Custom Panels

Creating Your Own Panels

In IzPack all of the actual work of installing an application is done in panels. The IzPack framework is primarily there to support the operation of the panels and to manage the navigation through the installation process. This enables a user to decide which operations are carried out during an installation and the order in which they are carried out, simply by listing the appropriate panels in the desired order.

As far as extending the functionality of IzPack is concerned, the result of this design is that new functionality can be integrated by adding new panels to the framework and then listing them in the install spec.

How to get started

To get started with writing your own panels, it is best to place all the IzPack code into a separate working directory, from where you can compile it. Then try to compile the code as is and get that to work.

The next step would be to have a look at another panel implementation, so you can see how things are done. Make sure you look at the less complicated panels, as the panels with advanced features will only be confusing. All the code for building UI and such, only detracts from the essentials of what a panel needs to do. This means that you shouldn't start with `UserInputPanel` or`ShortcutPanel`. `HelloPanel` is probably a much better choice at this stage. The source code for panels is located at: `/src/lib/com/izforge/izpack/panels`.

You will find that all panels are derived from `IzPanel`; do the same thing with your new panel. Please note that the `IzPanel` class itself is located in the `com.izforge.izpack.installer` package but your panels need to belong to `com.izforge.izpack.panels`. Perhaps you can just copy the code of a panel, remove all the functional stuff and then start filling in your own code. Start with something very simple to begin with, just to see how it works. The implementation is really quite straight forward.

Next Steps

Once you have a successful compilation, you must place the compiled result of your panel code at a special place, so that the installer compiler can fetch it when building an installer that uses your panel. Go to: `/bin/panels`

You will see that there is a subdirectory for each panel. Make a subdirectory for your new panel with the exact same name as your panel and place your compiled panel code there.

Once this is accomplished, you are ready to use your panel in an installer. Just list it in the spec file like any other panel, compile and in theory it will show up when running the installer. Once you made it this far, you can dig deeper and get going with your specific needs.

Oh, and one other thing: If you think the your code might be useful for a larger audience, please think about a contribution to IzPack.

Access to the Variable Substitution System

One thing many developers ask about is how to get access to the variable substitution system. This is not surprising, because customizing an installation for a particular target environment is one of the most important functions of an installer and the variable substitution system plays a big part in this operation.

You can get access to the variable substitution system through the protected variable `idata` in `IzPanel`. This variable is of the type `InstallData`, which is in turn subclassed from`AutomatedInstallData`. The Javadoc documentation will give you more details on these classes. Of particular interest in this context are the methods `getVariable()`, `setVariable()` and`getVariableValueMap()` in `AutomatedInstallData`.

Controlling Flow

Some of the interesting methods in `com.izforge.izpack.InstallerFrame` that you might want to explore are `lockPrevButton()` and `lockNextButton()`. They allow you to block the use of the button to move back to the previous panel and the button that moves to the next panel respectively. Being able to control the availability of these buttons to the user is important if one of your panels performs a task where the effects cannot be undone. If the user would navigate back to the previous panel your installation might get into an unknown or even unstable state. On the other hand, if the operations in one panel vitally depend that a task in the previous panel is completed, then you should block the use of the next button until that task is completed.

Reading XML

If you need configuration files for your panel you would want to use XML for that purpose. To read XML files you should use NanoXML, as it is

guaranteed to be available at installation time. In fact, all of the IzPack infrastructure uses NanoXML to read XML files. First you should read the NanoXML documentation. The documentation is included as PDF file with the IzPack distribution, have a look in `/doc/nanoxml`. In addition to that, the Javadoc-generated class documentation is an excellent resource to get help on NanoXML. And then, there is always the code of other panels to see practical examples. Generally, it is a much simpler matter to use NanoXML then to use the DOM included with the Java distribution, so don't hesitate to explore NanoXML.

Supporting Classes

If your panel requires any supporting classes that are part of the panels package, then you must place the .class files into the same directory with your panel .class file.

