

Desktop Shortcuts

Desktop Shortcuts

(by Elmar GROM and Marc EPPELMANN)

Defining Shortcuts

Introduction

On today's GUI oriented operating systems, users are used to launching applications, view web sites, look at documentation and perform a variety of other tasks, by simply clicking on an icon on the desktop or in a menu system located on the desktop. Depending on the operating system these icons have different names. In this context we will refer to them collectively as shortcuts.

Apart from actually placing an application on the target system, users routinely expect an installer to create the necessary shortcuts for the application as well. For you as application developer, this means that for a professional appearance of your product you should also consider creating shortcuts.

In contrast to the general specification of an IzPack installer, the specification of shortcuts in IzPack requires a little more effort. In addition, some of the concepts are a bit more complex and there are some operating system specific issues to observe. Fortunately, you only need to worry about operating system specifics if you want to deploy your application to multiple different operating systems. In any case, it will pay off to spend some time to study this documentation and the example spec files before you start to implement your own shortcuts.

At the time of writing this Chapter the current IzPack Version 3.7.0-M3 is only capable to creating shortcuts on

1. Microsoft Windows, and
2. Unix and Unix-based operating systems (like Linux), which use the `X11` GUI-System and `FreeDesktop.org` based shortcut handling (such as `KDE` and `Gnome`).

Other operating or GUI systems, such as MacOS X are not supported. However, there is a special UI-variant that automatically pops up on unsupported systems. It informs the user about the intended targets of your shortcuts and allows the user to save this information to a text file. While this is not an elegant solution, at least it aids the user in the manual creation of the shortcuts.

If you would like to review what an end user would see if the target operating system is not supported, you can do the following. Simply place the tag `<notSupported/>` in the spec file. This tag requires no attributes or other data. It must be placed under `<shortcuts>`, just like the individual shortcut specifications. Be sure to remove this tag before getting your application ready for shipment.

We expect other operating systems to be supported in the near future and as always, contributions are very welcome. At present someone is actively working on Mac support.

What to Add to the Installer

There are some things that you have to add to your installer to enable shortcut creation. Obviously you need to add the panel responsible for creating shortcuts. This panel is aptly enough called `ShortcutPanel`. However, in order for the `ShortcutPanel` to work properly a number of additional items are required. These must be added manually to the installer, as all other resources, since the front-end will be rewritten. In this chapter we will explain which of these items are required and for what reason.

First, we would like to discuss items that are supplied with IzPack and only need to be added to the installer. After that, we move on to the things you have to prepare yourself before you can add them. The way in which shortcuts are created varies widely among operating systems. In some cases it is possible to do this with pure Java code, while other systems such as MS Windows require native code to accomplish this task. On the other side, the current implementation, which creates shortcuts on Unix based systems needs no native library at all, since it works with 'these' pure Java code. The native library required for the Windows operating systems are supplied with IzPack is called `ShellLink.dll`. Note: They will not be automatically added to your installer file. You need to list them yourself in the XML file for the installer. A description how to do this follows in the next section.

Native libraries can be added to the installer by using the `<native>` tag. To add the `ShellLink.dll`, you just have to add the following line to the installer XML file: `file: <native type="izpack" name="ShellLink.dll" />` For more details about the use of the `<native>` tag see the chapter about the format of the XML file.

You have also to add an extra specification file for each platform family to enable shortcut creation on these platforms. At least one (the default file) is required by the shortcut panel. The format of all spec files is XML and they must be added to the installer as a resource. The source name of this specification does not matter, however its resource name (also called id or alias) when added to the installer must be `(prefix)+shortcutSpec.xml`. At this release, there are only two prefixes supported: "Win" for the Windows family and "Unix" for all Unixes. If the prefix is omitted the shortcut panel searches for a named resource: `shortcutSpec.xml`. This is the default resource name. As the default resource name will be used on Windows platforms, the `"Win_shortcutSpec.xml"` can be omitted. Hint: If the shortcut panel does not find one of these named resources, it will never appears. So, do not use different resource names and do not add a path to these.

ShortcutSpecs can be localized, see "Localizing shortcuts" below.

Example

```
<res
src="C:\MyDocuments\Installer\default_shortcut_specification.xml"
  id="shortcutSpec.xml" />
<res
src="C:\MyDocuments\Installer\unix_shortcut_specification.xml"
  id="Unix_shortcutSpec.xml" />
```

Why use different shortcut spec files?

