Automation for the people: Wielding wizard-based installers

Generating GUI installers using IzPack

Skill Level: Introductory

Paul Duvall (paul.duvall@stelligent.com)
CTO
Stelligent

25 Nov 2008

Installing software is often a painful chore for most users. The installation package you generate — the "last mile" of software development — can make the difference between user adoption and another product thrown into the virtual waste bin. In this installment of *Automation for the people*, automation expert Paul Duvall demonstrates how the freely available, open source IzPack tool for writing wizard-based installers can make installing your software a snap for users.

Most of my career, I've been involved in full life-cycle software development — not just requirements elicitation, design, development, and testing, but also activities such as deployment, build management, documentation, and installation. These days, with Agile adoption increasing, this range is probably more the norm. However, my experience on Agile projects has demonstrated that effective deployments and installations are not always treated as first-class citizens. This is ironic, because if potential users can't get your software to work easily or at all, you've lost them. Providing a simple way to install your software is vitally important to attracting and keeping users.

About this series
As developers, we work to automate processes for users; yet, many of us overlook opportunities to automate our own development processes. To that end, *Automation for the people* is a series of articles dedicated to exploring the practical uses of automating software development processes and teaching you *when* and *how* to apply automation successfully.

I've used a number of installer tools over the years. For a large project my team began working on earlier this year, we had to meet several highly specific requirements for creating enterprise-grade installers. We looked at Antigen, AntInstaller, Denova, install4j, InstallAnywhere, IzPack, NSIS, and some others. Based on our particular requirements, we decided to use IzPack for the following reasons:

- It works on multiple platforms. We needed to support Windows®, Linux®, and Macintosh.
- IzPack uses the Java™ language, in which our team has extensive experience.
- It can execute Apache Ant scripts. We had invested a lot of time in writing Ant scripts for software deployment.
- IzPack is freely available for download.

IzPack has been available since 2001. It provides a rich feature set for creating wizard-based installers. In this article, I demonstrate how to create installers using this tool. I provide examples of defining panels, scripting validators, setting resources, and more.

Downloading and installing IzPack

IzPack documentation
One of the first subfolders to look through is the doc folder. It contains HTML, PDF, and Javadoc versions of the documentation, which could become your constant companion as you navigate through how to script in IzPack.

Downloading and installing IzPack is quite simple. Perhaps not surprisingly, IzPack uses IzPack to install IzPack on your workstation. Visit the IzPack Web site to download the IzPack JAR file (see Resources).

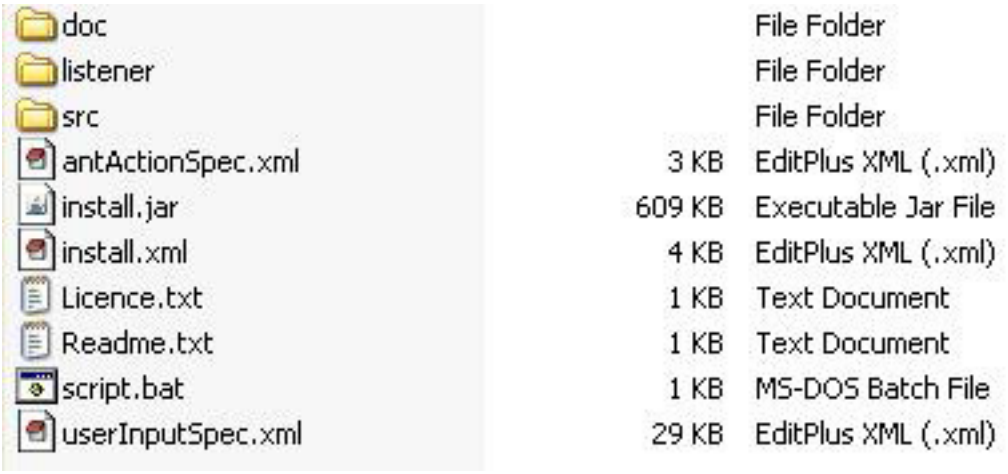
You must be running a Java Runtime Environment (JRE) to install IzPack. Open a command prompt and type `java -jar IzPack-install-4.1.0.jar`, modifying the version as necessary.

