Microsoft®

Microsoft Java Virtual Machine Migration Guide for Developers

Version 2.2

April 7, 2004

Contents

1

Overview

In a settlement agreement reached in January 2001 to resolve a dispute over Microsoft's distribution of its Java implementation, Sun and Microsoft agreed to limit the duration of Microsoft's use of Sun's source code and compatibility test suites to support the Microsoft® Java Virtual Machine (MSJVM). Because some developers and enterprises expressed concern about their ability to eliminate dependencies on the MSJVM in the time period originally provided, Sun and Microsoft agreed to extend Microsoft's license to use Sun's Java source code and compatibility test suites. This extension allows Microsoft to support the MSJVM and address potential security issues until December 31, 2007.

In preparation for the end of Microsoft's license to use Sun's Java source code and compatibility suites, Microsoft has been phasing out the MSJVM in its products since the settlement was reached. Going forward, the MSJVM will not be included in any future Microsoft products. The MSJVM is obsolete code and will no longer be enhanced or developed. Microsoft will only continue to provide security fixes to help keep customer machines secure.

Sites and applications that currently depend upon the MSJVM will not function correctly when accessed by systems that do not have the MSJVM installed. For an Internet or intranet site, users with problems would include those who have new Microsoft systems that do not contain the MSJVM (updated systems may not be affected as upgrades to Windows operating systems do not remove existing copies of the MSJVM).

Microsoft recommends that customers of Microsoft Visual J++® and the Microsoft SDK for Java who have built applications or Web sites that use the MSJVM begin identifying MSJVM dependencies and transition to an alternate solution. For customers choosing to pursue a Microsoft supported solution, information is provided on the Microsoft Visual J#® .NET and the Microsoft Java Language Conversion Assistant. These products assist developers in migrating existing source code to the .NET Framework.

Intended Audience

This guide is a technical guide intended for developers responsible with defining the types of migration or transition from the MSJVM that need to be made, determining the approaches to the work that are appropriate, and implementing those changes.

Knowledge Prerequisites

Knowledge of Java is assumed when reading this guide. Other knowledge requirements will depend on the transition solutions you choose, but could include ECMAScript, Dynamic HTML (DHTML), and Microsoft solutions such as the .NET Framework and the C# language.

How to Use This Solution Guide

This document provides specific guidance for MSJVM migration projects including migrating to a .NET environment, rewriting in other rendering technologies, and switching to another JRE.

For specifics on discovering the dependencies on the MSJVM and assessing business needs accordingly, refer to the *Microsoft Java Virtual Machine Transition Guide for IT Professionals*.

The following is a list of content for this guide's chapters:

Chapter 1—Overview. This chapter outlines the guide and describes why Microsoft is discontinuing support for the MSJVM and a brief summary of transition options.

Chapter 2—Migrate to .NET Using Visual J# and J# Browser Control. This chapter discusses Visual J# and the J# Browser Control automated tool for converting Java code to .NET.

Chapter 3—Migrate to .NET Using the Java Language Conversion Assistant. This chapter discusses the Java Language Conversion Assistant (JLCA) automated tool for converting Java code to .NET.

Chapter 4—Other Rendering Technologies. This chapter summarizes other rendering technologies available such as DHTML, ECMAScript, and Flash.

Chapter 5—Switch to Another JRE. This chapter discusses some of the issues involved in selecting, installing, and deploying an alternate Java runtime environment (JRE).

Why a Transition Is Needed

The future of the MSJVM is defined by the settlement between Sun and Microsoft, which permits Microsoft to continue support of the MSJVM until December 31, 2007.

One option is to do nothing to resolve MSJVM dependencies, however, this option is not recommended by Microsoft. The MSJVM will become unsupported software on December 31, 2007, and as a result, there are serious security implications of not transitioning MSJVM dependencies prior to this date. Although Microsoft will continue to support customers, the ability to do so will be extremely limited. In light of these restrictions, new Microsoft products will not include the MSJVM.

Running the MSJVM as unsupported software exposes your system to the possibility of severe security vulnerabilities. If a security issue is identified following the deadline, Microsoft will be unable to make security or functionality patches for the MSJVM available to users.

Prior to the deadline, every effort should be made to remove or transition away from the MSJVM. At a minimum, if you are unable to successfully transition or decide to not transition away from the MSJVM, locking down security for the MSJVM to trusted sites is recommended.

Transition Options Available

Microsoft strongly recommends that you determine the extent of your dependencies on the MSJVM immediately. Keep in mind that any solution chosen to address discovered dependencies requires extensive testing and an understanding of the range of systems affected.

Once you identify dependencies on the MSJVM, Microsoft recommends that you take one or more of the following actions, as appropriate:

- Remove the MSJVM or lock down security and restrict access to applets or applications as discussed in the *Microsoft Java Virtual Machine Transition Guide for IT Professionals*.
- Migrate to another solution:
    - Visual J# .NET or J# Browser Controls.
    - C# on .NET, using the Java Language Conversion Assistant.
    - Other rendering technologies, such as DHTML, ECMAScript, or a 3rd party display technology such as Flash.
    - An alternate JRE.

Note For specific information on how to identify MSJVM dependencies, please refer to the *Microsoft Java Virtual Machine Transition Guide for IT Professionals*.

Migrate to .NET

Migrating to .NET allows users to implement a Microsoft supported solution and reuse existing Java code. Visual J# and the Java Language Conversion Assistant (JLCA) are two options for migrating J++ or Java applications and applets to .NET. Although both provide wizards and command line utilities to automatically migrate existing Java applets and applications into the .NET Framework, some manual conversion of code may be necessary.