1. The Target filenames are most different.(batch files on Windows vs. shell scripts on Unix.)
2. The Icon file formats are different. ICOs on Windows-, PNGs on Unix-platforms.
3. The Target locations can be different.
4. If your installer is localized, your users sure will expect their shortcuts names and descriptions to be as well. Moreover, some paths can be different when OS are localized (`My Documents` becomes `Mes Documents` on Windows in french for example). See "Localizing shortcuts" below.

This is the simple reason.

Why Native Code to do the Job on Windows?

by Elmar

This little chapter is not strictly part of the documentation but i have been asked this question sufficiently often that i think it's worth explaining right here. It is certainly a natural question to ask. After all IzPack is an application completely written in Java and primarily targeted for the installation of Java based programs. So why wouldn't we try to keep everything pure Java and avoid the use of native code altogether? There must be some personal preference of the developer hidden behind this approach you might think. Well, not really, but i admit at first it seems quite feasible to write it all in Java. On virtually any operating system or GUI surface around, Shortcuts are simply files on the local file system. Files can be created and accessed directly from within Java, so why should there be a need for using native code?

Well, it turns out that just creating a file is not good enough, it also needs to have the right content. Shell Links as they are called in Windows land are binary files. i actually managed to find documentation on the format. Naturally this was hacker data, you won't get this sort of thing from Microsoft (by the way: thanks a lot to Jesse Hager for a smash job!). Armed with this information i tried to create these files myself in Java. The problem was that the documentation was not entirely accurate and had some gaps as well. i tried for over a month to get this to work but finally i had to give up. Even if i would have succeeded, it would have been a hack, since a shell link requires some information that is impossible to obtain from within Java. Usually you can successfully create a shell link by only filling in the bare minimum information and then ask Windows to resolve the link. Windows then repairs the shell link. Unfortunately this was only the beginning, soon i encountered a host of other problems. For one thing, the installer needs to know the correct directories for placing the links and it turns out they are named differently in different countries. In addition, there are ways of manually modifying them, which some people might actually have done. The only way to place the shortcut files reliably is through accessing the Windows Registry. Naturally, this operation also required native code. Same thing with asking Windows to resolve the link... On the bottom line, at every step and turn you run into an issue where you just need to use native code to do the trick. So i decided that i would do it the proper way all the way through. That is in a nutshell the reason why i used native code to create shortcuts on MS-Windows.

As i am writing this i am at work with a friend to replicate this work for the Mac and it looks very much like we need to take the same approach there as well. On the various Unix GUIs on the other hand, we are lucky that we can do the job without native libraries.

The Shortcut Specification

As we say above, the specification for shortcuts is provided to the ShortcutPanel in the XML fileformat. At the time of this writing (for IzPack version 3.7.0-M3) the front-end will be rewritten. Until these work is in progress you have to write the specification files manually. For your convenience, there are two annotated sample specification files in the sample subdirectory of your IzPack installation. At the beginning you might want to experiment with these files.

Both specification files have one root element called `<shortcuts>`. This root elements recognizes 3 different child elements: `<programGroup>`, `<skipIfNotSupported/>`, `<defaultCurrentUser/>` and `<shortcut>`.

`<skipIfNotSupported/>` can be used to avoid the panel to show the alternative UI which shows the shortcut information that would have been created on a system that supports it. In other words, using this tag will make the panel be silent on non-supported systems. The default is to show the alternative UI.

`<defaultCurrentUser/>` may be specified to make "current user" be the default selection on the panel. If not specified then "all users" will be the default selection (if available).

The `<programGroup>` tag allows you to specify the name of the menu, or more precise, the folder in which the shortcuts will be grouped. The exact location and appearance of the program group depends on the specific target system on which the application will be installed, however you can partially control it. Please note that `<programGroup>` may only appear once in the specification. If more than one instance occurs, only the first one will be used. This tag requires two attributes: `defaultName` and `location`. `defaultName` specifies the name that the group menu should have on the target system. You should be aware that the `ShortcutPanel` will present this name to the user as a choice. The user can then edit this name or select a group that already exists. As a result, there is no guarantee that the actual name of the program group on the target system is identical with your specification. `location` specifies where the group menu should show up. There are two choices: `applications` and `startMenu`. If you use `applications`, then the menu will be placed in the menu that is ordinarily used for application shortcuts. `applications` is recommended for Unix shortcuts. If you use `startMenu`, the group menu will be placed at the top most menu level available on the target system. Depending on the target system, it might not be possible to honor this specification exactly. In such cases, the `ShortcutPanel` will map the choice to the location that most closely resembles your choice. Unix shortcuts do not need to support the `startMenu`, because the `applications` menu is already on the highest level. This means this has no affect on the platform.