A wizard-based installer will request some basic installation information, such as where you want to install IzPack. Once IzPack is installed, it's easy to get a sample installer up and running.

Modifying sample scripts

IzPack provides a complete set of sample installation scripts. Using them as a basis for writing your own installer is the quickest approach to getting a working installer. The root directory where IzPack was installed has several subdirectories, including (among others) bin, doc, and lib. The sample installer is in the subfolder called sample, which contains virtually all you need for developing your own installers. I chose to make a copy of this sample directory so that I could modify it without disrupting its original contents. Figure 1 shows the sample directory's contents:

Figure 1. File listing of IzPack files



Here's a description of each file listed in Figure 1:

- **antActionSpec.xml**: The file that executes build-related Ant scripts.
- **install.jar**: The installation JAR file generated during IzPack's compilation. This is the installer file that users run.
- **install.xml**: The main installation script for IzPack. All resources used in an IzPack installer begin with this script.
- **Licence.txt**: A license file for your installer.
- **Readme.txt**: A readme file for the software users are installing.
- **userInputSpec.xml**: An IzPack-specific XML script for defining behavior (validation, default values, field size, and so on) when a user enters information into an installer panel.

Next, you'll look into IzPack in more detail by examining the install.xml script.

Resources

Resources define the different scripts, images, licenses, and other files that make up

the installer that I am constructing. I define a `<resources>` XML element within the install.xml script. Under the `<resources>` element, I can define multiple files that my installer will use, as shown in Listing 1:

Listing 1. Defining resource files in IzPack install.xml

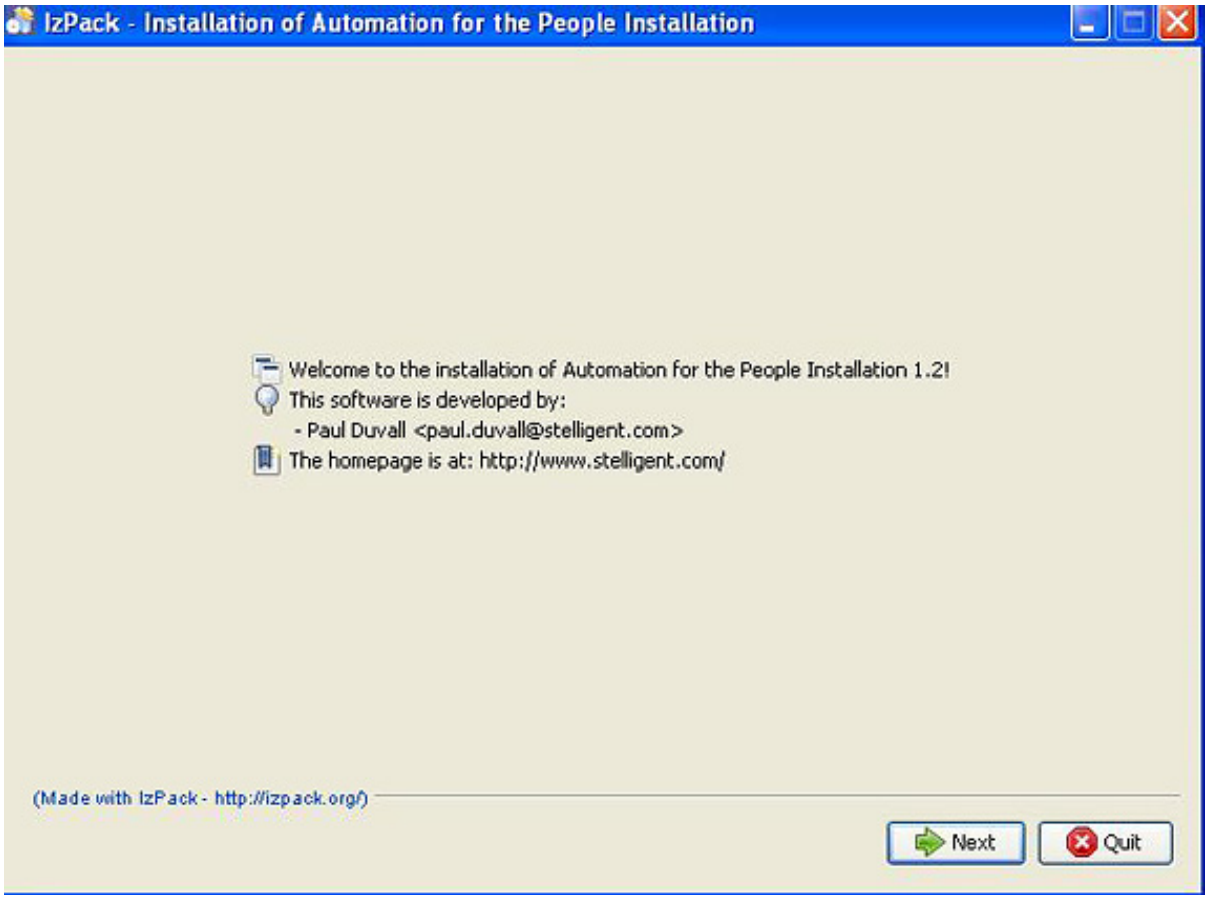
```
<resources>
  <res id="LicencePanel.licence" src="Licence.txt"/>
  <res id="InfoPanel.info" src="Readme.txt"/>
  <res id="AntActionsSpec.xml" src="antActionSpec.xml" />
  <res id="userInputSpec.xml" src="userInputSpec.xml" />
</resources>
```

