



J A D E TM

.NET Developer's Reference

VERSION 6.3



Copyright © 2009
Jade Software Corporation Limited
All rights reserved

Jade Software Corporation Limited cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages, or loss of profits. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Jade Software Corporation Limited.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Copyright © 2009 Jade Software Corporation Limited.
All rights reserved.

JADE is a trademark of Jade Software Corporation Limited. All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

For details about other licensing agreements for third party products, you must read the JADE **ReadMe.txt** file.

Contents

Before You Begin	v
Who Should Read this Reference	v
What's Included in this Reference	v
Related Documentation	v
Conventions	vi
Chapter 1 Using the .NET Framework	7
Overview	7
Exposing JADE Classes	7
.NET Class Library Framework	8
.NET and JADE	8
How a .NET Application Connects to JADE	8
Chapter 2 .NET Exposure	9
Overview	9
.NET Exposures	10
Configuration File.....	11
.NET Runtime Environment.....	13
Exposed JADE Classes	14
Exposed JADE Properties.....	15
Exposed JADE Methods	15
JADE Method Parameter Usage	16
Exposed JADE Class Constants	16
Application and Global Classes.....	17
Collections.....	17
Chapter 3 Developing Applications in .NET to Use JADE Classes	19
Overview	19
Using JADE Classes in .NET.....	19
Signing On to JADE	20
Signing Off from JADE	21
Pool of JadeDBProcess Instances	21
Accessing JADE Objects.....	22
Creating Objects.....	23
Deleting Objects.....	23

Chapter 3	Developing Applications in .NET to Use JADE Classes, continued	
	Handling JADE Lock Exceptions.....	23
	Notifications	24
	Exceptions	25
	Creating Web Services Using JADE Objects.....	25
	JADE Sign On	26
	Connecting to a JADE Process.....	26
Index		28

Before You Begin

The *JADE .NET Developer's Reference* is intended as a major source of information when using the JADE .NET class library to develop .NET applications accessing JADE Database Objects.

Who Should Read this Reference

The main audience for the *JADE .NET Developer's Reference* is expected to be .NET developers using the .NET Framework.

What's Included in this Reference

The *JADE .NET Developer's Reference* has three chapters.

Chapter 1	Gives an overview of the framework for developing .NET applications to connect to a JADE system
Chapter 2	Gives a reference to the .NET classes resulting from exposing JADE classes
Chapter 3	Gives a reference to the jomdotnet library and the API required to use the exposed classes

Related Documentation

Other documents that are referred to in this reference, or that may be helpful, are listed in the following table, with an indication of the JADE operation or tasks to which they relate.

Title	Related to...
JADE Database Administration Guide	Administering JADE databases
JADE Developer's Reference	Developing or maintaining JADE applications
JADE Development Environment Administration Guide	Administering JADE development environments
JADE Development Environment User's Guide	Using the JADE development environment
JADE Encyclopaedia of Classes	System classes (Volumes 1 and 2), Window classes (Volume 3)
JADE Encyclopaedia of Primitive Types	Primitive types and global constants
JADE Installation and Configuration Guide	Installing and configuring JADE
JADE Initialization File Reference	Maintaining JADE initialization file parameter values
JADE Object Manager Guide	JADE Object Manager administration
JADE Replication Framework User's Guide	Replicating selective activity between physically different but logically similar JADE systems
JADE Synchronized Database Service (SDS) Administration Guide	Administering JADE Synchronized Database Services (SDS), including Relational Population Services (RPS)
JADE Thin Client Guide	Administering JADE thin client environments
JADE Web Application Guide	Implementing, monitoring, and configuring Web applications

Conventions

The JADE .NET *Developer's Reference* uses consistent typographic conventions throughout.