Because of the cross-language interoperability of the .NET Framework, developers can choose to mix and match J# and the JLCA for a given migration project, depending on what the goals are for various components of that project.

Visual J# vs. Visual C#

If you decide on migrating to .NET, you must make an assessment on whether converting your existing code to Visual J# or to Visual C# is right for you. For example, if you have significant investments in J++ and an extensive knowledge of Java, migrating to Visual J# may be a viable option. Visual J# allows developers to migrate J++ or Java applications to .NET while preserving the Java-language syntax and, in most cases, the same JDK functionality. The J# Browser Control is a new addition to J# aimed at migrating applets.

However, if there is a need for significant architectural restructuring of your applets or applications, migrating to Visual C# can help accomplish those goals. The JLCA for C# converts both Java applets and applications to C# and the .NET Framework.

There are a number of other factors that you should take into consideration when deciding between Visual J# and Visual C#:

- **Conversion**
  Visual J# often converts approximately 95% of all MS Java or Java JDK 1.1.4 or less code. Less than 5% of code usually needs manual upgrading after converting to Visual J#. The JLCA converts approximately 90% of code and supports functionality in J2SE 1.3 and J2EE 1.3 or less. In most projects, less than 10% of code needs manual upgrading after converting to Visual C#.
- **.NET Framework and Language**
  Although Visual J# supports Java language syntax and uses Java data types, coercion between data types is not seamless and Value Types, Custom Attributes, and Enumerations are not supported. Visual C# allows full framework access and access to all .NET features in accordance with ISO standards.
- **Required Run Times**
  Visual J# requires the .NET Framework and J# Browser Controls runtimes to be installed on end-user computers. Visual C# only requires the .NET Framework runtime to be installed.
- **Developer Community**
  Visual J# is supported by a small niche development community. Visual C# is supported by a large and very active development community.

Note For more information on migrating to .NET, including case studies on migrating from J++ to Visual J# and instructions on downloading the JLCA, please refer to:

Visual J#: http://msdn.microsoft.com/vjsharp.

Visual C#: http://msdn.microsoft.com/vcsharp.

Microsoft Product Support Services

Microsoft Product Support Services (PSS) is available to help assist migrating existing applications and functionality to any Microsoft technology, including the .NET environment. Microsoft PSS can assist developers in migrating current Java code to Visual J# or Visual C#, enabling migration of existing applications with current feature sets. Technical assistance on adding additional functionality is not included and will continue to be fee-based. No other special offers will be made available after December 31, 2007. MSJVM migration support is now available worldwide. Please contact PSS at **800-936-5800** in the US or your local subsidiary.

Note Microsoft will continue support of existing products that have MSJVM dependencies according to existing support policies, but cannot support MSJVM related issues after December 31, 2007. For more information on International Support, refer to: http://support.microsoft.com/common/international.aspx.

2

Migrate to .NET Using Visual J# and J# Browser Controls

Microsoft Visual J# can be used by developers who are familiar with the Java-language syntax to build applications and services on the .NET Framework. It integrates the Java-language syntax into the Visual Studio® .NET integrated development environment (IDE). Visual J# also supports most of the functionality found in Visual J++ 6.0, including Microsoft extensions such as WFC.

Note Applications and services built with Visual J# will only run on the .NET Framework. Visual J# is independently developed by Microsoft and is not endorsed or approved by Sun Microsystems, Inc. For more information refer to: http://msdn.microsoft.com/vjsharp/

Migrating J++ Applications with the Visual J# Upgrade Wizard

Visual J# includes an upgrade wizard that converts Visual J++ 6.0 projects into Visual J# projects. The intuitive upgrade wizard is invoked by opening a Visual J++ project file in Visual J#. It is important to note that no Java source code changes are made to the project; only the project and solution files are upgraded to match the Visual Studio .NET format.

Once the upgrade is complete, a summary report is displayed. This report lists any items that are not fully supported in Visual J#. Approximately 95% of a Visual J++ project will upgrade to Visual J# without any required changes. However, for functionality not supported directly in J#, there is equivalent .NET Framework functionality that the developer can utilize.

Note Applet functionality was not originally supported in the release of Visual J#, therefore the upgrade report lists this as an unsupported feature. J# Browser Controls are now available to help developers migrate applet functionality.

Migrating Java Applets with J# Browser Controls

Microsoft J# Browser Controls allow developers to migrate Java applets to the .NET Framework. Java applets migrated to J# Browser Controls retain the same runtime characteristics without loss of functionality and run on end-user computers in Internet Explorer within the context of the .NET Framework. End users who wish to run J# Browser Controls hosted by a Web site must have the .NET Framework and the J# Browser Controls runtime installed on their computers. Similarly, developers who wish to upgrade Java applets to J# Browser Controls must have the J# Browser Controls runtime and the .NET Framework SDK or Visual Studio .NET 2003 installed on their development computers.

Note J# Browser Controls are not intended to run on a Java Virtual Machine. J# Browser Controls can only be run in Internet Explorer. For more information on J# Browser Controls, please refer to: http://msdn.microsoft.com/vjsharp/browsercontrols.

Migrating Java applets to J# Browser Controls is a three step process:

1. Compile the Java applet to J# Browser Controls using the Visual J# compiler.
2. Upgrade the HTML page that contains the <APPLET> tag of the applet.
3. Deploy the J# Browser Controls and HTML page to the Web server.