For each shortcut you want to create, you have to add one `<shortcut>` tag. Most details about the shortcut are listed as attributes with this tag. The following sections describe what each attribute does, which attributes are optional and which ones are required and what the values are that are accepted for each of the attributes. Note that all attributes that have a yes/no choice can also be omitted. Doing so has the same effect as using a value of no. The shortcut attributes can be divided into two groups

- attributes that describe properties of the shortcut
- attributes that define the location(s) at which a copy of the shortcut should be placed.

The following attributes are used to define location:

- `programGroup`
- `desktop`
- `applications`
- `startMenu`
- `startup`

Shortcut Attributes

There are three classes of attributes. Some are required, most are completely optional and some are semi-optional. The set of semi-optional attributes are all the attributes used to define the location of a shortcut. These are semi-optional because for any individual one it is your choice if you want to include it or not. However they are not completely optional. You must specify at least one location. If all were omitted, the instruction would essentially tell the panel that a copy of this shortcut is to be placed at no location. In other words no copy is to be placed anywhere.

name - required

The value of this attribute defines the name that the shortcut will have. This is the text that makes up the menu name if the shortcut is placed in a menu or the caption that is displayed with the shortcut if it is placed on the desktop.

target - required

The value of this attribute points to the application that should be launched when the shortcut is clicked. The value is translated through the variable substitutor. Therefore variables such as `$INSTALL_PATH` can be used to describe the location. **You should be aware that the use of this tag is likely to change once other operating systems are supported.**

commandLine - optional

The value of this attribute will be passed to the application as command line. i recommend to work without command line arguments, since these are not supported by all operating systems. As a result, your applications will not be portable if they depend on command line arguments. Instead, consider using system properties or configuration files.

workingDirectory - optional

This attribute defines the working directory for the application at the time it is launched. i would recommend some caution in relying on this too heavily if your application should be portable, since this might not be supported by all operating systems. At this time i don't have enough information to make a definite statement one way or the other. The value is translated through the variable substitutor. Therefore variables such as `$INSTALL_PATH` can be used to describe the directory.

description - optional

The value of this attribute will be visible to the user when a brief description about associated application is requested. The form of the request and the way in which this description is displayed varies between operating systems. On MS-Windows the description is shown as a tool tip when the mouse cursor hovers over the icon for a few seconds. On some operating systems this feature might not be supported but i think it is always a good idea to include a brief description.

iconFile - optional

The value of this attribute points to the file that holds the icon that should be displayed as a symbol for this shortcut. This value is also translated through the variable substitutor and consequently can contain variables such as `$INSTALL_PATH`. If this attribute is omitted, no icon will be specified for the shortcut. Usually this causes the OS to display an OS supplied default icon. **The use of this attribute is also likely to change once other operating systems are supported. Read the Section about Icons below, for more information.**

iconIndex - optional

If the file type for the icon supports multiple icons in one file, then this attribute may be used to specify the correct index for the icon. i would also advise against using this feature, because of operating system incompatibilities in this area. In file formats that do not support multiple icons, this values is ignored.

initialState - optional

There are four values accepted for this attribute: `noShow`, `normal`, `maximized` and `minimized`. If the target operating system supports this feature, then this value will have the appropriate influence on the initial window state of the application. `noShow` is particularly useful when launch scripts are used that cause a command window to open, because the command window will not be visible with this option. For instance on MS-Windows starting a batch file that launches a Java application has the less than pretty side effect that two windows show: the DOS command prompt and the Java application window. Even if the shortcut is configured to show minimized, there are buttons for both windows in the task bar. Using `noShow` will completely eliminate this effect, only the Java application window will be visible. *On Unix use `normal` , because this is not supported.*

programGroup - semi-optional

The value for this attribute can be either yes or no. Any other value will be interpreted as no. If the value is yes, then a copy of this shortcut will be placed in the group menu. *On Unix (KDE) this will always be placed on the top level.*

desktop - semi-optional

For this attribute the value should also be yes or no. If the value is yes, then a copy of the shortcut is placed on the desktop. *On Unix the shortcuts will only be placed on the (KDE-) desktop of the user, who currently runs the installer. For Gnome the user can simply copy the.desktop files from* /Desktop to /gnome-desktop. (This is already a TODO for the Unix-shortcut implementation.)*

