180] you can customize a number of project settings on a per-media file basis. Customizable settings are displayed in sub-steps that can be selected with the **[Choose customization category]** button or by clicking into the index on the left side.

The customization categories are:

- **Compiler variables**

  Compiler variables [p. 17] that are defined on the Compiler Variables tab [p. 58] can be overridden on a per-media file basis. For example, this would be useful to adjust the native library directories [p. 92] in a launcher definition.
  The variable table shows 4 columns:

  - **Override marker**

    If you have overridden a variable by editing the value column, the first column will display a marker to indicate that that variable has been overridden. In that case, the reset button column will display a button to restore the original value.
  - **Variable name**

    Shows the name of the variable.
  - **Variable value**

    Shows the value of the variable. This column is editable. To override the variable, double-click on the desired cell in this column and edit the value. The override marker column and the reset button column will then show that the variable has been overridden.
  - **Reset button**

    If a variable has been overridden, this column shows a **[Reset]** button that allows you to restore the original value as defined on the Compiler Variables tab [p. 58] .

  Overriding compiler variables on a per-media file basis is also possible from the command line [p. 197] and from the ant task [p. 201] .
- **Media file name**

  The file name pattern defined in the Media File Options tab [p. 56] determines the actual name of the media file. If you want to override that pattern, you can enter an individual name here. To enter an individual name, select the `custom name` radio button and enter a file name in the text field below it.
- **Principal language**

  By default, the language used by an installer is governed by the setting on the Languages [p. 54] tab. If you would like to generate installers with fixed languages, you leave those settings at their default values and override the principal language and custom localization file here.

  You can change the principal language for all media files or on a per-media file basis from the command line [p. 197] or from the ant task [p. 201] by defining the variable `LANGUAGE_ID` with the 2-letter ISO code of the desired language (see http://www.w3.org/WAI/ER/IG/ert/iso639.htm).
- **Exclude components**

  Here, you can select components that should **not** be distributed by selecting their attached check boxes. This is useful if you have installation components that do not work with specific media files, such as a Windows-only extension, for example.
- **Exclude files**

This step is useful to tailor your distribution to platform-specific needs. The distribution tree is shown in **expanded form** and shows all files. This is unlike the distribution tree in the Files step [p. 63] which shows the **definition** of the distribution tree.

Each file and subdirectory has a check box attached. If you select that check box, the entry will **not** be distributed. Selections of subdirectories are recursive. If you select a subdirectory, its contents are hidden from the tree since they will be excluded anyway.

- **Exclude launchers**

  This step is complementary to the "Exclude files" screen where launchers are not shown. Each launcher has a check box attached. If you select that check box, the launcher will **not** be generated.

- **Exclude installer elements**

  If you some installer applications, screens or actions should not be included with this media file, you can exclude those elements by selecting their attached check boxes in the tree of installer elements. Note that for more complex cases you can also skip screens by entering a condition property for screens [p. 105] and actions [p. 117] .

  For archives, only custom installer applications are shown, since installer and uninstaller are not present for archives.

B.6.4.6 Media File Wizard: 32-bit or 64-bit (Windows only)

This step of the media file wizard [p. 180] is specific to Windows media file types.

On Windows, a native executable can be either a 32-bit or a 64-bit executable. If you need a 64-bit JRE for your application you can choose to generate 64-bit installers and launchers for a media file.

Note that it is not possible to create launchers that work with both 64-bit and 32-bit JREs. Since the launcher starts the JVM with the JNI interface by loading the JVM DLL, the architecture has to be the same. If you target both 32-bit and 64-bit JREs and operating systems, you have to generate different media files for them.

On a 64-bit Windows, there are separate system directories for 32-bit and 64-bit applications. If you enable the 64-bit executable mode in this step, those system directories will be appropriate for 64-bit applications, e.g. `c:\Program Files` instead of `c:\Program Files (x86)`.