Compiling Java Applets to J# Browser Controls

A Java applet can be upgraded to J# Browser Controls by compiling it to a managed library using the “Visual J# Compiler” (vjc.exe): http://msdn.microsoft.com/library/en-us/dv_vjsharp/html/vjgrfVisualJCompilerOptions.asp.

Compiling Java applets to a managed library is similar to compiling any other library in Visual J#. If only the bytecode (.class files) exists for the Java applet, the “J# Binary Converter Tool” (JbImp.exe) can be used to convert the Java applet to a managed library: http://msdn.microsoft.com/library/en-us/dv_vjsharp/html/vjgrfJavaBinaryConverter.asp.

Compiling Java applets to J# Browser Controls does not require any changes to the Java applet source code because the J# Browser Controls runtime provides support for functionality equivalent to most of the JDK 1.1.4 level packages, including the **java.applet** package.

Managed libraries containing J# Browser Control are no different from other managed libraries in the .NET Framework. When a user visits a Web site hosting J# Browser Controls, the J# Browser Controls runtime will download the managed library and run the appropriate class that extends **java.applet.Applet** in Internet Explorer.

A Java applet can be compiled to a J# Browser Control either from the command prompt or from Visual Studio .NET.

Compiling a Java Applet from the Command Prompt

To compile a Java applet from the command line using the J# compiler (vjc.exe):

```
C:\AppletSources>vjc.exe /target:library /out:MyApplet.dll *.java
```

To compile a Java applet using the J# Binary Converter Tool:

```
C:\AppletSources>jbimp.exe /target:library /out:MyApplet.dll *.class
```

If you have Visual Studio .NET installed, the J# compiler (vjc.exe) and the J# Binary Converter Tool (JbImp.exe) are accessible from the Visual Studio .NET command window. If Visual Studio .NET is not installed, you must ensure that these tools are in the path of the command prompt.

When migrating Java applets that use resources, follow the steps described in "Upgrading Visual J++ 6.0 Applications That Use Resources": http://msdn.microsoft.com/library/en-us/dv_vjsharp/html/vjgrfUpgradingVisualJ60ApplicationsThatUseResources.asp and "How To: Resources in Visual J# .NET": http://www.gotdotnet.com/team/vjsharp/ResourcesHowTo.htm

Compiling a Java Applet using Visual Studio .NET

If the Java applet is a Visual J++ 6.0 Applet on HTML project, it can also be compiled to J# Browser Control using Visual Studio .NET:

1. Open the Visual J++ 6.0 project in Visual Studio .NET. This starts the Visual J# .NET project upgrade wizard.
2. Click **Next** through all steps of the upgrade wizard. The upgrade wizard converts the Visual J++ project to a Visual J# .NET Class Library project.
3. Open the upgrade report to view any issues detected during the upgrade.

Note The report will say **Applet projects are not supported**. Ignore this error and fix all other issues listed in the upgrade report before building the project. This error occurs because J# Browser Controls was shipped after Visual Studio .NET 2003 was released.

---

4. Build the project. This compiles the Java applet to a managed library.

   You cannot start J# Browser Controls in Internet Explorer from Visual Studio by clicking **Debug** and then **Start** or pressing F5. You must copy the control to a virtual directory on a Web server to run it.

Note You cannot start J# Browser Controls v 1.1 in Internet Explorer from Visual Studio by clicking **Debug** and then **Start** or pressing F5. You must copy the control to a virtual directory on a Web server to run it. J# Browser Controls v1.1b, currently in beta, removes the requirement to use a Web server to host J# Browser Controls.

For more information, refer to "Deploying J# Browser Controls": http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vjtskMigratingJavaAppletsToMicrosoftJBrowserControls.asp

and "How to: Debug J# Browser Controls": http://msdn.microsoft.com/library/en-us/dv_vstechart/html/vjtskhowtodebugjbrowsercontrols.asp.

For more information on upgrading Visual J++ 6.0 projects to Visual J# .NET, refer to "Upgrading from Visual J++ 6.0": http://msdn.microsoft.com/library/en-us/dv_vjsharp/html/vjsamUpgradingFromVisualJ60.asp.

Updating the HTML Page to use J# Browser Controls

After a Java applet is compiled to a managed library, the next step is to upgrade the HTML page hosting the Java applet. The <APPLET> tag or the Java applet <OBJECT> tag in the HTML page must be converted to the <OBJECT> tag supported by J# Browser Controls.

The J# Browser Controls runtime includes a tool called the HTML Applet to Object Tag Converter (TagConvert.exe) that automatically upgrades HTML pages to use J# Browser Controls. TagConvert.exe is located in the J# Browser Controls installation directory. The tool is used with the following syntax:

```
TagConvert [options] <source files>
```

For example:

```
TagConvert.exe MyAppletPage.html
```

The input to the tool can be any text file including files with the .html, .htm, .asp, and .aspx extensions.

The tool converts the <APPLET> tag or the Java applet<OBJECT> tag to the following J# Browser Control <OBJECT> tag:

```
<OBJECT
   CLASSID="clsid:a399591c-0fd0-41f8-9d25-bd76f632415f"
   WIDTH= pixels
   HEIGHT= pixels
   ID=browserControlName
   ALIGN= alignment
   HSPACE= pixels
   VSPACE= pixels
   VJSCODEBASE = codebaseURL
>
   <PARAM NAME = attribute1 VALUE = value>
   <PARAM NAME = attribute2 VALUE = value>
   . . .
   alternateHTML
</OBJECT>
```

In the above example, **CLASSID** is the CLASSID of the ActiveX control that downloads and executes J# Browser Controls. This exact CLASSID must be used in the J# Browser Controls <OBJECT> tag.