Convention	Description
Arrow bullet (>)	Step-by-step procedures. You can complete procedural instructions by using either the mouse or the keyboard.
Bold	Items that must be typed exactly as shown. For example, if instructed to type foreach , type all the bold characters exactly as they are printed. File, class, primitive type, method, and property names, menu commands, and dialog controls are also shown in bold type, as well as literal values stored, tested for, and sent by JADE instructions.
<i>Italic</i>	Parameter values or placeholders for information that must be provided; for example, if instructed to enter <i>class-name</i> , type the actual name of the class instead of the word or words shown in italic type. Italic type also signals a new term. An explanation accompanies the italicized type. Document titles and status and error messages are also shown in italic type.
Blue text	Enables you to click anywhere on the cross-reference text (the cursor symbol changes from an open hand to a hand with the index finger extended) to take you straight to that topic. For example, click on the " How a .NET Application Connects to JADE " cross-reference to display that topic.
Bracket symbols ([])	Indicate optional items.
Vertical bar ()	Separates alternative items.
Monospaced font	Syntax, code examples, and error and status message text.
ALL CAPITALS	Directory names, commands, and acronyms.
SMALL CAPITALS	Keyboard keys.

Key combinations and key sequences appear as follows.

Convention	Description
KEY1+KEY2	Press and hold down the first key and then press the second key. For example, "press SHIFT+F2" means to press and hold down the SHIFT key and press the F2 key. Then release both keys.
KEY1,KEY2	Press and release the first key, then press and release the second key. For example, "press ALT+F,X" means to hold down the ALT key, press the F key, and then release both keys before pressing and releasing the X key.

In this document, the term Microsoft *Windows* refers to Windows 2003 Server, Windows Vista, Windows XP, Windows 2000, or Windows CE. When there are differences between the versions of Microsoft Windows, the specific version of Microsoft Windows is stated. This also applies to Linux, which is a specific version of UNIX developed by SUSE or Red Hat. The term *UNIX* is used when an issue is generic to all versions of UNIX and the term *Linux* is used if the issue is specific to the SUSE or Red Hat implementation of UNIX.

With the exception of the **jade.exe** program, when referring to Windows program executables in this document, the **.exe** file suffix is omitted; for example, **jadclient** refers to **jadclient.exe** on Windows and **jadclient.sh** on UNIX. Similarly, the Windows **.dll** (Dynamic Link Library) and UNIX **.so** (shared object library) file suffixes are omitted. For example, **jomos** refers to **jomos.dll** (Windows) and **libjomos.so** (Linux).

This chapter covers the following topics.

- [Overview](#)
- [Exposing JADE Classes](#)
- [.NET Class Library Framework](#)
- [.NET and JADE](#)
- [How a .NET Application Connects to JADE](#)

Overview

The JADE .NET class library enables .NET developers to access classes and their associated properties and methods in the JADE database.

This is achieved by generating a set of classes in .NET that act as proxies for the actual JADE classes. Accessing the proxy classes is similar to the use of the actual classes in JADE.

The .NET library containing the classes is built by exposing the classes (and their methods and properties) of interest in JADE. The wizard used to expose JADE features generates C# class files that you can then build into a re-usable .NET class library.

Exposing JADE Classes

You can use .NET to write the application-related components of your system and access JADE database-related components. The approach consists of the following steps.

1. Use JADE to define the classes for your .NET application.
2. Define a .NET exposure and generate C# source files as the starting point for building the application. For details, see [Chapter 19](#) of the *JADE Development Environment User's Guide*.

The C# class definitions contain the properties and methods defined in the exposure that are required to access them in the JADE database.

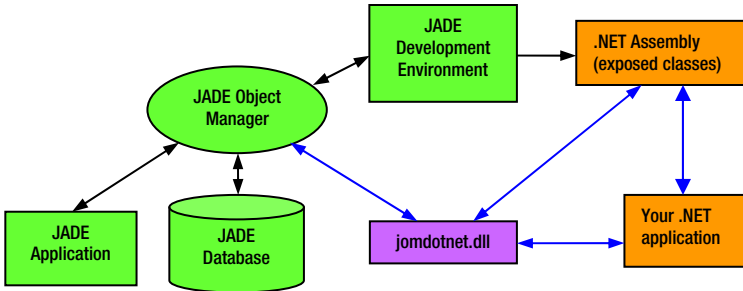
3. Use the generated C# Visual Studio project file (or create your own) to build a Dynamic Link Library (assembly) that contains the proxy classes.
4. Add a reference to the assembly, generated in the previous step of this instruction, to your own project.

5. Add instructions to your .NET code relating to JADE connection, JADE class access, and transaction control.
6. Build and run your .NET project.

