JNBridgePro™ for .NET Core Users' Guide

Version 11.0

jnbridge.com

JNBridge, LLC
jnbridge.com

September 27, 2021

Table of Contents

Preface

JNBridgePro for .NET Core is a variant of JNBridgePro that allows interoperability between .NET Core (and .NET 5) code and Java code. It works very similarly to the way that traditional JNBridgePro (for .NET Framework) works. This document explains how JNBridgePro for .NET Core works, when it differs from traditional JNBridgePro. If you have a question on how to use JNBridgePro for .NET Core, and you cannot find the answer here, you should consult the *JNBridgePro Users' Guide*, where you are likely to find the answer.

System Requirements

The .NET side of applications using JNBridgePro must run on .NET Core 3.0 or 3.1 or .NET 5. The Java side must run on Java 5 through 17.

The .NET side can run on any Windows or Linux platform on which .NET Core 3.0/3.1 and .NET 5 is supported (except for shared memory on ARM32, ARM64, and Alpine Linux). (MacOS X is not currently supported.) For a full list of supported platforms, see https://github.com/dotnet/core/blob/master/release-notes/3.0/3.0-supported-os.md or similar Microsoft documentation. Depending on the platform (as indicated in the abovementioned documentation), 32-bit, 64-bit, or both, may be supported.

The Java side can run on any platform that that supports Java 5 through Java 17. Please see Oracle documentation for further information.

Differences between .NET Framework and .NET Core versions of JNBridgePro

The following features of "traditional" JNBridgePro are not available in JNBridgePro for .NET Core:

- Failover.
- HTTP/SOAP communications.
- Embedding of Java UI elements in .NET Windows Forms and WPF applications
- Embedding of .NET Windows Forms and WPF elements in Java AWT, Swing, or SWT applications.

Proxy generation

To generate .NET proxy DLLs for .NET Core-to-Java projects, you must use a proxy generation tool (either the standalone GUI-based tool, command-line tool, or Visual Studio plugin) from the "traditional" JNBridgePro that works with .NET Framework 4.x. This means that such proxies must be generated on a Windows machine with .NET Framework 4.x installed. The resulting proxies will work with .NET Core applications. The reason that there is no .NET Core-targeted proxy generation tool for .NET Core-to-Java projects is that while .NET Core allows the creation of dynamic

assemblies (something that is used in the proxy generation tool) in memory, it does not currently allow these dynamic assemblies to be written out to disk as DLL files. We hope that this will be remedied in future versions of .NET Core (there is a package being developed on GitHub that is designed to do this), but it is not ready for use in a product.

To generate Java proxy JAR files for Java-to-.NET Core projects, we provide a proxy generation tool designed to work with .NET Core. This proxy generation tool is command-line only. (.NET Core 3.0 does support Windows Forms and WPF applications, but only on Windows, and we wanted this tool to work on all platforms supported by .NET Core and Java.) This command-line-based proxy generation tool works in the same way as the command-line-based "traditional" proxy generation tool in the Java-to-.NET direction. (See the *Users' Guide* for more details.) If you attempt to use this proxy generation tool to generate proxy DLLs for .NET Core-to-Java projects, it will display an error message.

Components

JNBridgePro for .NET Core includes a number of components. Note that applications are distributed as framework-dependent DLLs, not as framework-independent EXEs.

- jnbproxy.dll: The command-line proxy generation tool for generating proxy JAR files for Java-to-.NET Core projects. jnbpcommon.dll, which is also part of the "traditional" JNBridgePro proxy generation tool, is not needed in the .NET Core-targeted tool. Also, JNBProxy.deps.json and JNBProxy.runtimeconfig.json – both these files must be in the same folder as jnbproxy.dll.

- registrationTool.dll: A command-line version of the registration tool that can run on .NET Core. Also, registrationTool.deps.json and registrationTool.runtimeconfig.json – both these files must be in the same folder as registrationTool.dll.

- jnbdotnetside.dll: A command-line version of the .NET-side for Java-to-.NET Core projects that can run on .NET Core. Also, jnbdotnetside.deps.json and jnbdotnetside.runtimeconfig.json – both these files must be in the same folder as registrationTool.dll.

- jnbshare.dll: The central JNBridgePro .NET-side runtime component. It performs the same functions as the jnbshare.dll in the "traditional" JNBridgePro.

  To run, jnbshare.dll depends on the following NuGet packages, which must be downloaded and included with your project:

  - Microsoft.Extensions.Configuration.Abstractions
  - Microsoft.Extensions.Configuration.Binder
  - Microsoft.Extensions.Configuration.Json

- jnbsharedmem_x64.dll, jnbsharedmem_x86.dll, libJNBSharedMem_x64.so: The .NET-side runtime component containing functionality supporting shared memory communication, for 64-bit Windows, 32-bit Windows, and 64-bit Linux, respectively. Use the particular version as appropriate for your application. Alternatively, include all of them and the appropriate one will be used, depending on underlying platform and process bitness. Only 64-bit Linux is being supported.