VJSCODEBASE is the URL of the J# Browser Control class and the managed library containing this class. The '#' token separates the managed library file name and the J# Browser Controls class name. The file name must also include the file extension. For example:

```
VJSCODEBASE =
http://www.MyWebSite.com/MyApplet/MyAppletClass.dll#MyAppletClass
```

Conversion Process

When converting HTML pages, the tool deletes the original <APPLET> tag or Java applet <OBJECT> tag and replaces them with the J# Browser Controls <OBJECT> tag. The tool also generates a JavaScript file named jbctagconvert.js in every directory that it is run in. When the updated HTML page is loaded in a browser, the updated HTML, in conjunction with the JavaScript files, renders the J# Browser Control in the browser window. The tool generates the JavaScript file to enhance the browsing experience of J# Browser Controls in the upcoming Internet Explorer update.

Note The <APPLET> tag or the Java applet <OBJECT> tag is replaced with the J# Browser Control <OBJECT> tag enclosed within comments and preceded by the <SPAN> tag. This comment block and the associated <SPAN> tag must not be deleted as they are required to display the J# Browser Control. The JavaScript file, jbctagconvert.js, generated by the tool also must not be deleted and must be present in any directory that has an HTML page hosting a J# Browser Control.

The tool creates a backup of the original files before converting the tags. Backup copies of files have the .vjsbak extension and are located in the directory of the original file. For example, index.htm will be backed up as index.htm.vjsbak.

The tool uses the values in the CODE and CODEBASE attributes of the original <APPLET> tag (or Java applet <OBJECT> tag) to create the value of the VJSCODEBASE attribute. For example:

```
CODE = "MyAppletClass"
CODEBASE = http://www.MyWebSite.com/MyApplet
```

is modified to:

```
VJSCODEBASE =
http://www.MyWebSite.com/MyApplet/MyAppletClass.dll#MyAppletClass
```

By default, the tool assumes that the name of the J# Browser Control class (for example, MyAppletClass) to be the same as the name of the DLL (MyAppletClass.dll). If the names of the J# Browser Control class and the DLL are different, you will have to modify the VJSCODEBASE attribute value accordingly.

When using this tool, it is recommended that you compile the Java applet to a managed library with the same name as the applet class. For example:

```
C:\MyAppletClassSources>vjc /target:library /out:MyAppletClass.dll *.java
```

The J# Browser Controls runtime only supports the HTTP and HTTPS protocols in the VJSCODEBASE attribute. All absolute paths in VJSCODEBASE must begin either with http:// or https://. When a relative path is specified, the J# Browser Control is loaded using the HTTP protocol. J# Browser Controls do not support loading controls from a location that is different from the DOCBASE, the location from where the HTML page is loaded. The value in VJSCODEBASE must be the same as the DOCBASE or must one of the subdirectories of DOCBASE.

Many attributes of the OBJECT are left intact during the conversion. The details of attributes converted by TagConvert are specified in the following section. Many parameters in the J# Browser Controls <OBJECT> tag are the same as the corresponding parameters of the <APPLET> or Java applet <OBJECT> tag.

Command-Line Options for TagConvert.exe

The supported command-line options for TagConvert.exe are:

`/recurse:<wildcard>`

Causes the tool to search the current directory and all subdirectories for files to convert according to the wildcard specifications. For example, to upgrade all files with the .htm and .html extensions in the current directory and its subdirectories:

```
TagConvert /recurse *.htm *.html
```

`/verbose`

Causes the tool to print the names of files changed during the conversion. The file name includes the fully qualified path to the file. The tool also prints the total number of files parsed and total number of files converted. For example, to upgrade all files with the .htm extension in the specified directory, and dump the names of files modified to the changedfiles.txt file:

```
TagConvert /verbose \AppletSources\Pages\*.htm >
changedfiles.txt
```

Attribute Mapping Between <APPLET> Tag and J# Browser Controls <OBJECT> Tag

The following table shows the mapping between the attributes in the <APPLET> tag and the attributes in the J# Browser Control <OBJECT> tag.

<APPLET> tag syntax (includes Internet Explorer extensions)	**J# Browser Controls <OBJECT> tag syntax**
CODEBASE	VJSCODEBASE
CODE	VJSCODEBASE
WIDTH	WIDTH
HEIGHT	HEIGHT
NAME	ID
ID	ID
ALIGN	ALIGN
VSPACE	VSPACE
HSPACE	HSPACE
ARCHIVE	Unsupported in current release
ALT	ALT
<PARAM>	<PARAM>
alternateHTML	alternateHTML
<PARAM NAME = FireScriptEvents VALUE = True>	Left intact but unsupported in current release
<PARAM NAME = cabbase VALUE = cabFileName> <PARAM NAME = cabinets VALUE = cabFileNames>	Left intact but unsupported in current release
<PARAM NAME = useslibrary VALUE = DUFriendlyName> <PARAM NAME = useslibrarycodebase VALUE = DUFileName> <PARAM NAME = useslibraryversion VALUE= DUVersionNumber>	Left intact but unsupported in current release
<PARAM NAME = namespace VALUE = applicationNamespace>	Left intact but unsupported in current release

In the current release, the following attributes in the <APPLET> tag are unsupported and have no equivalent attributes in the J# Browser Controls <OBJECT> tag:

- The **archive**, **cabbase** and **cabinets** attributes. Packaging J# Browser Controls into .cab, .zip, or .jar files is not supported in the current release. J# Browser Controls must be deployed to the Web server as stand-alone .dll files.