applications - semi-optional

This is also a yes/no attribute. If the value is yes, then a copy of the shortcut is placed in the applications menu (if the target operating system supports this). This is the same location as the applications choice for the program group. *This makes no sense on Unix.*

startMenu - semi-optional

This is a yes/no attribute as well. If the value is yes, then a copy of the shortcut is placed directly in the top most menu that is available for placing application shortcuts. *This is not supported on Unix. see above.*

startup - semi-optional

This is also a yes/no attribute. If the value is yes, then a copy of the shortcut is placed in a location where all applications get automatically started at OS launch time, if this is available on the target OS. *This is also not supported on Unix.*

Unix specific shortcut attributes

This extension was programmed by MARC EPPELMANN. This is still in development and may be changed in one of the next releases of IzPack.

type - required

This must be one of `Application` or `Link`

Application: To start any application, native, Java or shell-script based, the `*type` has to be `Application`. The GUI-System will launch this Application, so as is, thru their native shell or application launcher. In this case, note that the right `workingDirectory` is always important on Unix platforms. If the users `PATH` environment variable does not contain the path, where the application is located, this can never be run, until the `workingDirectory` does not contain these path. The needed current path: ".", this is the case on most systems, should be in the users `PATH` environment variable. Consult the Unix manuals for more details.

Link: If you want to open a URL in the users default Webbrowser, you have to set the `*type` to `Link`. Note: The `url` attribute must be set to work properly.

- Other: There are more supported types on KDE, like `FSDevice`, but these types makes no sense for IzPack, in my opinion.

Without the type the Unix shortcut does not work.

url - semi-optional

If you want to create a shortcut as type `Link`, then you have to set the `url` attribute. The value can be a locally installed html or another document, with a known MIME type, like plain text, or a WWW Url i.e. 'http://izpack.org/'.

A local document can be referenced by i.e. "\$INSTALL_PATH/doc/index.html".

The IzPack variable substitution system is supported by the **url**.

encoding - required

This should always set to **UTF-8**.

terminal - optional

If you want, the user can see the console output of a program (in Java applications "System.outs"), set the `terminal` attribute to **true**.

KdeSubstUID - optional

This is the sudo option for a shortcut. If set to **true** you will be asked for the password of the following `KdeUsername`.

KdeUsername - optional

This could be the user, with the right permission. This is `root` in most cases.

createForAll - optional

If an user with administrative rights, such as `root`, performs the installation this flag allows to determine, if the shortcut should be appear to the other users or only for `root`.

Categories - optional

- This is the category where to put in the Desktop Application Menu.
Here are some Sample Categories and their apps examine the desktop files in `/usr/share/applications ...`
Categories="Application;Network;WebDevelopment;" - Nvu, Categories="Qt;Development;GUIDesigner;" - QtDesigner3, Categories="Application;System;" - VMwareServer-console,
Categories="Network;WebBrowser;" - Opera, Categories="Development;Debugger;" - DDD debugger,
Categories="Development;IDE;" - Eclipse IDE, Categories="SystemSetup;X-SuSE-Core-System;" - Yast2,
Categories="System;Archiving;" - Sesam archiving, Categories="System;Database;" - MySQL Administrator,

TryExec - optional

TryExec="aTryExecCommand" will passes raw thru and tries to execute the command.

Selective Creation of Shortcuts

Usually all shortcuts that are listed will be created when the user clicks the 'Next' button. However it is possible to control to some degree if specific shortcuts should be created or not. This is based on install conditions. By including one or more `<createForPack name=a pack name />` tags in the specification for a shortcut, you can direct the ShortcutPanel to create the shortcut only if any of the listed packs are actually installed. The 'name' attribute is used to define the name of one of the packs for which the shortcut should be created. You do not need to list all packs if a shortcut should always be created. In this case simply omit this tag altogether.

A word of caution

For any shortcut that is always created, i would recommend to omit this tag, since i have seen a number of problems related to changing pack names. You can save yourself some troubleshooting and some Aspirin by not using this feature if it's not required. On the other hand if you need it i would advise to be very careful about changing pack names.

Localizing shortcuts

The izPack installer allows localization of your installer. Obviously, you will want your shortcuts labels to be localized, too. Thus, you can provide shortcutSpec files for each language you support.