.NET Class Library Framework

To use the .NET class library, you must have the tools to build .NET projects; for example, Microsoft Visual Studio 2005 or Microsoft Visual Studio 2008.

The .NET Class Library is implemented as an assembly that provides a set of .NET classes released in a single (**jomdotnet.dll**) file. The following diagram shows the relationship between these components.



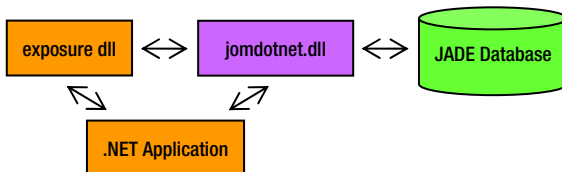
.NET and JADE

The JADE object model includes a class hierarchy that is similar to that of .NET.

This common object-oriented approach means there is a natural fit between JADE and .NET, enabling .NET objects to be modeled from the JADE database.

How a .NET Application Connects to JADE

A .NET runtime application connects to a JADE system as a standard JADE client using the **jomdotnet.dll** library file on Windows. For more details about coding that causes the sign on, see “[Signing On to JADE](#)”, in Chapter 3.



The **jomdotnet.dll** file is one of the files supplied from JADE as part of a standard JADE client installation, and is typically located in the **bin** directory; for example, **c:\jade\bin**.

This chapter covers the following topics.

- [Overview](#)
- [.NET Exposures](#)
 - [Configuration File](#)
- [.NET Runtime Environment](#)
- [Exposed JADE Classes](#)
- [Exposed JADE Properties](#)
- [Exposed JADE Methods](#)
 - [JADE Method Parameter Usage](#)
- [Exposed JADE Class Constants](#)
- [Application and Global Classes](#)
- [Collections](#)

Overview

JADE classes, properties, methods, and class constants are exposed using the .NET Exposure Wizard in the JADE development environment. For details, see [Chapter 19](#) of the *JADE Development Environment User's Guide*.

The exposure generates C# class files that correspond to the exposed JADE classes. These exposed classes are then built into a .NET class library. You can then use this class library in .NET applications to access, create, and update JADE objects in the JADE database.

The generated class library and your application can use the .NET JADE Application Programming Interface (API) contained in the **jomdotnet.dll** file.

Any .NET application or library using the .NET JADE API must include the **jomdotnet.dll** file as a reference. This file contains the namespaces listed in the following table.

Namespace	Contains ...
JadeWorld.Jade.RootSchema	The classes corresponding to JADE RootSchema classes. The Application and Global classes enable access to methods on user-defined subclasses of these classes. The System , Node , and Process classes enable access to methods on the corresponding JADE RootSchema classes. The other classes (that is, ObjectArray , ObjectSet , MemberKeyDictionary , and ExtKeyDictionary) are used when defining collections and accessing them at run time.
JadeWorld.Jade.Interop	The JadeDBConnection , JadeDBProcess , and related classes, which are the primary classes that implement the JADE .NET API. These classes allow connection to the JADE system, accessing the object, deleting objects, transaction control, and so on.
JadeWorld.Jade.ObjectCache	Internal package for caching JADE objects.

.NET Exposures

The **Exposures** command from the Browse menu in the JADE development environment enables you to define a .NET exposure. For details about using the .NET Exposure wizard, see “Using the .NET Exposure Wizard”, in Chapter 19 of your *JADE Development Environment User's Guide*.

A .NET *exposure definition* is a collection of JADE classes, methods, and properties that are generated to create a .NET class library for an existing JADE schema.

When a JADE .NET exposure is generated, the following files are created.

File	Contains ...
<i>Exposure-name.csproj</i>	A simple C# project file that includes all class files and creates a C# library with the namespace <i>Exposure-name</i> .
<i>Exposure-name.config</i>	An example application configuration file that can be used without modification for a test application or to provide code fragments for production configuration files.
<i>Class-name.cs</i>	For each exposed class, a C# class file is created containing the class definition in the namespace <i>Exposure-name</i> .