- The **FireScriptEvents** attribute. In the current release, there is no support to sink events fired by J# Browser Controls in scripts on the HTML page. J# Browser Controls v1.1b, currently in beta, introduces scripting functionality.
- The **useslibrary**, **useslibrarycodebase**, **useslibraryversion** and **namespace** attributes. The J# Browser Controls runtime does not support the Java Package Manager semantics of the MSJVM.

The attributes in the <APPLET> tag that have direct equivalents in the <OBJECT> tag can be copied as is. They have the same meaning in the J# Browser Controls <OBJECT> tag as in the original <APPLET> tag.

Deploying J# Browser Controls

To deploy J# Browser Controls, copy the managed library and updated HTML pages to the appropriate directory on the Web sever. You must copy the managed library either to the same directory as the HTML page or to a sub-directory.

When using the IIS Web server, the permissions on the virtual directory that the J# Browser Control is copied to must be set correctly; the **Execute Permissions** field on the virtual directory must be set to **Scripts only**, the default permission level of a virtual directory in IIS. Any resource files used in the J# Browser Control, such as image, audio, or data files, also need to be copied to the same location as the managed library.

Note J# Browser Controls v1.1b, currently in beta, removes the requirement to host J# Browser Controls on a Web server.

Deploying a Multi-DLL Browser Control

A J# Browser Control can be split into multiple DLL files. In such cases, the DLL containing the main J# Browser Control class must be referenced in the HTML page. The J# Browser Controls runtime will download additional DLLs as required at run time.

When deploying a multi-DLL control to a Web server, all files related to the same J# Browser Control must be copied to the same directory as standalone files and cannot be packaged as .cab, .zip, or .jar files.

Packaging Multiple J# Browser Controls into the Same Library

Multiple J# Browser Controls can also be packaged into the same managed library. In such cases, the <OBJECT> tag for each browser control must point to the same managed library but with different class names.

For example, if MyApplets.dll in the directory AppletDir contains J# Browser Controls with names `MyApplet1` and `MyApplet2`, you can refer to the two browser controls as shown in the following example:

```
VJSCODEBASE = http://www.microsoft.com/AppletDir/MyApplets.dll#MyApplet1
VJSCODEBASE = http://www.microsoft.com/AppletDir/MyApplets.dll#MyApplet2
```

Because J# Browser Controls can only be downloaded from the same location as the HTML page or one of its subdirectories, you may need to copy the managed library to multiple locations if the directories of the HTML pages differ.

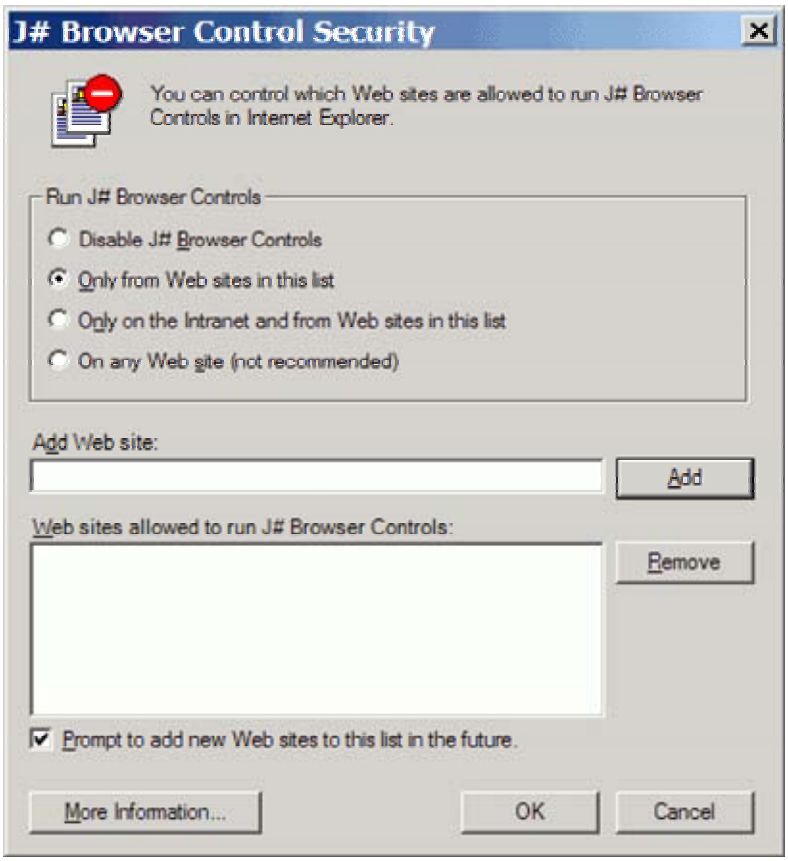
Running J# Browser Controls on User Computers

For users to run J# Browser Controls in Internet Explorer, the J# Browser Controls runtime must be installed. By default, the J# Browser Control runtime prompts end users before running controls hosted on Web pages. When a user browses to a web site that contains a J# Browser Control, the following dialog box is displayed:

J# Browser Controls are downloaded and run if the user clicks **Yes**. Clicking **No** prevents the browser control from being run.

Selecting the **Add this site to the list of sites allowed to run J# Browser Controls and don't ask me again** check box and clicking **Yes** adds the Web site to the list of sites allowed to run J# Browser Controls, and the user is no longer prompted for pages on the Web site.