B.6.4.7 Media File Wizard: Code Signing

In this step of the media file wizard [p. 180] you can optionally configure code signing for all generated executables. Code-signing ensures that the installer as well as the uninstaller are trusted executables on Windows Vista. For unsigned applications that require admin privileges, Window Vista will display a special warning dialog. Also, Windows XP users who use Internet Explorer receive a different warning dialog when trying to execute a downloaded installer.

The install4j compiler can invoke a post-processor for each executable that is generated. This includes

- generated launchers
- the installer
- the uninstaller

In the post processor text field you can use the `$EXECUTABLE` variable to reference the executable that is currently being post-processed. The working directory of the executed process is the directory your config is located in so you can use relative filenames for key or certificate files.

If you run the build on Windows, you can use the Authenticode tools from the Windows SDK to sign the executable. Older Platform SDKs as well as the .NET v1 SDKs contain the tool signcode.exe. The newer SDKs contain the tool signtool.exe. Both tools are equally suited for code signing with install4j. Please refer to the MSDN documentation for detailed information.

It is also possible to sign executables on other platforms. The *`$INSTALL4J_HOME/resource/signcode.exe`* executable is a mono executable modified by ej-technologies to support signing of 64-bit executables. This executable can only be executed if mono is installed. Mono is available for a number of platforms and can be downloaded free of charge.

The tool has the same syntax as the one from Microsoft. A typical entry would be *`mono /opt/install4j/resource/signcode.exe -spc mycert.spc -v mykey.pvk -vp password -t http://timestamp.verisign.com/scripts/timstamp.dll $EXECUTABLE`*

Some SPC files cannot be read directly by this tool. If this is the case for your certificate, you can export all CER files from the SPC file and generate a new SPC file with the *`cert2spc`* tool included with mono. You have to add the CER files in the order of the certificate chain (your own certificate is the last one on the command line).

B.6.4.8 Media File Wizard: Launcher (Mac OS X single bundle only)

This step of the media file wizard [p. 180] is specific to single bundle media file types for Mac OS X.

A Mac OS X single bundle media file supports a single launcher only. Please choose a launcher from the drop-down list of defined launchers. Only GUI launchers are shown.

Note: external launchers are not supported for single bundles and are not shown in the list of launchers.

B.7 Step 6: Build

B.7.1 Step 6: Build the Project

In the **Build step**, all defined media files [p. 177] are generated. There are two different build modes:

- **Regular build**

  Build the media files and place them in the media file output directory [p. 56] .

- **Test build**

  Build the media files in the temporary directory only. This mode allows you to check whether your configuration is ok while not making any changes to your file system.

When a build or a test build is started, important status messages are displayed in the text area labeled `build output`. A progress bar appears in the status bar of install4j's main menu which indicates what percentage of the total build has been completed so far. The build process is asychroneous, so you can change to other steps while it is running. The status bar will inform you when the build process has finished.

With the `Build selection` section you can choose which media files should actually be built. This can be useful for testing purposes, or if you have defined media files that should not be built by default.

With the standard setting, all media files will be built. If you select the radio button `build selected`, only the media files that are selected in the adjacent list will be built.

These settings are persistent and are saved in your project file, however, when you build from the command line, your build selection will be ignored unless you specify a special parameter. Please see the help on the command line compiler options [p. 198] for further details.

Also, have a look at the available build options [p. 193] .

B.7.2 Build options

The following options are available for the build process [p. 193] :

- **Enable extra verbose output**

  If this option is checked, much more information than usual will be printed in the build output text area. This can be useful for getting more information about the reason of a build failure.

- **Do not delete temporary directory**

  If this option is checked, the temporary directory where install4j creates the project is not deleted after completion or failure. This can be useful for trouble shooting.

- **Create additional debug installer**

  If this option is checked, directories with debug installers will be created in the media file output directory that allow you to execute the installer as well as the uninstaller with a plain Java invocation from the command line as well as from your IDE. Please see the API documentation [p. 44] for more information.

B.8 JRE Download Wizard

The JRE download wizard lets you download JREs from ej-technologies' servers for easy bundling with your applications. For more information on JRE bundles and on how they are used by install4j, please see the corresponding help topic [p. 25] .