You can open the ***Exposure-name.csproj*** C# project file using Microsoft Visual Studio 2005 or Visual Studio 2008 and create the library file ***Exposure-name.dll***, which defines the namespace *Exposure-name* containing the exposed classes. You can then include this DLL as a reference in a .NET application project, to enable access to the JADE classes.

.NET DLL files or executables to be used with:

- 32-bit JADE executables must be built for the x86 (32-bit) platform.
- 64-bit JADE executables must be built for the x64 (64-bit) platform.

Configuration File

The following is an example of a generated configuration file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="jade"
      type="JadeWorld.Jade.Configuration.JadeConfigurationSection,
        jomDotNet" />
  </configSections>
  <jade>
    <jomdotnet.logging directory="g:\jade\logs"
      fileName="jomdotnet"
      level="None"/>
  <connections>
    <connection name="MyConnection"
      path="g:\jade\system"
      ini="g:\jade\system\jade.ini"
      server="SingleUser">
      <applications>
        <application name="App1"
          schema="ErewhonInvestmentsModelSchema"
          app="ErewhonInvestmentsModelSchema"
          user="ErewhonUser"
          password=""
          numberProcesses="1">
          <classExposureAssemblies>
            <assembly name="ErewhonTesting"/>
          </classExposureAssemblies>
        </application>
      </applications>
    </connection>
  </connections>
</jade>
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="jomDotNet"
        publicKeyToken="27227940b4239a69"
        culture="neutral" />
      <codeBase version="6.3.0.0" href="g:\jade\bin\jomDotNet.dll"/>
    </dependentAssembly>
  </assemblyBinding>
</runtime>
</configuration>
```

Within the **<configSections>** element there is a **jade** section, which defines the JADE configuration and which is required if you intend using JADE configuration parameters. Within the **jade** section are a number of options that enable you to customize your application.

The `<jomdotnet.logging>` element customizes `jomdotnet` logging, which provides a trace of internal actions being performed at a level of detail specified by the `level` attribute. The attributes of the `<jomdotnet.logging>` element are listed in the following table.

Property Name	Description
directory	Full path to the log file directory
filename	Name of the log file
level	None , Minimal , or Verbose

The `<connection>` element provides the JADE connection parameters. You can specify a number of different `<connection>` elements within a `<connections>` element. The attributes of the `<connection>` element are listed in the following table.

Property Name	Description
name	A key to access this section from the <code>JadeDBConnection</code> constructor
path	JADE system file path
ini	JADE initialization file name
server	SingleUser or MultiUser

The `<application>` element provides the JADE sign-on parameters. You can specify a number of different `<application>` elements within an `<applications>` element. The attributes of the `<application>` element are listed in the following table.

Property Name	Description
name	A key to access this section from the <code>JadeDBConnection</code> constructor
schema	JADE schema name
app	JADE application name
user	User name
password	User password
numberProcesses	Number of instances of the application to create; that is, the size of the application pool

The `<classExposureAssemblies>` element provides a list of assemblies that define JADE *exposures*. The .NET interface provides a mapping between .NET classes defined in an exposure and JADE objects in a JADE database.

The DLLs that define exposed classes are listed in the `<classExposureAssemblies>` element so that they can be easily located. If no DLLs are defined, `jomdotnet` scans for loaded assemblies that define types derived from `JadeObject`.

The following table lists the attribute of the `<classExposureAssemblies>` element.

Property Name	Description
name	Name of exposure DLL

For details about the `<runtime>` element, see the following section, [“.NET Runtime Environment”](#).

.NET Runtime Environment

There are several ways to specify the location of the JADE DLL files used by your application. You can use a configuration file to specify a `<codebase>` element for `jomdotnet.dll` or if no such file is specified, the standard Windows dynamic link library search order applies.

The configuration file must have the same name as your application with a `.config` extension. The contents of a configuration file are shown in the following example. For more details, consult your .NET documentation.

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="jomDotNet"
          publicKeyToken="27227940b4239a69"
          culture="neutral" />
        <codeBase version="6.3.0.0" href="G:\Jade63\bin\jomDotNet.dll"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

As the `jomdotnet.dll` needs to load other JADE DLL files from the installed JADE `bin` directory, this directory must be located in the search path in one of the following ways if you do not have a configuration file.

- Build the .NET executable (and any .NET-dependent DLL files) in the JADE `bin` directory and run the executables from that directory.
- Set the JADE `bin` directory as the *current directory* when the .NET executable is run.
- Add the JADE `bin` directory to the Windows **PATH** environment variable.

Alternatively, you could also copy a subset of the JADE DLL files to your .NET project `bin` directory, which is a more-likely scenario when developing a Web application. However, the DLL files that are required depend on the JADE methods to be invoked; for example, as a starting point (to log on) you could copy the following DLLs.

- `jom*.dll`
- `jadcnet.dll`
- `jlnttp.dll`
- `jrnreader.dll`
- `libeay32.dll`

If you copy JADE DLL files for a Web application, do not include the following managed DLL files, which the Web server will attempt to preload.

- `jaddotnetimp.dll`
- `jadedotnet.dll`
- `jadedotnetthin.dll`
- `jadedotnetdesignerloader.dll`
- `jadewpf.dll`

Exposed JADE Classes

Each exposed JADE class is:

- Generated as a single *Class-name.cs* C# class file
- Generated in the namespace *Exposure-name*
- Inherits from **JadeObject** as the base class
- Contains a constructor to create an object as a transient or persistent JADE object
- Contains static methods to return the first and the last instances of the JADE class
- Contains exposed properties, methods, and constants

If the exposed JADE class is a subclass of another exposed JADE class, the class hierarchy is retained in the C# definition; for example, if **SubClass1** is a subclass of **Class1** in the JADE definition, the C# class **SubClass1** is derived from **Class1**.

If the exposed JADE class has been renamed in the exposure (that is, the JADE class name differs from the C# class name), an annotation is added to the C# class to define the JADE class name, as shown in the following code fragment.