The user can also use the J# Browser Controls Security Options dialog box available under Administrative Tools in Control Panel to manage the list of sites allowed to run J# Browser Controls. In Windows XP, you can access this dialog box from **Performance and Maintenance** in Control Panel. Double-click the **J# Browser Control Security** icon to open the following dialog box:



The options are:

- **Disable J# Browser Controls**
  Prevents J# Browser Controls from any Web site to run on the computer.

- **Only from Web sites in this list**
  Only J# Browser Controls from Web sites the user adds to the list are allowed to run. **This is the default setting**.

- **Only on the Intranet and from Web sites in this list**
  Only J# Browser Controls on the Intranet or from Web sites the user adds to the list are allowed to run on the computer.

- **On any Web site**
  J# Browser Controls from any site are allowed to run on the computer.

Entering the address of a Web site in the **Add Web site** text box and clicking **Add** adds the Web site to the list of sites allowed to run J# Browser Controls. Similarly, selecting a Web site from the **Web sites allowed to run J# Browser Controls** list and clicking **Remove** removes the site from the list. When a Web site is added to the list of sites allowed to run J# Browser Controls, the user is not prompted before running J# Browser Controls on the Web site.

By default, the **Prompt to add new Web sites to this list in the future** check box is selected, causing the J# Browser Controls runtime to prompt the user before running J# Browser Controls from Web sites not in the list. When this check box is cleared, J# Browser Controls from Web sites not in the list are not run and the user is not prompted.

Unsupported Features in the Current Release

The following features are unsupported in the current release of J# Browser Controls:

- **Accessing J# Browser Controls from scripts in HTML pages**
  Accessing the public methods and public variables of a J# Browser Control from scripts in HTML pages is not supported. Also, sinking events fired by J# Browser Controls in scripts using the **<param name=FireScriptEvents value=True>** attribute is not supported. J# Browser Controls v1.1b, currently in beta, will introduce scripting functionality.
- **Trust-based security**
  There is no support for trust-based security semantics as supported by the MSJVM.
- **Java Package Manager**
  The Java Package Manager functionality as supported in the MSJVM is unsupported in J# Browser Controls. J# Browser Controls do not support an object cache that can be used to install classes locally and then run them with a restricted set of permissions using permission signing. Therefore, there is also no support to install and run Distribution Units.
- **Archive files**
  Archive files are unsupported. A J# Browser Control can be split into multiple managed libraries. However, when deploying a multi-library control to a Web server, the files must be copied as stand-alone files and must not be packaged into .cab, .zip, or .jar files.
- **No designer support**
  There is no designer support in Visual Studio .NET for J# Browser Controls.

3

Migrate to .NET Using the Java Language Conversion Assistant

The Java Language Conversion Assistant (JLCA) is a tool that converts Visual J++ 6.0 projects and Java-language files to Visual C#. By converting these files to Visual C#, you can leverage existing code base and take advantage of the benefits of the .NET Framework. The JLCA does not modify your existing project, but creates a new Visual C# project based upon the original project. The JLCA is not limited to Java code that targets the MSJVM and can also convert server-side Java programs such as Java Server Pages (JSPs) and Servlets, along with J2SE and J2EE libraries. The converted Visual C# project contains all the new Visual C# code generated automatically from the existing Visual J++ or Java-language code.

To convert some of the functionality in the original project that is unavailable in Microsoft Visual C#®, JLCA creates *support classes* that duplicate the original functionality. Although every effort is made to preserve the original architecture of your application in the converted project, support classes are sometimes substantially different architecturally from the classes they emulate.

Some code in your project may not be converted automatically. Any errors, warnings, or issues generated in the course of conversion are displayed in a conversion report after the new project is generated. Unconverted code is noted in the code of the new project by comments labeled UPGRADE_TODO. You can view conversion comments on the task list. Each conversion comment contains a link to a Help topic on how to convert that code manually.

Note If you are new to Visual C#, take some time to familiarize yourself with the language before attempting this process. For more information, refer to “C# Language Tour”: http://msdn.microsoft.com/library/en-us/cscon/html/vclrfGettingStarted_PG.asp.

and “Converting Visual J++ or Java-language Project to Visual C#”: http://msdn.microsoft.com/library/en-us/dv_jlca/html/vbtskusingjavalanguageconversionassistant.asp

---

Converting Visual J++ Projects to Visual C#

To convert a Visual J++ project using the JLCA:

1. Start Visual Studio .NET.
2. On the **File** menu, point to **Open**, and click **Convert**.
3. Select **Java Language Conversion Assistant**, and click **OK**.
4. On the **Source Files** page, select **Visual J++ 6.0 project** .
5. On the **Select a project file** page, click **Browse**.
6. Browse to the correct .vjp file, and select it.

Note If the CLASSPATH directive of your .vjp file specifies .jar or .class files, they are ignored.

---

7. On the **Specify a directory for your new project** page, specify the name and directory of the new project to be created.
8. On the **Begin conversion** page, click **Next**.
9. Fix code that was not converted automatically.

Converting Java-Language Projects to Visual C#

To convert a Java-language project using the JLCA:

1. Start Visual Studio .NET
2. On the **File** menu, point to **Open**, and click **Convert**.
3. Select **Java Language Conversion Assistant**, and click **OK**.
4. On the **Source Files** page, click **A directory containing the project's files**.
5. On the **Select source directory** page, click **Browse**.
6. Browse to the correct project, and select it.

Note You will not see the files in the directory you selected, but all .jav and .java files in it are converted. All other files in the directory are ignored.