This works just like localizing other specs for other panels : add an underscore and the iso3 language code to the end of the resource name, and set the source name to the appropriate localized spec file. As for the platforms, if a specific spec file is not declared for the current language, the installer will revert to the default `shortcutSpec.xml` resource. This way, you can for example have a resource `shortcutSpec.xml_fra` for french installs and have the default resource `shortcutSpec.xml` (maybe containing strings in english) be used for other languages.

The platform prefix and language suffix are combined like in this example : `Unix_shortcutSpec.xml_fra`.

When the installer can't find such a platform-and-language-specific resource, it will fallback to a suitable default using the following search order : `shortcutSpec.xml_<iso3>` (default platform, language-specific) `<platform>_shortcutSpec.xml` (default language, platform-specific) `shortcutSpec.xml` (default platform and language)

Reminder: If the shortcut panel does not find one of these named resources, it will never appear. So, do not use different resource names and do not add a path to these.

Example

```
<res
src="C:\MyDocuments\Installer\default_shortc
ut_specification.xml"
  id="shortcutSpec.xml"/>
<res
src="C:\MyDocuments\Installer\default_shortc
ut_specification_fr.xml"
  id="shortcutSpec.xml_fra"/>
<res
src="C:\MyDocuments\Installer\unix_shortcut_
specification_en.xml"
  id="Unix_shortcutSpec.xml"/>
<res
src="C:\MyDocuments\Installer\unix_shortcut_
specification_fr.xml"
  id="Unix_shortcutSpec.xml_fra"/>
```

DesktopShortcutCheckboxEnabled Builtin Variable

`$DesktopShortcutCheckboxEnabled` : When set to true, it automatically checks the "Create Desktop Shortcuts" button. To see how to use it, go to `The Variables Element <variables>

Be careful this variable is case sensitive !

ApplicationShortcutPath Builtin Variable

`$ApplicationShortcutPath` : To define the path for the application shortcuts, absolute or relative to the `$$INSTALL_ALL_PATH`. If undefined, the application shortcuts will be directly written in the `$$INSTALL_PATH`.

From Version 4.1.1 the `xdg-desktop-icon` commandline tool will be used to copy this application shortcuts to your, or all users desktop.

Summary

Native Libraries

- ShellLink.dll - required by Microsoft Windows
- 'Nothing' - for KDE/Gnome shortcuts

Names of the Specification Files `shortcutSpec.xml` for Windows and as default. `Unix_shortcutSpec.xml` for Unix.

Specification File Layout - Windows

```
<shortcuts>
  <skipIfNotSupported/>
  <programGroup
defaultName="MyOrganization\MyApplication"

location="applications|startMenu"/>
  <shortcut
    name="Start MyApplication"

target="$INSTALL_PATH\Path\to\MyApplication\
launcher.bat"
    commandLine=""

workingDirectory="$INSTALL_PATH\Path\to\MyAp
```

```
plication"
    description="This starts MyApplication"

iconFile="$INSTALL_PATH\Path\to\MyApplication\Icons\start.ico"
    iconIndex="0"

initialState="noShow | normal | maximized | minimized"
    programGroup="yes | no"
    desktop="yes | no"
    applications="yes | no"
    startMenu="yes | no"
    startup="yes | no">

    <createForPack name="MyApplication
Binaries"/>
    <createForPack name="MyApplication
```

```
Batchfiles" />  
    </shortcut>  
</shortcuts>
```

A sample Specification File for Unix is at the end of this chapter

Shortcut Tips

I wrote this section to provide additional information about issues surrounding the creation of shortcuts. Reading this section is not necessary to successfully create shortcuts, but it might help you creating an installation that works more smoothly. In addition, it might give you some knowledge about operating systems that you don't know so well. In fact most of the issues described in this section are focused on differences in operating system specifics.

The Desktop

You should recognize that the desktop is precious real estate for many people. They like to keep it uncluttered and keep only the things there that they use on a regular basis. This is not true for everybody and you might personally think different about this. Still, the fact remains that a lot of people might have different feelings about it, so you should not automatically assume that it is ok to place all of your shortcuts on the desktop proper. While your application is certainly one of the most important things for you, for your customers it is probably one of many applications they use and maybe not even the most important one. Accordingly, placing more shortcut icons there than they feel they will use on a regular basis and especially doing this without asking for permission might trigger some bad temper.

Annotation: But even the experienced user should be able to organize their Desktop. On Linux the users desktop is the only place, which supports any kind of shortcuts.

