

NewJ Library for C++ Developer's Guide



PureNative Software

www.pure-native.com

Copyright © 2002-2004 PureNative Software Corporation. All rights reserved.

PureNative, the PureNative logo, NewJ, the NewJ logo, NewJNI, NewJNIInterop, and Pie are trademarks of PureNative Software Corporation. Java is a registered trademark of Sun Microsystems, Inc. All other trademarks and registered trademarks are property of their respective owners.

The software described herein is licensed. Unauthorized reproduction or distribution of this software, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law. Consult accompanying license agreement for additional permissions and restrictions.

NewJ Library for C++ is based on the Java 2 Platform, Standard Edition, version 1.2.2 API Specification from Sun Microsystems, Inc. *NewJNI* and *NewJNIInterop* are compatible with the JNI Specification from Sun Microsystems, Inc., and support any JNI-compatible version of the Java platform, including version 1.2, 1.3, or later. *NewJ Library for C++* is not a product of nor endorsed by Sun Microsystems, Inc.

Patent Pending.

Developer release 2: May 2004.

{6} Adding NewJ/NewJNI/NewJNInterop Support to Your Application or Library

Adding NewJ Library Support to Your Application or Library

This chapter explains how to add NewJ Library support to your application or library, including the JNI interoperability features of NewJNI and NewJNInterop. NewJ AppWizards are provided to automate this process. On the other hand, if you need to have finer control over this process this chapter also explains how to manually add NewJ support to your application or library.

NewJ AppWizards

Today many software developers organize their source code into projects, often doing so with the aid of an integrated development environment (IDE) like Microsoft Visual C++. Such development environments empower developers to easily create, modify, and build their projects. The NewJ Library for C++ provides its own AppWizards which are seamlessly integrated with the Visual C++ 6.0 IDE. This following sections explain how to utilize the NewJ AppWizards to create new application projects which use the Pie Run-time Library and Core J2 Library.

Creating a New Project using Visual C++ 6.0

Prerequisites: Before you can create a new project which uses the Pie Run-time Library and Core J2 Library, your include file paths and library file paths must be set properly. See the heading “Setting up Your Development Environment: Visual C++ 6.0” in the Installation Instructions chapter for how to do this.

There are two ways to create a new project: (1) automatically using the NewJ AppWizards, (2) manually using the standard Win32 AppWizards in Visual C++. First, we'll consider how to use the NewJ AppWizards, and then we'll explain how to create a new project manually using the standard AppWizards.

Creating a New Project Automatically with the NewJ AppWizards

There are three different NewJ AppWizards: (1) NewJ Console Application, (2) NewJ MFC Application, (3) NewJ Win32 Application. The NewJ Console Application AppWizard, which is based on the standard Win32 Console Application AppWizard, creates Win32 console applications that use the NewJ Library for C++. The NewJ MFC Application AppWizard is based on the standard MFC AppWizard (.exe), and it produces MFC applications that use the NewJ Library. The NewJ Win32 Application AppWizard, which is

based on the standard Win32 Application AppWizard, makes Win32 applications that use the NewJ Library.

To create a new console application using the NewJ Console Application AppWizards, follow these steps in order:

1. From within the Visual C++ 6.0 IDE, select the menu item File > New...
2. Select the Projects tab
3. Select “NewJ Console Application”
4. Enter the name of your application, which is usually the name of your main application class
5. Choose OK.
6. Enter the name of your main application class. Note that your main application class name must be a valid C++ class name.
7. If you would like to have JNI interoperability features, select the JNI option.
8. Choose Finish.
9. When the New Project Information window appears, choose OK.

Your NewJ console application may now be edited, compiled, and run at will.

To create a new MFC application that makes use of the NewJ Library, do the following:

1. Select the menu item File > New...
2. Select the Projects tab
3. Choose “NewJ MFC Application”
4. Enter the name of your application.
5. Choose OK.
6. For each step, select the desired options, and then choose Next.
7. When you reach the final step, you will be asked: “What advanced features would you like to include?” If you wish for your application to support XP themed look and feel, select the Common Control Manifest option.
8. If you would like to have JNI interoperability features, select the JNI option.
9. Choose Finish.
10. When the New Project Information window appears, choose OK.

Your NewJ MFC application is ready to be edited, compiled, and run.

To create a NewJ Win32 application, follow these steps:

1. Select the menu item File > New...
2. Select the Projects tab
3. Choose “NewJ Win32 Application”
4. Enter the name of your application.
5. Choose OK.
6. Select the kind of Win32 application you wish to create: An empty project, A simple Win32 application, or A typical “Hello World!” application.