```
[JadeClassName("TypeTest")]
```

The following example shows an exposed JADE class.

```
public class TC1 : JadeObject
{
    public TC1() : base() { }
    public TC1(ClassPersistence lifetime) : base(lifetime) {}

    /// <summary>
    /// Gets the <see cref="TC1"/> object that represents the first
    /// instance of this type in the Jade Database.
    /// </summary>
    /// <returns>A <see cref="TC1"/> representing the first instance
    /// of this type in the Jade Database.</returns>
    public static TC1 FirstInstance()
    {
        return JadeObject.FirstJadeInstance(typeof(TC1)) as TC1;
    }

    /// <summary>
    /// Gets the <see cref="TC1"/> object that represents the last
    /// instance of this type in the Jade Database.
    /// </summary>
    /// <returns>A <see cref="TC1"/> representing the last instance
    /// of this type in the Jade Database.</returns>
    public static TC1 LastInstance()
    {
        return JadeObject.LastJadeInstance(typeof(TC1)) as TC1;
    }

    #region Jade Properties
    #region Jade Methods
    #region Jade Constants
}
```

Exposed JADE Properties

Each exposed JADE property is defined in the C# class in the **Jade Properties** region. Each property has an accessor and mutator (that is, a **get** method and a **set** method).

The data type of the property is based on the JADE type. The conversions for JADE primitive types are listed in the following table.

JADE Type	.NET Type
Binary	Byte[]
Boolean	Boolean
Byte	Byte
Character	Char
Date	Date (defined in JadeWorld.Jade.RootSchema)
Decimal	Decimal
Integer	Int32
Integer64	Int64
MemoryAddress	IntPtr
Point	Point (defined in JadeWorld.Jade.RootSchema)
Real	Double
String	String
StringUtf8	String
Time	Time (defined in JadeWorld.Jade.RootSchema)
TimeStamp	TimeStamp (defined in JadeWorld.Jade.RootSchema)
TimeStampInterval	TimeSpan
TimeStampOffset	TimeStampOffset (defined in JadeWorld.Jade.RootSchema)

The following example shows an exposed JADE property.