The JRE download wizard is started by

- clicking on the toolbar button of the main window.
- clicking on **[JRE download wizard]** in the Bundled JRE [p. 186] step of the media file wizard [p. 180] .

The JRE download wizard leads you step by step through the process of connecting to the server, choosing the desired JREs and downloading them to your local disk.

The "Connection" step allows you to configure a proxy, with an optional proxy authentication. Note that the password will not be saved in the install4j configuration, you will have to reenter it each time you run the JRE download wizard.

ej-technologies offers JREs for a number of common platforms. The Windows JREs whose names end with `_us_only` do not include support for non-English locales.
If you wish to bundle a JRE that is not available from ej-technologies' download server or that has custom modifications (like an installation of the `javax.comm` API), please see the JRE bundle creation wizard [p. 195] on how to create your own JRE bundle.

B.9 JRE Bundle Wizard

With the JRE bundle wizard you can create JRE bundles for install4j from any JRE installation on your disk. For more information on JRE bundles and on how they are used by install4j, please see the corresponding help topic [p. 25] .

Please check first, if one of the JREs provided by ej-technologies' JRE download server [p. 194] fits your needs.

The JRE bundle wizard is started by choosing *Project->Create a JRE bundle* from install4j's main menu. The JRE bundle wizard leads you step by step through the process of choosing the desired JRE and creating an install4j bundle from it.

In the "Select JRE" step of the wizard you define the origin, the version and the identifier for your JRE bundle:

- **Java home directory**

  Select the Java home directory of the JRE. It is possible to select a JDK, however, after a confirmation dialog the bundle will be created from the included JRE.

- **Java version**

  Enter the version of the JRE. Do not include any build information, the version number should look like `1.4.2`.

- **Custom ID**

  In order to be able to identify your JRE in the "Bundled JRE" step of the media file wizard and to distinguish it from JREs with the same OS, architecture and Java version, a custom ID is required. The custom ID should be a short identifier, like `custom`.

B.10 Preferences Dialog

The preferences dialog is displayed by clicking *Project->Preferences* in install4j's main menu.

The preferences dialog has two panels:

- **Appearance**

  Here, you can select a look and feel for the install4j IDE. This does not affect generated installers, they always use the native look and feel. The newly selected look and feel will be used when install4j is started the next time.

- **Miscellaneous**

  In the `Startup` section of this tab you can specify that install4j should load the last edited project at startup. If this option is not selected, install4j starts with a blank screen. If you specify a project file at the command line, that project will be loaded in any case.

  In the the `Browser section` section of this tab, the browser start command for your preferred browser can be adjusted. Use `$URL` as a variable for the URL to be displayed. This setting is important for generating project reports [p. 7] . If you leave the text box empty, install4j will use the system defaults on Windows and Mac OS X and try to invoke *`netscape`* on Linux and Solaris.

B.11 Command line compiler

B.11.1 Command Line Compiler

install4j's command line compiler *install4jc[.exe]* can be found in the *bin* directory of your install4j installation. It operates on project files with extension *.install4j* that have been produced with the install4j IDE. (*install4j[.exe]*). The install4j command line compiler is invoked as follows:

```
install4jc [OPTIONS] [config file]
```

The available options are described here [p. 198] . A quick help for all options is printed to the terminal when invoking

```
install4jc --help
```

In order to facilitate usage of install4jc with automated build processes, the destination directory [p. 56] for the media files and the application version [p. 51] can be overridden with command line options [p. 198] . Furthermore you can achieve internationalization and powerful customizations with compiler variables [p. 17] . As a last resort, since the file format of install4j's config files is xml-based, you can achieve arbitrary customizations by replacing tokens (see the ant integration help page [p. 201] for an example) or by applying XSLT stylesheets to the config file.

