

Excelsior Knowledge Base

HOWTO: Use third-party tools to create setup packages for JET-compiled applications (Excelsior JET 3.7 and below)

Article ID: 000017

Last Revised On: 29-Sep-2005

The information in this article applies to:

- Excelsior JET versions 3.1 through 3.7

This article **does not** apply to Excelsior JET 4.0 and above.

The similar article for Excelsior JET 4.0 and above can be found here [000026](#).

SUMMARY

Commercial Editions of Excelsior JET include Excelsior Installer, which you may use to deploy your JET-compiled Java applications. However, the capabilities of Excelsior Installer are limited to very basic operations, such as file copying and creation of Start Menu shortcuts. For more sophisticated setups, you need to write custom extension DLLs or use a third-party setup generator.

This article describes creation of setup packages for JET-compiled applications using third-party tools. The example section contains step-by-step instructions for creating a setup package with NSIS (Nullsoft Scriptable Install System.)

Control files for this article may be found [here](#).

MORE INFORMATION

Introduction

After you have created an executable using Excelsior JET, you need to make a setup package for further distribution of the compiled application to your end users. Before you begin, you should know several things:

1. By default, JET compiles applications in the dynamic link model, i.e. the compiled classes of the application are linked into an executable that in turn links to one or more JET runtime DLLs containing precompiled Java 2 platform classes.

Certain applications may be processed with JetPerfect Global Optimizer (available only in Excelsior JET, Professional Edition.) It links both application classes and platform classes together into a single executable that does not need the above mentioned runtime DLLs.

If you do not use JetPerfect, it is necessary to deploy one or more runtime DLLs along with the application executable. The JetPackII tool will help you determine which runtime DLLs are needed for your executable to work.

2. JET-compiled programs may also require the Java Runtime Environment (JRE) to work. The reason for this requirement is more legal than technical: your executable will actually use just a few native method DLLs from the JRE, but the JRE license prohibits partial redistribution.

The exceptions from this rule are applications that do not use AWT/Swing, i.e. those built using third-party AWT-independent GUI toolkits, such as SWT or LWJGL, or not providing GUI at all (batch/console/server-side). Such applications may work without the JRE if you compile them using Excelsior JET, Professional Edition.

(**Note:** Your program may use AWT and Swing indirectly; for example, there is an xml-to-pdf converter which for no apparent reason requires AWT).

For details, see the "JRE-less operation" section of Excelsior JET User's Guide, Chapter "Application Considerations".

3. JET-compiled applications must be "informed" about the locations on the target system of JET runtime DLLs and/or the JRE, if either or both are required. This is done through patching of the executables. In some cases, it will be performed by JetPackII before packaging, otherwise it will have to be done after installation with the help of the XBIND utility included in your Excelsior JET package.

To summarize, if you have created a JRE-independent executable using JetPerfect, the installation process can be as simple as just copying the application to the target system. If that is not the case, additional files have to be copied and executable patching may be required.

Setup Scenarios

Two options define the actions the installer must perform: placement of the JRE and the number of root directories.

A *root directory* is an abstract destination directory for some or all of the installation package files. The actual pathname for each root, such as `C:\Program Files\My App`, is usually assigned at install time.

single root

All files are copied into one application-specific directory and its subdirectories. The relative directory hierarchy positions of JET runtime DLLs and/or the JRE with respect to your JET-compiled executables are known beforehand, so those executables are patched by JetPackII.

multiple roots

Files are copied to completely different locations, maybe even to different drives. JET-compiled executables must be patched at install time.

Options for JRE placement are as follows:

JRE-independent

The executable(s) being deployed do not require the JRE at all. No extra installer action necessary.

internal JRE

The required version of the JRE is included in the setup package. The installer must copy it to the target system. In case of a single root, the executables will be patched by JetPackII and no further action is necessary to make them work. Otherwise, XBIND must be run to write the JRE location into the executables.

external JRE

The JRE is not included into the setup package but the application requires the JRE. In other words, it relies upon the presence of the proper version of the JRE on the target system. So the installer must find the JRE and “inform” the executable(s) of its location through patching.