It is common practice to create a program group in the application menu system of the OS and place all shortcuts that go with an application in that program group. In addition, only one shortcut to the key access point of the application is placed directly on the desktop. Many installers first ask for permission to do so, as does the ShortcutPanel in IzPack.

I would like to recommend that you always create a shortcut in the menu system, even if your application has only one access point and you are placing this on the desktop. Note that shortcuts can also be placed directly in the menu, they don't need to be in a program group. There are two reasons for doing so.

- If the user elects not to create shortcuts on the desktop, they will end up with no access point to your application
- Even if this works fine, occasionally people 'clean up' their desktop. They might later find that they accidentally deleted the only access point to your application. For the less technology savvy users, recreating the shortcut might be a rough experience.

Icons

Icons are supplied in image files, usually in some kind of bitmap format. Unfortunately there is no format that is universally recognized by all operating systems. If you would like to create shortcuts on a variety of operating systems that use your own icons, you must supply each icon in a number of different formats. This chapter discusses icon file formats used on various operating systems. Fortunately there are good programs available that allow you to convert between these formats, so that creating the different files is not much of a problem once the icons themselves are created.

Microsoft Windows

Windows prefers to use its native icon file format. Files of this type usually use the extension .ico. These so called ICO files can hold multiple icons in one file, which can be useful if the same icon is to be provided in a number of sizes and color-depths.

Windows itself selects the icon with the most matching dimensions and displays it. While the Start menu displays the icon with 16x16 pixel if available, the desktop displays the 32x32 pixel resolution of the same ICO if this is in.

In other words, a ICO file has embedded one or more dimensions of the same Icon. We recommend to play with `microangelo`.

Dlls and Exe files on the other side, can store, amongst other things, a collection of different Icons. You can select your desired Icon by its index. The lowest index is 0. Use the iconIndex attribute in the spec file to specify this index.

As a sample look into

```
%SystemRoot%\system32\shell32.dll
```

These contains a lot of Windows own icons. You can use the `PE Explorer` or another Resource Editor to extract or modify Icons in dlls or exe files. But be warned. You can also destroy a working application with these kind of tools.

At least Windows also supports the use of bitmap files in the .bmp format as icons. Note that this format does not support multiple icons.

We might have overlooked other file formats that are supported by Windows. However, we suggest to test other formats for compatibility as they might not work all the way back to Windows 95 or on the NT/non-NT strain. Sticking with one of these two formats should keep you out of trouble.

Apple

Apple Macintosh systems use the Macintosh PICT format, extension .pct. If you are working with an apple system you know a whole lot more about this format than i do. If you don't but would like to be able to install your application on a Mac, simply start with any bitmap format that you feel comfortable to work with. Then find an application that is capable of converting this format into a .pct file. i like to use Paint Shop Pro (PC based), because it provides conversion capabilities among several dozen different file formats.

UNIX flavors

by Marc Eppelmann

As my knowledge, all X based Unix Window systems supports the (ASCII-) XBM (X-Bitmap) and the better XPM (X-PixMap) format. The modern GUI systems like KDE and Gnome can display additionally a lot of other Imagemagick formats, such as GIF, JPG, and PNG.

i suggest to use PNG, because this can lossless compress like the GIF format, however this format is absolutely free. And not least, this can store true transparency informations (It has an alpha channel).

Targets

So, you thought you could escape the ugly mess of operating system dependencies at least with the way how your

Java application is started? Sorry but i have just another bad message. The one positive thing is that here you have a way of escaping, even if doing so has a few less pretty side effects. At first, i would like to discuss various launching options you have available on different operating systems. At the end of the chapter i write about a way to make launching your application OS independent.

Microsoft Windows

On Microsoft Windows you have a variety of options for launching your application. Probably the most simple case is directly starting the Java VM from the command line and typing out all parameters, such as class path, the class name etc. In principle, this can be placed right in a shortcut and should work.

A little more elegant solution is to place this in a batch file and have the shortcut point to this batch file. This will also make it more likely that users can repair or recreate shortcuts. Recreating shortcuts with sophisticated command lines is practically impossible.

Another method is less commonly used but just as possible. Implement a native executable that launches the VM with your Java application. The VM comes as DLL and is used by java.exe in just the same way. As a sample look at the `exlipse.exe` provided by the `Eclipse-IDE`

Clearly, even though the first option is a bit ugly and has some restrictions, it is the most portable solution among the three.