---

7. On the **Configure your new project** page, specify the following:
   - Name of the project to be created.
   - Directory in which any additional files for your project are located.
   - Output type of the project.
8. On the **Specify a directory for your new project** page, specify the name and directory of the new project to be created.
9. On the **Begin your conversion** page, click **Next**.
10. Fix code that was not converted automatically.

Manually Upgrading Unconverted Code

After the JLCA converts your Visual J++ project or Java-language files, the new Visual C# project might contain code that was not converted automatically. Comments are inserted in the code of the new project to help you convert that code to Visual C# manually.

Comment Type	Description
UPGRADE_TODO	Code that could not be converted automatically
UPGRADE_WARNING	Code that may be problematic
UPGRADE_NOTE	Code that may exhibit a different behavior from the original code

There may also be some compiler errors that must be corrected before the application compiles. Each comment contains a brief description of the issue and a link to a Help topic about how to convert the code. To manually upgrade unconverted code:

1. After running the Java Language Conversion Assistant wizard, open your project in Visual Studio .NET.
2. On the **View** menu, select **Show Tasks**, and then either **Comment** or **All**. The conversion notes are displayed on the **Task List**.
3. On the **Task List**, double-click the first conversion note. The focus moves to the location of that comment in the code.
4. Examine the note and the code it references. For more information on correcting that error manually, click the link in the conversion note to display a Help topic for that error.

   Some of the conversion notes in the converted code for JSP pages are not linked. In this case you can cut and paste the note into the Help **Search** box.
5. Upgrade the code manually.
6. Repeat Steps 3 to 5 until all conversion notes are addressed.

Note For more information on manually upgrading, refer to “Manually Upgrading Unconverted Code”. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_jlca/html/vbtskmanuallyupgradingnonconvertedcode.asp

For a list of errors, warnings, and issues, refer to “Java Language Errors, Warnings, and Issues”: http://msdn.microsoft.com/library/en-us/dv_jlca/html/vberrjavalanguageerrorswarningsissues.asp.

For a list of Help topics for JSP classes and methods that cannot be converted automatically, refer to “Javax.servlet.jsp Error Messages”: http://msdn.microsoft.com/library/en-us/dv_jlca/html/vberrjavaxservletjsperrormessages.asp

---

Converting Applets using the JLCA

Converting applets using the JLCA is a two-step process. The first step is to convert the actual applet code to C# using the JLCA. The second step is to convert the HTML page which references the applet to the .NET equivalent. To convert the applet code to C#, use the exact same process as when converting an application. The applet code is converted as a class library project.

After fixing any conversion issues you may have in the converted applet, you will have a Windows Form Control (WFC). You can host converted user controls in Internet Explorer just as you would an applet. Hosted controls are declared in HTML pages with the <OBJECT> tag instead of the <APPLET> tag. Use the classid attribute to identify the control by specifying the path to the control and the fully qualified name of the control, separated by the pound sign (#):

```
<OBJECT id="myControl"
classid="http:ControlLibrary1.dll#ControlLibrary1.myControl"
VIEWASTEXT></OBJECT>
```

For the control to be displayed properly, the .dll file containing the control must either be in the same virtual directory as the Web page displaying it or installed in the global assembly cache.

Upgrading the HTML Page

The WFC does not support the deprecated <APPLET> tag or the <OBJECT> tag syntax used by the MSJVM and other Virtual Machines to run Java applets in Internet Explorer. In order to run your converted applet in the browser, first change your Web page to utilize the WFC <OBJECT> tag syntax. This is done either automatically or manually.

The simplest way to upgrade your Web page references is to allow the JLCA to automatically update the references. If you have a Visual J++ applet on HTML project and choose to convert using the Visual J++ file, the JLCA automatically parses the associated HTML file and changes all <APPLET> tag references as necessary. You can then copy and paste this tag text into other pages that reference the same applet.

If you don’t have a Visual J++ project for your applet or don’t want to use it for conversion, you can also convert the HTML references by hand. Here is an example of the <APPLET> tag syntax, as used in HTML pages including MSJVM extensions:

```
<APPLET
CODE = appletFile
CODEBASE = codebaseURL
WIDTH = pixels
HEIGHT = pixels
VSPACE = pixels
HSPACE = pixels
ALIGN = alignment
ID = appletInstanceName
ARCHIVE = archiveList
ALT = alternateText
>
<PARAM NAME = FireScriptEvents VALUE = True>
<PARAM NAME = cabbase VALUE = cabFileName>
<PARAM NAME = cabinets VALUE = cabFileNames>
```

```
<PARAM NAME = useslibrary VALUE = DUFriendlyName>
<PARAM NAME = useslibrarycodebase VALUE = DUFileName>
<PARAM NAME = useslibraryversion VALUE= DUVersionNumber>
<PARAM NAME = namespace VALUE = applicationNamespace>
<PARAM NAME = appletAttribute1 VALUE = value>
<PARAM NAME = appletAttribute2 VALUE = value>
. . .
alternateHTML
</APPLET>
```

The syntax as supported by Windows Forms Controls is:

```
<OBJECT
CLASSID="http:<OutputFileName>#<ClassName>"
WIDTH= pixels
HEIGHT= pixels
ID=browserControlInstanceName
ALIGN= alignment
HSPACE= pixels
VSPACE= pixels
>
<PARAM NAME = attribute1 VALUE = value>
<PARAM NAME = attribute2 VALUE = value>
. . .
alternateHTML
</OBJECT>
```

In the above WFC syntax example **CLASSID** is the CLASSID string comprised of the output file name and the name of the Windows Form Control class itself. Both of these settings can be found in the Project Properties dialog. To open the Project Properties dialog, right-click your converted applet project in the Solution Explorer and select “Properties”.