You can think of IzPack's resources as a type of "bill of materials" for your installers where all files for the installer are defined.

Panels

Panels are what the users see in each step in the installation wizard. IzPack provides many types of panels right out of the box, and you can customize any of them based on your needs. In Figure 2, I've customized IzPack's `HelloPanel` to provide some introductory information to users:

Figure 2. A custom wizard-based GUI installation provided by IzPack



The standard panels include `LicensePanel`, `UserInputPanel` and `PacksPanel`, among several others. Within the install.xml file, you define the panels you want to show by using the `<panels>` element, followed by the panels templates you'll use when scripting your installer. The example in Listing 2 demonstrates how to define the panel templates:

Listing 2. Identifying the panels to be shown in the installer

```
<panels>
  <panel classname="HelloPanel"/>
  <panel classname="InfoPanel"/>
  <panel classname="LicensePanel"/>
  <panel classname="UserInputPanel" id="UserInputPanel.0" />
  <panel classname="TargetPanel"/>
  <panel classname="PacksPanel"/>
  <panel classname="InstallPanel"/>
  <panel classname="FinishPanel"/>
</panels>
```

More than likely, the panel type you will most often use is the `UserInputPanel`. It's the panel template that you customize for users to enter any variable information. This might include the user's contact information, authentication credentials, directory locations, and so on. Users may need to provide several panels of input

based on their particular environment. Because our team needed users to connect to a database and set up multiple JBoss containers, we used panels to prompt users for site-specific information.

Listing 3 is an example taken from userInputSpec.xml, which I defined as a resource in Listing 1. In this example, I'm collecting user-specific information for connection to a particular database.