7. For “What advanced features would you like to include?” If you wish for your application to support XP themed look and feel, select the Common Control Manifest option.
8. If you would like to have JNI interoperability features, select the JNI option.
9. Choose Finish.
10. When the New Project Information window appears, choose OK.

Your NewJ Win32 application is now ready to be edited, compiled, and run.

Creating a New Project Manually with the Standard Win32 AppWizards

To create a new project using the standard AppWizards, follow these steps in order:

Create a new Win32 Application (or Win32 Console Application) project

1. Select menu item File > New...
2. Select the Projects tab.
3. Select "Win32 Application".
4. Enter the name of your application, which is usually the same as the name of your main application class.
5. Choose OK.
6. Select "Empty project".
7. Choose Finish.

Change project settings

8. Select menu item Project > Settings...
9. Under "Settings for:", select "All Configurations" option.
10. Select the first line in the tree view on the left.
Enable Run-Time Type Information (RTTI) for all files in project
11. Select the "C++" tab on the right.
12. Under category, select "C++ Language".
13. Select "Enable Run-Time Type Information (RTTI)" check box so that it is checked.
Disable Precompiled Header support
14. Select the "C++" tab on the right.
15. Under Category, select "Precompiled Headers".
16. Select the "Not using precompiled headers" option.
17. Define "PIE_NATIVE_ENVIRONMENT" symbol.
18. Select the "C++" tab on the right.
19. Under Category, select "Preprocessor".
20. Under "Preprocessor symbols", define PIE_NATIVE_ENVIRONMENT
Enable multithreading support.
21. Select the "C++" tab on the right.
22. Under Category, select "Code Generation".
23. Under "Use run-time library", select "Debug Multithreaded" for debug, and select "Multithreaded" for release.
Link to Pie static library
24. Select "Link" tab in the right.
25. For the "Debug" configuration, under "Object/library modules", go to the end of the field, enter a space, and add the libraries "newjd.lib" and "fdlibmd.lib".

26. For the “Release” configuration, under "Object/library modules", go to the end of the field, enter a space, and add the libraries “newj.lib” and “fclibm.lib”.

27. Choose OK.

28. Delete "Source Files", "Header Files", "Resource Files" folders (optional)

28. Create file named "PieMain.cpp". For example:

```
// #define PIE_STARTER_CLASS_NAMESPACE // this statement is optional
#define PIE_STARTER_CLASS MyApp
#include "MyApp.h"
#include <pie/x-startup/shellstarter.cpp>
```

29. Add "PieMain.cpp" to project.

30. Create source file and header file for application class.

Adding NewJ Library Support to an Existing Application or Library

Instead of creating new projects for legacy applications, it is much easier to add NewJ Library support to such applications. NewJ Library support may be added to practically any kind of C++ application. To add NewJ Library support to an existing project, do the following:

1. Open your project's workspace in the Visual C++ 6.0 IDE.
2. Select your project, which will be done automatically if your workspace only contains a single project.
3. Select the menu item Project > Settings...
4. Under “Settings For:” select “All Configurations”.
5. If your project uses MFC in a Shared DLL, change it to use MFC in a Static Library. (This is required by the NewJ Library for C++. This requirement will likely be eliminated in the future.)
6. Select the “C/C++” tab.
7. Under “Category” select “C++ Language”.
8. Enable Run-time Type Information (RTTI).
9. Under “Category” select “Preprocessor”.
10. Under “Preprocessor definitions” add the symbol `PIE_NATIVE_ENVIRONMENT`.
11. Under “Settings For:” select the Release configuration.
12. Select the “C/C++” tab.
13. Under “Category” select “Code Generation”.
14. Under “Use run-time library” select “Multithreaded”. Make sure *not* to select “Multithreaded DLL”.
15. Select the “Link” tab.
16. Under “Category” select “General”.
17. For “Object/library modules” add “newj.lib” and “fclibm.lib”.
18. If you would like to have JNI interoperability features, also add “newjninterop.lib”, “newjni2.lib”, and “jvm.lib”.

19. Under “Settings For:” select the Debug configuration
20. Select the “C/C++” tab
21. Under “Category” select “Code Generation”
22. Under “Use run-time library” select “Debug Multithreaded”. Make sure *not* to select “Debug Multithreaded DLL”.
23. Select the “Link” tab
24. Under “Category” select “General”
25. For “Object/library modules” add “newjd.lib” and “fdlibmd.lib”
26. If you would like to have JNI interoperability features, also add “newjninteropd.lib”, “newjni2d.lib”, and “jvm.lib”.

Your legacy application project is now ready to use the NewJ Library for C++!