- jnbjavaentry_x64.dll, jnbjavaentrry_x86.dll, and libJNBJavaEntry_x64.so: .NET-side runtime components containing functionality for shared memory in the Java-to-.NET Core direction, for 64-bit Windows, 32-bit Windows, and 64-bit Linux, respectively. Use the particular version as appropriate for your application. Alternatively, include all of them and the appropriate one will be used, depending on underlying platform and process bitness. Only 64-bit Linux is being supported.

- jnbjavaentry2.dll: .NET-side runtime components containing additional functionality for shared memory in the Java-to-.NET Core direction. It will work on all supported .NET Core platforms. Note that, unlike "traditional" JNBridgePro, there is only a single version of this assembly, not multiple versions depending on the bitness of the application's process.

- jnbauth_x64.dll, jnbauth_x86.dll, jnbauth_x64.so: Components containing licensing functionality for JNBridgePro, for 64-bit Windows, 32-bit Windows, and 64-bit Linux, respectively. Use the particular version as appropriate for your application. Alternatively, include all of them and the appropriate one will be used, depending on underlying platform and process bitness. Only 64-bit Linux is being supported.

All the above components are version 11.0.

Use the same Java-side components (jnbcore.jar and bcel-5.1-jnbridge.jar) that are used in "traditional" JNBridgePro.

Configuration

One of the biggest differences between traditional JNBridgePro and JNBridgePro for .NET Core is that the .NET Core version uses JSON to specify JNBridgePro configuration. This is because .NET Core applications do not use app.config files. If configured using configuration files rather than programmatically, an application using JNBridgePro for .NET Core must have a file jnbridgeConfig.json, with the following form:

```
{
  "dotNetToJavaConfig": {
    "scheme": "jtcp or sharedmem",
    "host": "remote host name or IP address",
    "port": remotePortNumber,
    "useSSL": "true or false",
    "SSLAlternateServerNames":  [ "list", "of", "acceptable server names" ],
    "SSLClientCertificateLocation": "path to client certificate",
    "SSLClientCertificatePassword": "client certificate password, if necessary",
    "SSLClientCertificatePasswordFileLocation":
       "path to file containing client certificate password, if necessary",
```


```
    "jvmOptions": [ "list", "of", "jvm options" ],
    "jvm": "path to jvm.dll if using shared memory",
    "jvm32": "path to 32-bit jvm.dll if using shared memory",
    "jvm64": "path to 64-bit jvm.dll if using shared memory",
    "jnbcore": "path to jnbcore.jar",
    "bcel": "path to bcel-5.1-jnbridge.jar",
    "classpath": "semicolon-separated classpath"
  },
  "javaToDotNetConfig": {
    "scheme": "jtcp",
    "port": "portNumber",
    "useSSL": "true or false",
    "certificateLocation": "path to SSL certificate",
    "useIPv6": "true or false"
    "hostIP": "IP address for which .NET side will take requests",
    "ipWhitelist": [ "list", "of", "whitelisted client addresses"],
  },
  "assemblyList": [ "list", "of", "assembly paths"],
  "tcpNoDelay": "true or false",
  "licenseLocation": {
    "host": "host",
    "port": "portNumber",
    "directory": "C:\\Program Files (x86)\\JNBridge\\JNBridgePro v11.0"
  },
  "javaSideDeclarations": [
    {
      "javaSideName": "second_PrimaryURL",
      "javaSideURL": "jtcp://localhost:8090/JNBDispatcher"
    },
    {
      "javaSideName": "js2",
      "javaSideURL": "url2"
    }
  ]
}
```

The elements above correspond directly to elements in the traditional JNBridgePro app.config file. Clearly, not all elements are necessary, and some sets of elements are mutually exclusive. See the *JNBridgePro Users' Guide* for more information on how these elements are used.

The file jnbridgeConfig.json must be in the same folder as the copy of jnbshare.dll being used by your application.

Note that, while the JSON specification does not allow for comments (a serious oversight, in our opinion), Microsoft's implementation does support comments. Everything from a '//' sequence to the end of a line is considered a comment and ignored.

Programmatic configurating using the JNBRemotingConfiguration.specifyRemotingConfiguration() APIs is also available and works the same way as it does in "traditional" JNBridgePro.