```
public Boolean Bool
{
    get {JadeParamBoolean propValue = new JadeParamBoolean(Usage.Output);
        GetProperty("bool", propValue);
        return propValue.Value;}
    set {JadeParamBoolean propValue = new JadeParamBoolean(value);
        SetProperty("bool", propValue);}
}
```

Exposed JADE Methods

Each exposed JADE method is defined in the C# class in the **Jade Methods** region.

The data type of the return value and parameters is based on the JADE type. The conversions for JADE primitive types are the same as those for properties. For details, see [“Exposed JADE Properties”](#), in the previous section.

JADE Method Parameter Usage

In the signature of a JADE method, the type of each parameter must be declared along with the **Usage** value, which can be **Usage.Constant**, **Usage.Input**, **Usage.IO** (combination of input and output), or **Usage.Output**.

If you do not specify a usage, a default of **Usage.Constant** is assumed.

The parameter usages are as follows.

- For **Usage.Constant** and **Usage.Input** parameters, the parameter value specified in the method call is passed to the corresponding parameter in the called method.

Note You cannot assign to a **Usage.Constant** or **Usage.Input** parameter.

- For **Usage.Output** parameters, the value is passed in the reverse direction, from the parameter of the called method back to the corresponding parameter of the caller. This copying back of the parameter value occurs when the called method returns.
- For **Usage.IO** parameters, the parameter value is passed in both directions, as follows.
 - From the caller to the called method when the method begins
 - From the called method back to the caller when it returns

The following is an example of an exposed JADE method. The parameter **b1** has a **Usage** of **Usage.Constant**. The parameter **b2** has a usage of **Usage.IO**. The parameter **b3** has a usage of **Usage.Output**.

```
public Int32 TestIntegerMeth(Int32 b1, ref Int32 b2, out Int32 b3)
{
    JadeParamInteger retiParam = new JadeParamInteger(Usage.Output);
    JadeParamInteger jadeParam1 = new JadeParamInteger(b1);
    JadeParamInteger jadeParam2 = new JadeParamInteger(b2, Usage.IO);
    JadeParamInteger jadeParam3 = new JadeParamInteger(Usage.Output);
    JadeParam[] paramList = {jadeParam1, jadeParam2, jadeParam3};
    JadeParamParamList jadeParams =
        new JadeParamParamList(paramList, Usage.IO);
    SendMsg("testIntegerMeth", retiParam, jadeParams);
    b2 = jadeParam2.Value;
    b3 = jadeParam3.Value;
    return retiParam.Value;
}
```

Exposed JADE Class Constants

Each exposed JADE class constant is defined in the C# class in the **Jade Constants** region.

The data type of the constant is based on the JADE type. The conversions for JADE primitive types are the same as those for properties. For details, see “[Exposed JADE Properties](#)”, earlier in this chapter.

Application and Global Classes

User-defined subclasses of the JADE **Application** and **Global** classes can be exposed in .NET.

The .NET definition of these classes differs from the normal JADE class, as there is one instance of **Application** and one instance of **Global** in a JADE application. You cannot create instances of these classes. These classes do not contain a constructor but they contain a static method to return the single instance.

An exposed **Application** or **Global** JADE class is:

- Generated as a single *Class-name.cs* C# class file
- Generated in the *Exposure-name* namespace
- Contains a static method to return the single instance of the JADE class
- Contains exposed properties, methods, and constants

The following is an example exposed JADE **Application** class

```
public class CSharpUnitTest1 : JadeWorld.Jade.RootSchema.Object
{
    public CSharpUnitTest1() : base() { }

    public static CSharpUnitTest1 GetInstance()
    {
        JadeProcess jp = JadeProcess.GetInstance();
        return jp.App as CSharpUnitTest1;
    }
    #region Jade Properties
    #region Jade Methods
    #region Jade Constants
}
```

Collections

Collection classes are derived from a subclass of **Jadeworld.Jade.RootSchema.Collection** listed in the following table, to inherit the required behavior.

Collection	Description
Array	Array of JADE objects
Set	Set of JADE objects
MemberKeyDictionary	Dictionary of JADE objects with one or more keys that are in the collection member class
ExtKeyDictionary	Dictionary of JADE objects with one or more keys that are externally supplied

Each class, which implements the .NET **IEnumerable** interface, can be used like a regular .NET collection.

The following example shows an exposed JADE **Collection** class called **TC1Dict** that contains JADE objects of type **TC1**.