Listing 3. Defining one panel's attributes in an installer

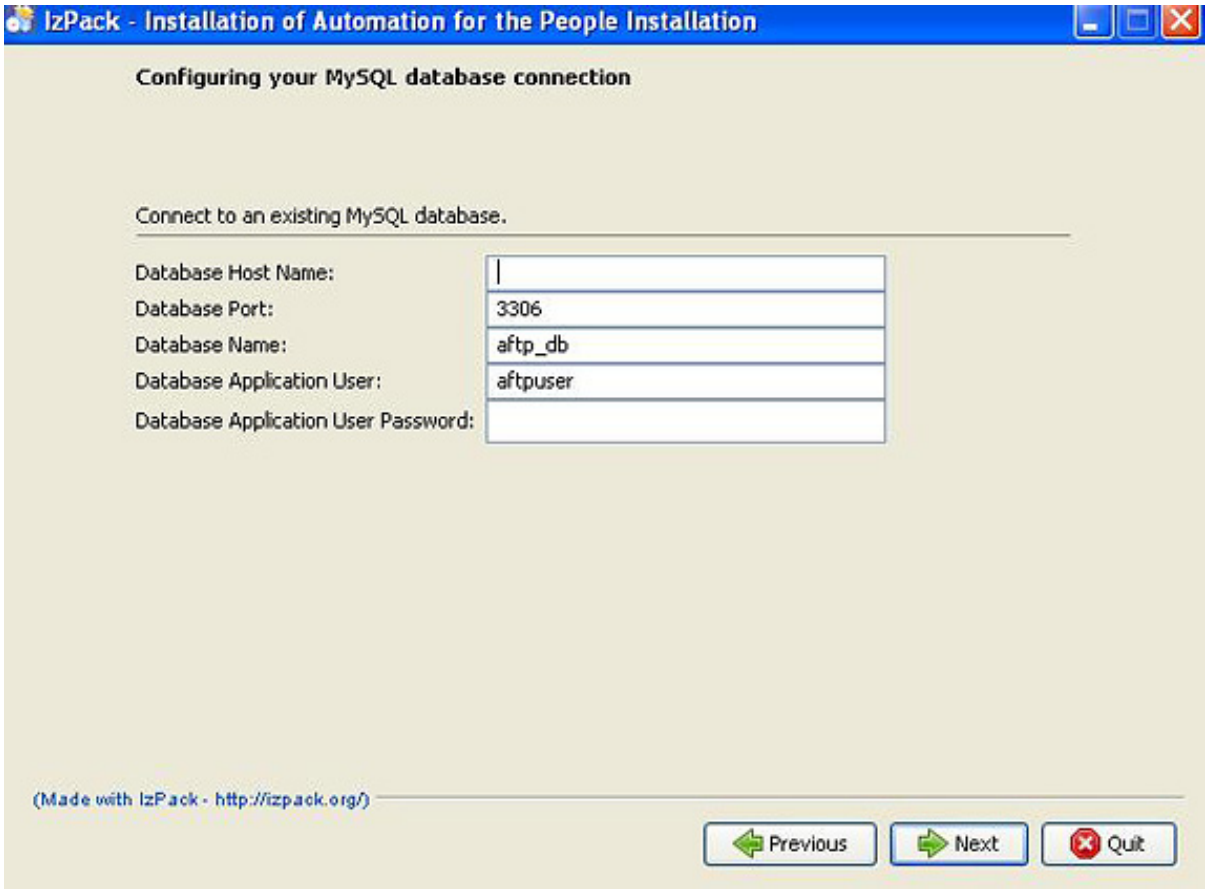
```
<panel order="0">
<field type="title" txt="Configuring your database connection" bold="true" size="1" />
<field type="staticText" align="left" txt="Connect to an existing database..."/>
<field type="divider" align="top"/>

<field type="text" variable="database.hostname">
  <spec txt="Database Host Name:" id="databasehostname.label" size="40" set=""/>
  <validator class="com.izforge.izpack.util.NotEmptyValidator"
      txt=" Database Hostname is a required field" />
</field>
...
</panel>
```

The panel's `order` attribute is given the value `0`, which corresponds to the number of the `UserInputPanel` defined in Listing 2's collection of panels.

In the user-input panel, I can define informational text for users. Furthermore, I've included a `NotEmptyValidator`, which requires a user to enter a value in the field. This prevents users from forgetting to enter required information, which would cause installation errors. The `UserInputPanel` based on Listing 3 is shown in Figure 3:

Figure 3. Panel for user's input



Users will often judge your installer's effectiveness based on how easy it is to enter information and display useful messages. Be sure to take extra care to make what they see easy to use.