B.11.2 Options for the install4j Command Line Compiler

The install4j command line compiler [p. 197] has the following options:

- **-h or --help**

  Displays a quick help for all available options.
- **-V or --version**

  Displays the version of install4j in the following format:

  ```
  install4j version 2.0, built on 2003-10-15
  ```
- **-v or --verbose**

  Enables verbose mode. In verbose mode, install4j prints out information about internal processes. If you experience problems with install4j, please make sure to include the verbose terminal output with your bug report.
- **-q or --quiet**

  Enables quiet mode. In quiet mode, no terminal output short of a fatal error will be printed.
- **-t or --test**

  Enables test mode. In test mode, no media files will be generated in the media file directory.
- **-g or --debug**

  Create additional debug installers for each media file. For each built media file, a directory that is named like the media file will be created in the media file output directory.
- **-n or --faster**

  Disable LZMA and Pack200 compression. If you have enabled LZMA or Pack200 compression on the media file options tab [p. 56] , this allows you to create development builds much faster, since LZMA and Pack200 are expensive compression algorithms.
- **-L or --license=KEY**

  Update the license key on the command line. This is useful if you have installed install4j on a headless system and cannot start the GUI. `KEY` must be replaced with your license key.
- **-r STRING or --release=STRING**

  Override the application version defined in the General Settings step [p. 51] . `STRING` must be replaced with the desired version number. The version number may only contain numbers and dots.
- **-d STRING or --destination=STRING**

  Override the output directory for the generated media files. `STRING` must be replaced with the desired directory. If the directory contains spaces, you must enclose STRING in quotation marks.
- **-s or --build-selected**

  Only build the media files which have been selected in the install4j IDE. By default, all media files are built regardless of the selection in the Build step [p. 193] .
- **-b LIST or --build-ids=LIST**

  Only build the media files with the specified IDs. `LIST` must be replaced with a comma separated list of numeric IDs. The IDs for media files can be shown in the install4j IDE by choosing *Media->Show media file IDs* from the main menu when the media file step [p. 177] is visible. Examples would be:

  ```
  -b 2,5,9
  --build-ids=2,5,9
  ```

- **-m or --media-types=T[,T]**

  Only build media files of the specified type. `T` must be replaced with a media file type recognized by install4j. To see the list of supported media types, execute `install4jc --list-media-types`. Examples would be:

  ```
  -m win32,macos,macosFolder
  --media-types=win32,macos,macosFolder
  ```

- **-D NAME=VALUE[,NAME=VALUE]**

  Override a compiler variable [p. 17] with a different value. You can override multiple variables by specifying a comma separated list of name value pairs. `NAME` must be the name of a variable that has been defined on the Compiler Variables tab [p. 58] of the General Settings step [p. 50] . The value can be empty.

  To override a variable for a specific media file definition only, you can prefix `NAME` with `ID:` to specify the ID of the media file. The IDs for media files can be shown in the install4j IDE by choosing *Media->Show media file IDs* from the main menu when the media file step [p. 177] is visible.
  Examples would be:

  ```
  -D MYVARIABLE=15,OTHERVARIABLE=
  "-D MYVARIABLE=15,OTHERVARIABLE=test,8:MEDIASETTITLE=my title"
  ```

  A special system variable that you can override from the command line is `LANGUAGE_ID`. `LANGUAGE_ID` must be set to the ISO code of the language displayed in the Language selection dialog [p. 60] and determines the principal installer language [p. 54] for the project or the media file.

- **-f or --var-file**

  Load variable definitions from a file. This option can be used together with the `-D` option, which takes precedence if a variable occurs twice. The file can contain

  - **variable definitions**

    One variable definition per line of the form `NAME=VALUE`.

  - **blank lines**

    blank lines will be ignored.

  - **comments**

    lines that start with # will be ignored.

  The file is assumed to be encoded in the UTF-8 format. Should you require a different encoding you can prefix the filename with `CHARSET:`, where CHARSET is replaced with the name of the encoding.