It is also possible to have supporting classes that are not part of the panels package. In fact, these classes don't even have to be in the `com.izpack...` tree. You simply have to ensure that the .class files are located in the proper directory structure inside `/lib/installer.jar`. If this is done, they will be available at install time. For your first experiments you can simply compile your classes and add them to the .jar file using the jar tool or a zip utility. However, ultimately it is much easier to use Ant and the IzPack build script to accomplish this task.

Panel UI Definition.

Layout

Field Types

Hidden and Read-only fields.

Panels that are not visible

Because the existing panels all have a visible GUI and because the term panel hints at something visible, it is not obvious that a panel does not have to contain any visible GUI to function in this framework.

If you have a task that needs to be performed at a particular step in the installation process, but that does not need any user interaction, you can implement a panel that is not visible. To implement this, you should first familiarize yourself with the Javadoc documentation of `com.izforge.izpack.InstallerFrame`. In your panel code you get access to the right instance of `InstallerFrame`through the protected variable `parent` in `IzFrame`.

To begin with, do not configure any UI. In other words, do not set a layout and do not place any GUI elements on your panel. In this context the method `skipPanel()` is what gets the job done. In your `panelActivate()` method you simply perform your task and then call `parent.skipPanel()`. This gets the job done without the user being aware that there was another panel in the flow.

A word about building IzPack

IzPack uses Maven to do its builds. You should not have to make any changes to the Maven POM but you should review it to be sure that you have included all of the dependencies that you use.

The `IzPanel` Class

? Unknown Attachment

The two data members are : the install data (refer to the `InstallData` Javadoc reference) and a reference to the parent installer frame.

The methods have the following functionality :

(constructor)* : called just after the language selection dialog. All the panels are constructed at this time and then the installer is shown. So be aware of the fact that the installer window is*not yet visible when the panel is created. If you need to do some work when the window is created, it is in most cases better do it in `panelActivate`.

- `isValidated` returns `true` if the user is allowed to go a step further in the installation process. Returning `false` will lock it. For instance the LicencePanel returns `true` only if the user has agreed with the license agreement. The default is to return `true`.
- `panelActivate` is called when the panel becomes active. This is the best place for most initialization tasks. The default is to do nothing.
- `makeXMLData` is called to build the automated installer data. The default is to do nothing. `panelRoot` refers to the node in the XML tree where you can save your data. Each panel is given a node. You can organize it as you want with the markups you want starting from `panelRoot`. It's that simple.
- `runAutomated` is called by an automated-mode installation. Each panel is called and can do its job by picking the data collected during

a previous installation as saved in `panelRoot` by`makeXMLData`.

- `setInitialFocus` with this method it is possible to set a hint which component should be get the focus at activation of the panel. It is only a hint. Not all components are supported. For more information see java.awt.Component.requestFocusInWindow or java.awt.Component.requestFocus if the VM version is less than 1.4.
- `getInitialFocus` returns the component which should be get the focos at activation of the panel. If no component was set, null returns.
- `getSummaryBody` this method will be called from the SummaryPanel to get the summary of this class which should be placed in the SummaryPanel. The returned text should not contain a caption of this item. The caption will be requested from the method getCaption. If null returns, no summary for this panel will be enerated. Default behaviour is to return null.
- `getSummaryCaption` this method will be called from the SummaryPanel to get the caption for this class which should be placed in the SummaryPanel. Default behaviour is to return the string given by langpack for the key ClassName.summaryCaption.

Additionally, there are some helper methods to simplify grid bag layout handling and creation of some common used components.

The `Internationalization` of custom panels

A common way to define language dependant messages for custom panels is to define a resource for custom langpacks.

```
<resources>
    ...
    <res id="CustomLangpack.xml_deu"
        src="myConfigSubPath/CustomLangpack_deu.xml"/>
    ...
</resources>
```

This appears in the install.xml file and defines a German language pack for a custom panel.

The id should be written as shown, the src file sub path and name can be other than in the example. This file should be using the same DTD as the common langpack.

For each language, a separate file with the ISO3 extension in the id should be used and the src should point to the corresponding language file.