Consider the following sample scenarios:

Scenario 1: Single root, internal JRE/JRE-less

These scenarios take place if the following selections are made in JetPackII:

1. JRE packaging is set to either "Do not use JRE" or "Use internal JRE" (step 4 of 10).
2. Backend is set to "Third-party installer, single root placement" (step 7 of 10).

On the final page (step 10 of 10), specify the location where to put the files prepared for the distribution (the image), for example `C:\MyAppImage`, and then click **Create**. When JetPackII finishes creating the image you will find the set of files ready for distribution in the `C:\MyAppImage` directory. All the installer has to do is copy the contents of this directory to the target computer, preserving the subdirectory structure.

Scenario 2: Single root, external JRE

The following selections are made in JetPackII:

1. Choose the JRE packaging option: "Use external JRE" (step 4 of 10).
2. Select the "Third-party installer, single root placement" Back-end (step 7 of 10).

After JetPackII creates the image you will find the following content in the specified image directory (suppose its name is `C:\MyAppImage`):

```
C:\MyAppImage\Root\  
C:\MyAppImage\xbind.exe  
C:\MyAppImage\xbind.script  
C:\MyAppImage\jrelookup.dll      (since version 3.5)
```

where the `C:\MyAppImage\Root` directory contains the files and directories constituting the installation package, `xbind.exe` is the patching utility, and `xbind.script` is its control file. `jrelookup.dll`, introduced in Excelsior JET 3.5, is a helper DLL providing JRE lookup API (see below.)

Now, the installation program should perform the following steps to make the application work on target system:

1. Find the JRE of the required version, possibly using the API provided by `jrelookup.dll`. Suppose it finds the JRE in `C:\JRE`.

Note: JET-compiled applications require exactly the same JRE version as specified in the respective JET Profile.

2. Copy the contents of `C:\MyAppImage\Root` to the desired directory on the destination computer (let it be `C:\Program Files\My App`).
3. Copy `xbind.exe` and `xbind.script` to the destination system. Since these files are only needed at install time, you may copy them to a temporary directory. Let it be `C:\tmp`.
4. Create the `xbind` redirection file (let it be `C:\tmp\xbind.red`) with the following content:

```
Root="C:\Program Files\My App"
JRE="C:\JRE"
```

Note that quotes around the paths are required.

5. Run the following command:

```
C:\tmp\xbind.exe C:\tmp\xbind.script C:\tmp\xbind.red
```

If `xbind` returns zero errorcode then the utility succeeded and the patched executables are ready to work. The only thing the installer has to do now is to remove the files it created in the temporary directory.

Scenario 3: Multiple roots, internal/external JRE

These scenarios take place if the following selections are made in JetPackII:

1. JRE packaging is set to either "Use internal JRE" or "Use external JRE" (step 4 of 10).
2. Backend is set to "Third-party installer, multiple roots placement" (step 7 of 10).

When JetPackII create the setup image you will find the following content at the specified location (it is similar to the image for the "Single root, external JRE" scenario):

```
C:\MyAppImage\Root1\
C:\MyAppImage\Root1\
. . .
C:\MyAppImage\RootN\
C:\MyAppImage\xbind.exe
C:\MyAppImage\xbind.script
C:\MyAppImage\jrelookup.dll (appeared since version 3.5)
```

where the `C:\MyAppImage\RootX` directories contain directory structures and files you defined in JetPackII for each respective root.

To make the application work on the target system, the installation program should perform the following steps:

1. If the JRE is not included in the package ("external JRE"), find a JRE of the required version, possibly using the API provided by `jrelookup.dll`. Note that JET-compiled applications require exactly the same JRE version as the one specified in the respective JET Profile. Suppose the JRE was found in *JRE-Location*.

2. Copy the contents of all `RootX` directories to desired locations on the target systems (let them be *Root1-Location*, *Root2-Location*, and so on.)