Apple

We hope, there is a IzPack developer currently researching for the details for the Mac environment. We expect an updated chapter in one of the next releases.

UNIX

UNIX provides essentially the same options as Windows. You can simply use the command line option, you can write a shell script and you can write a native launcher. Naturally this stuff is in no way compatible with the equivalent Windows implementations. The native option is even more problematic in this environment, since the code can not even be moved from one UNIX platform to another, without recompilation.

OS Independent Launching

So, after all this rather discouraging news, there is actually a portable way to launch Java applications? You bet! although i have to admit that it is not necessarily the most pretty way of doing things.

This approach is currently used by IzPack. Package your application in a `.jar` file if you don't already do so and make it executable by including the necessary `meta-INF/MANIFEST.MF` information file. i am not going into all the details on how exactly to do this, the Java documentation will have to do. You might have noticed that even though the instructions to install IzPack say to type :

```
java -jar IzPack-install.jar
```

You can just as well double click on `IzPack-install.jar` and it will start up. This procedure will work on all GUI based Java supported operating systems -though you might have to replace double clicking with dropping the file on the VM. In just the same way, you can make the `.jar` file itself the target of a shortcut. Note: This works only, if jars are registered as files, which have to launch by the installed JRE (with: `javaw.exe -jar`)

The one restriction with this approach is that a `.jar` file can only have one main file. So, if you have multiple targets,

they need to be packaged each into a different .jar file. They can be in one .jar file but then you have to start them explicitly, which gets you back to the problems that i mentioned before. This brings me to the ugly part. If you have just one target, then you are all set. If you have multiple targets, you need to create a .jar file for each of them. In addition, you have a much harder time setting the classpath, because each of the .jar files that contain supporting code must be listed. In fact, at present there is no way of setting this during the installation, because IzPack does not yet (version 3.0) support the setting and modification of environment variables.

Command Line

Before i start to write a lot about the use of command line arguments let me state this: If you can avoid using them, do it! Not that there is anything wrong with command line arguments as such. The issue is simply that if you want your application to be usable cross platform (the big Java promise) you should shy away from using command line arguments. The problem here is that not all operating systems actually support command line arguments. To be more precise, to my knowledge only Apple operating systems do not support command line parameters. If you don't care for running your application on a Mac, then you might not worry about this at all. If you are interested to support the Mac as well, read on.

In fact the Mac lower than MacOSX supports command line parameters in a way. More to the point, it supports a single parameter that your application should interpret as the name of a data file to open. You have no way of supplying this to your application through the command line attribute. The operating system generates this when the user drops the file on your application and then passes it as command line argument. That's it. This same behavior will probably fly well on pretty much any system and should therefore be an ok implementation.

So what to do if you want to modify program behavior based on runtime switches? For one thing, you could set system properties accordingly. The disadvantage here is the same as with the command line parameters: the way of setting these might vary between operating systems. The best way seems to be using a property file that contains the configuration data.

Trouble Shooting

by Elmar

It has been some time since i wrote this chapter during which a good number of users had a chance to gather experience. Unfortunately i never know how many have used it successfully without much difficulty. i only hear from those that have encountered one problem or another. The type of problems that i have seen prompted me to write this section, because i think it will help you in locating most problems that you might encounter or at least give you some idea where the problem might be located.

Problems You Can Solve

If you see an exception that essentially says that a library can not be loaded (ShellLink.dll) you have an easy problem to deal with. Your installer file is probably missing the native tag that adds the Windows dll to the installer or something with this tag is not quite right. Read 'What to Add to the Installer' for all details on this topic.

Most other problems cause the ShortcutPanel not to show at all during the installation process. The reason is simply that the ShortcutPanel skips if it does not know what to do or if it has nothing to do (no point showing then and confusing the user). The problem is that this is not always what you intended. The most simple but not so uncommon case is, that the ShortcutPanel cannot find their spec file. This can be caused by a number of reasons. The associated resource tag might be missing in the installer specification file, the target file name might be misspelled (the name you specify for the `id` attribute) or the target file name has a path or package name pre-pended. You have only to use `shortcutSpec.xml` or `Unix_shortcutSpec.xml` and nothing else, just as described in 'What to Add to the Installer'. You can always verify if this part is ok by inspecting the content of the

installer .jar file. The file shortcutSpec.xml should be located in the directory `res`. This inspection can be performed with any zip tool. If the file is not there, first correct this before proceeding.