  Instead of a single variable file you can also specify a list of files separated by semicolons. The optional charset prefix must be specified for each file separately.
  Examples would be:

  ```
  -f varfile.txt
  --var-file=ISO-8859-3:varfile.txt
  --var-file=one.txt;two.txt
  --var-file=ISO-8859-3:one.txt;ISO-8859-1:two.txt
  ```

- **-M or --list-media-types**

Prints out a lists of supported media types for the `--media-types` option and quits.

B.11.3 Using install4j with ant

Integrating install4j with your ant script (read about ant at ant.apache.org) is easy. Just use the `install4j` task that is provided in *`$INSTALL4J_HOME/bin/ant.jar`* and set the `projectfile` parameter to the install4j project file that you want to build.

To make the `install4j` task available to ant, you must first insert a `taskdef` element that tells ant where to find the task definition. Here is an example of using the task in an ant build file:

```
<taskdef name="install4j"
classname="com.install4j.Install4JTask"
classpath="C:\Program Files\install4j\bin\ant.jar"/>

<target name="media">
<install4j projectfile="myapp.install4j"/>
</target>
```

The `taskdef` definition must occur only once per ant-build file and can appear anywhere on the top level below the `project` element.

Note: it is **not possible** to copy the *`ant.jar`* archive to the *`lib`* folder of your ant distribution. You have to reference a full installation of install4j in the task definition.

The `install4j` task supports the following parameters:

Attribute	Description	Required
projectfile	The install4j project file that should be build.	Yes
verbose	Corresponds to the `--verbose` command line option [p. 198] . Either `true` or `false`.	No, verbose and quiet cannot **both** be `true`
quiet	Corresponds to the `--quiet` command line option [p. 198] . Either `true` or `false`.	
test	Corresponds to the `--test` command line option [p. 198] . Either `true` or `false`.	No
debug	Corresponds to the `--debug` command line option [p. 198] . Either `true` or `false`.	No
faster	Corresponds to the `--faster` command line option [p. 198] . Either `true` or `false`.	No
release	Corresponds to the `--release` command line option [p. 198] . Enter a version number like "`3.1.2`". The version number may only contain numbers and dots.	No
destination	Corresponds to the `--destination` command line option [p. 198] . Enter a directory where the generated media files should be placed.	No
buildselected	Corresponds to the `--build-selected` command line option [p. 198] . Either `true` or `false`.	No

buildids	Corresponds to the `--build-ids` command line option [p. 198] . Enter a list of media file ids. The IDs for media files can be shown in the install4j IDE by choosing *Media->Show media file IDs* from the main menu when the media file step [p. 177] is visible.	No
mediatypes	Corresponds to the `--media-types` command line option [p. 198] . Enter a list of media types. To see the list of supported media types, execute `install4jc --list-media-types`.	No
variablefile	Corresponds to the `--var-file` command line option [p. 198] . Enter the name of file with variable definitions.	No

Contained elements:

The `install4j` task can contain `variable` elements. These elements override compiler variables [p. 17] in the project and correspond to the `-D` command line option [p. 198] . Definitions with `variable` elements take precedence before definitions in the variable file referenced by the `variablefile` parameter.

The `variable` element supports the following parameters:

Attribute	Description	Required
name	The name of the variable. This must be the name of a variable that has been defined on the Compiler Variables tab [p. 58] of the General Settings step [p. 50] .	Yes
value	The value for the variable. The value may be empty.	Yes
mediafileid	The ID of the media file for which the variable should be overridden. The IDs for media files can be shown in the install4j IDE by choosing *Media->Show media file IDs* from the main menu when the media file step [p. 177] is visible.	No

Example:

```
<install4j projectfile="myapp.install4j">
<variable name="MYVARIABLE" value="15"/>
<variable name="OTHERVARIABLE" value="test" mediafileid="8"/>
</install4j>
```

B.11.4 Using Relative Resource Paths

If you would like to use relative paths for splash screen images, icon files, directories and other external resources, (e.g. for automated build processes in distributed environments) you can choose "`make all paths relative when saving project file`" on the Project Options tab [p. 59] of the General Settings step [p. 50] .

Note: relative paths are always interpreted relative to the location of the project file.