Packs

IzPack uses the term *pack* to refer to the component that does all heavy lifting of actually installing the software the development team has implemented. All of the other IzPack components (panels, user input, validators, and so on) just prep you for running these packs. On my project, we used packs to do a couple of things: download a ZIP installation distribution we had already written in Ant, and then run this installation. This approach let us reuse what we'd previously written to run from the command line. Listing 5 defines a `<pack>`:

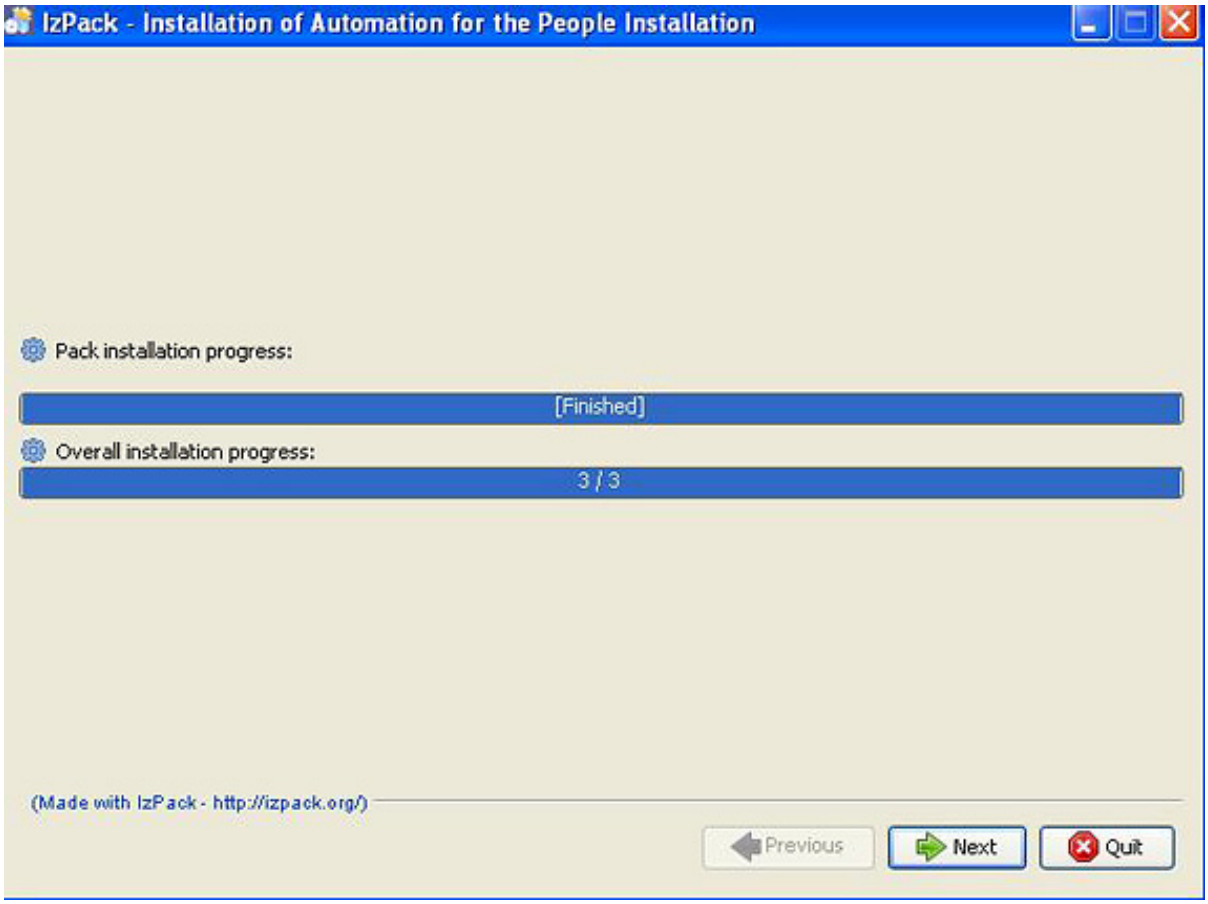
Listing 4. Defining a <pack> in install.xml

```
<packs>
  <pack name="download_install" id="download_install"
    installGroups="ap" required="no">
    <description>The base files</description>
    <file src="autopeople.zip.file"
      targetdir="$SYSTEM_user_home/@{installer.dir}"/>
    <file src="build.xml"
      targetdir="$SYSTEM_user_home/@{installer.dir}"/>
```

```
    <file src="property-template"
      targetdir="$SYSTEM_user_home/@{installer.dir}">
      <excludes>**/.svn/**</excludes>
    </file>
  </pack>
```

Figure 4 shows a pack installation in progress:

Figure 4. Executing a pack



Packs put the *Pack* in IzPack! If you've done the upfront work of user validation, diagnostics, and acquiring environment-specific information, running your installation packages should be a breeze for users.