Java-side configuration works the same way as in "traditional" JNBridgePro, with one additional property that may be configured:

dotNetSide.coreClrPath=*path_to_folder_containing_coreClr.dll_or_libcoreclr.so*

Use this in Java-to-.NET Core projects that use shared memory communication. The property specifies the folder containing coreClr.dll (on Windows) or libcoreclr.so (on Linux). This is the file containing the .NET Core CLR (Common Language Runtime) implementation. For example, on Windows it could be:

dotNetSide.coreClrPath=C:/Program Files/dotnet/shared/Microsoft.NETCore.App/3.0.0

As with other paths in the Java-side configuration, use forward-slashes ('/') rather than backslashes ('\').

.NET-side configuration file examples

.NET Core-to-Java projects using TCP/binary communications

```
{
  "dotNetToJavaConfig": {
    "scheme": "jtcp",
    "host": "remote host name or IP address",
    "port": remotePortNumber,
    "useSSL": "true or false",     // optional.
                                   // If true, additional properties are
                                   // necessary
  },
  "tcpNoDelay": "true or false",   // optional
  "licenseLocation": {      // use host/port, or directory, but not both
    "host": "host",
    "port": "portNumber",
    "directory": "C:\\Program Files (x86)\\JNBridge\\JNBridgePro v11.0"
  },
  "javaSideDeclarations": [ // optional: only use with multiple Java sides
    {
      "javaSideName": "second_PrimaryURL",
      "javaSideURL": "jtcp://localhost:8090/JNBDispatcher"
    },
    {
      "javaSideName": "js2",
      "javaSideURL": "url2"
    }
  ]
}
```

Example .NET Core-to-Java projects using shared memory communications (Windows)

```
{
  "dotNetToJavaConfig": {
    "scheme": "sharedmem",
    "jvmOptions": [ "list", "of", "jvm options" ],     // optional
      // if jvm32 and jvm64 are both used, the one of the
```

```
        // appropriate process bitness will be used.
        // jvm is available for backward compatibility, but is deprecated
      "jvm": "path to jvm.dll ",
      "jvm32": "path to 32-bit jvm.dll",
      "jvm64": "path to 64-bit jvm.dll ",
      "jnbcore": "path to jnbcore.jar",
      "bcel": "path to bcel-5.1-jnbridge.jar",
      "classpath": "semicolon-separated classpath"
    },
    "licenseLocation": {      // use host/port, or directory, but not both
      "host": "host",
      "port": "portNumber",
      "directory": "C:\\Program Files (x86)\\JNBridge\\JNBridgePro v11.0"
    }
  }
```

Example .NET Core-to-Java projects using shared memory communications (Linux)

```
{
  "dotNetToJavaConfig": {
    "scheme": "sharedmem",
    "jvm64": "/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server/libjvm.so",
    "jnbcore": "/home/myName/N2JProject/netcoreapp3.0/jnbcore.jar",
    "bcel": "/home/myName/N2JProject/netcoreapp3.0/bcel-5.1-jnbridge.jar",
    "classpath":
"/home/myName/N2JProject/netcoreapp3.0/Java:/home/citrin/Downloads/netcoreapp3.0/Java
2nd Proxy"
  },
  "licenseLocation": {
    "directory": "/home/myName/N2JProject/netcoreapp3.0"
  }
}
```

Java-to-.NET Core projects using TCP/binary communications

```
  {
    "javaToDotNetConfig": {
      "scheme": "jtcp",
      "port": "portNumber",
      "useSSL": "true or false",
      "certificateLocation": "path to SSL certificate",
      "useIPv6": "true or false"
      "useClassWhiteList": "true or false",
      "classWhiteListFile": "path to class whitelist file",
    },
    "assemblyList": [ "list", "of", "assembly paths"],
    "tcpNoDelay": "true or false",   // optional
    "licenseLocation": {      // use host/port, or directory, but not both
      "host": "host",
      "port": "portNumber",
      "directory": "C:\\Program Files (x86)\\JNBridge\\JNBridgePro v11.0"
    }
  }
```

Example: Java-to-.NET Core projects using shared memory communications

When using shared memory in Java-to-.NET Core projects, you still must have a jnbridgeConfig.json file, but it will only include the location of the license file or the host and port of the license server:

```
{
  "licenseLocation": {  // use host/port or directory, but not both
    //"host": "host",
    //"port": "port",
    "directory": "C:\\Program Files (x86)\\JNBridge\\JNBridgePro v11.0"
  }
}
```

Licensing

In JNBridgePro for .NET Core, the or development or deployment license file must either be in the same folder as the startup DLL, or its location must be indicated in the jnbridgeConfig.json file, in the licenseLocation:directory element. Alternatively, if you are using a license server, specify its location in jnbridgeConfig.json using the licenseLocation:host and licenseLocation:port elements.

The same JNBridgePro license file will work for both .NET Framework and .NET Core applications on any machine on which the license is valid.