4

Other Rendering Technologies

Alternate rendering technologies including DHTML, ECMAScript, VBScript, JScript, and Flash, among others, are also viable options. These technologies can implement most simple navigational items (for example, menus, rollovers, simple calculators, and chat clients), and are an option for converting low to medium complexity applets dedicated to navigation and user interface tasks.

Note Although Microsoft cannot vouch for the security and reliability of solutions offered by other companies, non-Microsoft solutions are available. Customers choosing to explore such solutions should engage in sufficient testing prior to pursuing this migration path.

DHTML and Client-side Scripting

In this guide, the acronym DHTML (Dynamic HTML) is used broadly to include combinations of CSS, scripting, and HTML in the Document Object Model (DOM). The advantage of DHTML is that used carefully it is a flexible low-overhead, multi-browser, multi-platform solution. Because there is no plug-in or download required, interpretation and rendering of the navigational aid occur within the browser, rather than in a virtual machine or plug-in. Many Web sites already provide public domain or low-cost code for implementing simple interface features such as drop-down menus, text scrollers, clocks, rollover buttons, and complicated utilities such as calculators, chat clients, and editors.

The disadvantage of DHTML is that it must be used with care to ensure cross-browser compatibility and must be tested on the appropriate platforms and browsers. Because DHTML relies heavily on the features of the scripting language and the implementation of style sheets, there are issues with how well DHTML works on each browser and each platform. Version 6.0 browsers handle style sheets reasonably well, and there is a core set of functions in the scripting language (ECMAScript and extensions) that are reliable across most browsers.

The following are links to sites where you can view samples of various effects implemented in DHTML. If those effects are similar to your goals, the samples will provide a starting point for your implementations.

A discussion of DHTML in Internet Explorer:
http://msdn.microsoft.com/ie

Variety of examples, including a clock and a chat client:
http://msdn.microsoft.com/downloads/samples/internet/default.asp

Non-Microsoft sites:
http://www.codetoad.com/dhtml/
http://simplythebest.net
http://www.dynamicdrive.com/dynamicindex1.

Note Microsoft cannot guarantee or warrant that any of these samples will work correctly in your environment. Test all samples sufficiently if you are implementing something similar to ensure your needs are met.

ECMAScript and JScript

JavaScript was originally Netscape's version of ECMAScript (European Computer Manufacturers Association), a standard language specification, whereas JScript is Microsoft's version of ECMAScript. JScript .NET is the next generation of Microsoft's implementation of ECMA 262 language and combines the existing feature set of JScript with the best features of class-based languages.

In many cases the functionality of Java applets can be provided by using the ECMAScript language. An Internet search for JavaScript will yield a variety of resources, many with ready-to-use scripts that could potentially replace Java applet code.

Note There is no direct conversion method available for new scripting languages. It is necessary to rewrite the Java code into the appropriate language. For more information on Windows scripting technologies, including JScript and VBScript, refer to: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vtoriMicrosoftWindowsScriptTechnologies.asp.

Flash

Flash is another rendering technology that can provide the functionality of certain Java applets that currently are dependent on the MSJVM. In some cases Flash may also extend the function of these applets. Flash is a client-based plug-in that allows the rendering of interactive graphics and animation.

As with any other rendering technology, migrating to Flash is not an automated process and will include extensive rewriting of existing code and may require extensive knowledge of the associated scripting language (ActionScript). Flash is not a Microsoft supported technology and will therefore require you to seek support options from the appropriate vendor if you choose this technology as your migration solution.

Note For more information on Flash, its features, and implementation considerations, please refer to: http://www.macromedia.com/software/flash/.

5

Switch to Another JRE

Note Microsoft does not guarantee or warrant that alternate JREs will work correctly in your environment. Test all JREs sufficiently during implementation to ensure that all of your needs are met.

Several vendors have announced that they will ship an alternate Java runtime environment (JRE) on their computers when Microsoft is no longer able to support the MSJVM. Whether you adopt this solution depends upon your business needs, including portability, reliability, versions, and maintenance.

Although vendors represent that their JREs comply with relevant Java specifications, all JREs are not necessarily equivalent for your needs. Not all implementations of Java specifications are identical, and these differences may cause difficulties for you.

Current Visual J++ code may need to be ported to work with an alternate JRE. The use of Microsoft extensions is easy to find in current J++ source files; you can find Java source files that make use of the @dll directive by using a CMD.EXE command of the form:

```
find /I "@com" c:\src\*.*
```

Similar commands will find other extensions, such as the @com and @security directives. Once you have identified the use of the extensions, you can gauge the effects of removing them.

The use or removal of the Microsoft extensions makes no syntactic difference to the Java code because the directives are embedded in comments. It is possible to turn the extensions off with options to the javac command, so you can easily compare the effect of running the application with and without the extensions.

Alternate JREs are deployed throughout your organization using the same means by which you normally install new applications on user machines. For more information on properly installing an alternate JRE please refer to vendor documentation. If you choose to install an alternate JRE, any problems you experience with Java support will not be addressed by Microsoft, and are subject to relevant support and license agreements with the appropriate vendor of the alternate JRE.

The following is a partial list of available alternate JREs and is not intended to represent **all** available alternate JREs:

Sun JRE for Windows:
http://www.java.com/en/download/windows_automatic.jsp

IBM JRE (as part of the IBM WebSphere SDK for Web Services):
http://www-106.ibm.com/developerworks/webservices/wsdk/

BEA WebLogic JRockit:
http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/jrockit