Running Ant scripts

In my team's case, we had invested a lot of time in creating distribution-based installers using Ant. We didn't want to implement all of this functionality again in IzPack. Fortunately, IzPack provides support for calling existing Ant files. Recall that when I defined resources in Listing 1, I listed antActionSpec.xml as one of the resources. Listing 5 shows a snippet from the antActionSpec.xml script:

Listing 5. Executing pack behavior in antActionSpec.xml

```
<antactions>
  <pack name="download_install">
    <antcall buildfile="$SYSTEM_user_home/${installer.dir}/build.xml"
          order="afterpack"
          verbose="yes"
          logfile="$SYSTEM_user_home/${installer.dir}/antlog_installer.txt"
          inheritall="false"
      messageid="AntAction.download-install">
      <target name="install"/>
          <property name="install.path" value="$SYSTEM_user_home/${installer.dir}"/>
        </antcall>
  </pack>
...
</antactions>
```

The most relevant part of this script is when it executes the `build.xml`. This is the existing Ant build script that performs the installation process of downloading and extracting an installation ZIP file, installing and configuring Web containers, and the rest of the installation. antActionSpec.xml gives me a way to reuse my existing Ant scripts.

Compiling installers

The last step is known as IzPack's *compilation*. Once you've written an install.xml and its associated scripts, you're ready to generate the installer. Listing 6 is an example of the single-line command you can use to generate the install.jar (you can modify this file's name):

Listing 6. Creating an installer

```
compile ../sample/install.xml -b ../sample
```

The command in Listing 6 assumes you are running it from IzPack's bin subdirectory. `sample` is a reference to the sample subdirectory provided with IzPack. Once the installer is generated, you can test it by running `java -jar install.jar` from the location where install.jar was generated in the sample subdirectory.

Installers for the people

In this article, I've shown how to use the different components of IzPack to create a simple-to-use installation package for users. They might be users installing client-based software, users at remote sites needing to install and configure multiple servers, or perhaps an operations team installing and configuring an enterprise suite of tools. When your software is easy to install, the word gets around, especially in

complex environments where difficult installations are notorious. When installations require numerous manual steps or are virtually impossible, you can lose users quickly through exasperation or a lack of confidence in the software being delivered. Making installations easier through a tool like IzPack can help you acquire and retain a passionate user base.

Resources

Learn

- IzPack: Access IzPack resources at the project Web site, including the complete list of best practices that IzPack's developers recommend.
- "Deployment: Easy Installing with IzPack" (R. J. Lorimer, javalobby.com, November 2004): An introduction to and demonstration of some of IzPack's features.
- Open Source Installers Generators in Java (Java-Source.net): A roundup of open source Java installer generators.
- Browse the technology bookstore for books on these and other technical topics.
- developerWorks Java technology zone: Hundreds of articles about every aspect of Java programming.

Get products and technologies

- IzPack: Download IzPack to begin creating cross-platform installers.
- Ant: Download Ant and start building software in a predictable and repeatable manner.

Discuss

- Improve Your Code Quality discussion forum: Regular developerWorks contributor Andrew Glover brings his considerable expertise as a consultant focused on improving code quality to this moderated discussion forum.
- Accelerate development space: Regular developerWorks contributor Andrew Glover hosts a one-stop portal for all things related to developer testing, Continuous Integration, code metrics, and refactoring.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Paul Duvall
Paul Duvall is the CTO of Stelligent, a consulting firm that helps development teams deliver production-ready software. He is the co-author of the Addison-Wesley Signature Series book, *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley Professional, 2007; Jolt Award 2008 winner). He also contributed to the *UML 2 Toolkit* (Wiley, 2003) and the *No Fluff Just Stuff Anthology* (Pragmatic Programmers, 2007).

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.