When the JRE is included in the package (“internal JRE”), it is placed in one of the root directories and therefore must be copied to the target system on this step. Suppose the JRE was copied to *JRE-Location*.

3. Copy `xbind.exe` and `xbind.script` to the destination system. Since these files are only needed at install time, you may copy them to a temporary directory. Let it be `C:\tmp`.
4. Create a redirection file (let it be `C:\tmp\xbind.red`) with the following contents:

```
Root1="Root1-Location"
Root2="Root2-Location"
.
.
.
RootN="RootN-Location"
JRE="JRE-Location"
```

Note that quotes around the pathnames are required.

5. Run the following command line:

```
C:\tmp\xbind.exe C:\tmp\xbind.script C:\tmp\xbind.red
```

If `xbind` returns zero errorcode then the utility succeeded and the patched executables are ready to work. The only thing the installer has to do now is to remove the files it created in the temporary directory.

JRE Lookup API (since JET 3.5)

In order to reduce the size of the resulting setup package you may decide not to redistribute the JRE, if you can assume it will be present at enduser systems (which is a reasonably safe assumption in a corporate environment.) As it was mentioned above, in this case the installer will have to locate the JRE of the proper version on the target system.

The JRE Lookup API was made for helping the installer to locate the JRE. It includes `jrelookup.dll` (which can be found alongside the `xbind` tool in the installation package image) and the header file `jrelookup.h` (located in the `include` subdirectory of your JET installation).

The API provides the following set of functions (please refer to Excelsior JET User’s Guide or the `jrelookup.h` files for the exact description of each function.)

```
int STDCALL JRELookup_GetVersion (char* buffer, int bufferlen);
int STDCALL JRELookup_Lookup ();
int STDCALL JRELookup_GetCount ();
int STDCALL JRELookup_GetPath (int index, char* buffer, int bufferlen);
int STDCALL JRELookup_IsInternational (int index);
int STDCALL JRELookup_IsPublic (int index);
int STDCALL JRELookup_AddDirectory (const char* dir);
```

The common usage scenario of the JRE Lookup API would be:

1. Copy `jrelookup.dll` to a temporary directory on the target computer and load it.
2. Call `JRELookup_Lookup()` to conduct the search for the JRE.

3. Call `JRELookup_GetCount()` to get the number of found JREs.
4. If the number is more than one, the installer should choose somehow which JRE it is going to use, for example it could take the first one or prompt the user. To get extended info about the found JREs, it may call `JRELookup_GetPath()`, `JRELookup_IsInternational()`, `JRELookup_IsPublic()`.

If the API fails to find any JRE, the setup program should prompt the user for the path to the JRE of the proper version, which can be obtained by calling `JRELookup_GetVersion()`. Then setup should pass the pathname specified by user to `JRELookup_AddDirectory()` to check if it really contains a proper JRE.

Example

In this example it would be shown how to create a "Single root, external JRE" setup for the `hello` application using NSIS (Nullsoft Scriptable Install System). Let's perform the following steps:

1. Go to the `samples\Hello` subdirectory of your JET installation.
2. To compile `hello.java` to `hello.exe`, run the `build.bat` script.
3. Run `JetPackII` and define package contents in the following way:

```
/Root
  /JET RT
    hello.exe
```

4. On the JRE selection page, choose "Use external JRE".
5. Perform trial run.
6. Choose "Third-party installer, single root placement" back-end.
7. Choose package image location and create the image.
8. Unpack the `.zip` file for this article into some directory, correct paths in `pack.bat` and run it to create a setup package with NSIS.

REFERENCES

1. JET User's Guide <http://www.excelsior-usa.com/doc/jc.html>, Chapters "JetPerfect Global Optimizer", "Deployment Automation".
2. Nullsoft Scriptable Install System <http://nsis.sourceforge.net>
3. Inno Setup <http://www.jrsoftware.org/isinfo.htm>