If the file is there and the panel does not appear, you have a problem within the specification file. In most cases that i have seen, it comes down to a spelling mistake of an attribute or tag name. You just have to carefully make sure that everything is spelled correctly. Don't forget that all names are case sensitive! In a few cases it is also happen, that required or semi-optional attributes are omitted, so you might want to verify if all attributes that you need are actually supplied.

If everything is correct up to this point the problem becomes more elusive. Most likely the panel will not be displayed, because it is instructed not to show. There are be several reasons for this. The simple case is that no location has been specified for the shortcuts in your installation. This can happen if all five location attributes are omitted or if all the ones that are listed are set to `no`. Remember, you have to specify at least one location for every shortcut. If this is also correct, you might have used the `<createForPack>` tag. Review the details in 'Selective Creation of Shortcuts'. One possibility for the panel not to show is that based on the packs that are currently selected for installation no shortcut qualifies for creation. In this case the panel will not show, this is perfectly normal behavior. More likely this condition is true because of some accident and not because it's intended. Make sure the packs that you list for the shortcut are actually defined in your installation and verify that they are all spelled correctly. Remember: case matters! Did the ShortcutPanel use to work in your installation and all of a sudden stopped working? Very likely you are dealing with the last problem. A package name might have been modified and the shortcut spec was not adjusted to stay in sync.

Problems That Have No Solution (yet)

Unfortunately one problem has been very persistent and only recently one user found the reason. The problem occurs when installing on some target systems where non-English characters are used in the storage path for the shortcuts. The problem is that these characters don't seem to be properly translated across the Java Native Interface. This leads to a situation where the proper path can not be located and the shortcut creation fails. i write 'some target systems' because it does not fail everywhere. After much agonizing over this problem, one user found the solution: The shortcut creation works fine if a Sun virtual machine is installed, but fails if a version from IBM happens to be installed. So far i have no solution for this problem but i am trying to find a workaround the problem.

A sample shortcut specification file for Unix

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes" ?>

<shortcuts>

    <programGroup defaultName="IzForge/IzPack"
location="applications"/>

    <!-- Disabled since there is no Frontend
shortcut
```

```
name="IzPack"
programGroup="yes"
desktop="yes"
applications="no"
startMenu="yes"
startup="no"
target="$INSTALL_PATH/bin/izpack-fe.sh"
commandLine=""
workingDirectory="$INSTALL_PATH/bin"
description="Front-End for IzPack
installation tool"
```

```
iconFile="$INSTALL_PATH/bin/icons/izpack.png"
"
```

```
    iconIndex="0"
    type="Application"
    encoding="UTF-8"
    terminal="true"
    KdeSubstUID="false"
    initialState="normal">
    <createForPack name="Core"/>
</shortcut -->
```

```
<shortcut
    name="IzPack Documentation"
    programGroup="yes"
    desktop="yes"
    applications="no"
    startMenu="yes"
    startup="no"
    target="konqueror"
```

```
workingDirectory=""
commandLine=""
initialState="noShow"
iconFile="help"
iconIndex="0"

url="$INSTALL_PATH/doc/izpack/html/izpack-doc.html"
type="Link"
encoding="UTF-8"
description="IzPack user documentation
(html format)">

    <createForPack
name="Documentation-html"/>
    </shortcut>

<shortcut
name="Documentation"
programGroup="yes"
desktop="yes"
applications="no"
startMenu="yes"
startup="no"
target="acroread"
workingDirectory=""

commandLine="$INSTALL_PATH/doc/izpack/pdf/izpack-doc.pdf"
initialState="noShow"
iconFile="acroread"
```

```
        iconIndex="0"

url="$INSTALL_PATH/doc/izpack/pdf/izpack-doc
.pdf"
    type="Application"
    encoding="UTF-8"
    description="IzPack user documentation
(PDF format)">

    <createForPack
name="Documentation-PDF" />
    </shortcut>

<shortcut
    name="Uninstaller"
    programGroup="yes"
    desktop="yes"
    applications="no"
    startMenu="no"
    startup="no"
    target="/usr/lib/java/bin/java"
    commandLine="-jar

&quot;$INSTALL_PATH/Uninstaller/uninstal
ler.jar&quot;"
    initialState="noShow"
    iconFile="trashcan_full"
    iconIndex="0"
    workingDirectory=""
    type="Application"
    encoding="UTF-8"
```

```
description="IzPack uninstaller">  
<createForPack name="Core" />
```

```
</shortcut>
```

```
</shortcuts>
```