```
public class TC1Dict : TC1Dict<TC1>
{
}

public class TC1Dict<T> : MemberKeyDictionary<T> where T : TC1, new()
{
    public TC1Dict() : base() { }
    <collection methods>
    #region Jade Properties
    #region Jade Methods
    #region Jade Constants
}
}
```

In the following example, **tc2** contains a reference **allTC1s** of type **TC1Dict**.

```
foreach (TC1 tc1 in tc2.AllTC1s)
{
    tc1.someMethod();
}
```

When indexing a JADE array in .NET, the .NET convention of zero-based arrays is used; that is, the element at position zero (**0**) is the first element in the array.

In the following example, a C# class called **SomeClass** has an **IntegerArray** property called **MyIntegerArray**.

```
SomeClass obj = SomeClass.FirstInstance();
Int32 i = obj.MyIntegerArray[0]; // returns first element of array
Int32 j = obj.MyIntegerArray.At(0); // also returns first element of array
```

Chapter 3

Developing Applications in .NET to Use JADE Classes

This chapter covers the following topics.

- [Overview](#)
- [Using JADE Classes in .NET](#)
 - [Signing On to JADE](#)
 - [Signing Off from JADE](#)
 - [Pool of JadeDBProcess Instances](#)
 - [Accessing JADE Objects](#)
 - [Creating Objects](#)
 - [Deleting Objects](#)
 - [Handling JADE Lock Exceptions](#)
 - [Notifications](#)
 - [Exceptions](#)
 - [Creating Web Services Using JADE Objects](#)
 - [JADE Sign On](#)
 - [Connecting to a JADE Process](#)

Overview

This chapter documents classes in the **jomdotnet** library and the application-related .NET code required to use the JADE classes documented in Chapter 2.

Using JADE Classes in .NET

The following classes in the **JadeWorld.Jade.Interop** namespace provide the link between a .NET application and a JADE system.

- **JadeDBConnection**
- **JadeDBProcess**

To access these classes in a C# project, add:

- `jomdotnet.dll` to the references in the C# project
- The following `using` clause to simplify access.

```
using JadeWorld.Jade.Interop;
```

Signing On to JADE

An instance of the `JadeDBConnection` class is used to connect to JADE. There are two overloaded constructors for the `JadeDBConnection` class, as follows.

```
JadeDbConnection(String connection-name, String application-name)  
JadeDBConnection(JadeConnectionParams connectionParams)
```

The `JadeConnectionParams` class has a constructor that enables JADE Object Manager initialization and sign-on parameters to be packed into a single object. This object is then passed to the `JadeDBConnection` constructor.

Alternatively, you can specify a connection and application name. The names used must be specified in the configuration file for the application. For details, see “[Configuration File](#)”, in Chapter 2.

The following C# code example shows the establishment of a connection to JADE.

```
JadeConnectionParams jcp = new JadeConnectionParams (  
    "c:/jade/system",  
    "c:/jade/system/jade.ini",  
    JadeDBConnection.ConnectionType.SingleUser,  
    "ErewhonInvestmentsModelSchema",  
    "ErewhonInvestmentsModelSchema",  
    "ErewhonUser",  
    "",  
    1);  
JadeDBConnection connection = new JadeDBConnection(jcp);
```

The following alternative uses the configuration file example under “[Configuration File](#)”, in Chapter 2.

```
JadeDBConnection connection = new JadeDBConnection("MyConnection", "App1");
```

When you have finished with the instance of `JadeDBConnection`, delete the resources used and disconnect from JADE by calling the `Dispose` method. In C#, this method is implicitly called by the `using` statement, for example:

```
using (JadeDBConnection connection = new JadeDBConnection(jcp))  
{  
}
```

If you do not use the `using` statement, call the `Dispose` method explicitly.

VB.NET does not have a `using` statement, so the creation of a `JadeDBConnection` instance must be placed inside a `try/finally` block, with a call to the `Dispose` method in the `finally` clause. In C++, the `delete` method calls the `Dispose` method.

The next step in the initialization process is to access a **JadeDBProcess** object by using the **GetProcess** method of the **JadeDBConnection** class. The **JadeDBConnection** instance provides a pool of **JadeDBProcess** objects containing one process, by default. You can increase this number by specifying the **numberProcesses** attribute of the **<application>** element in the configuration file. For more details about the process pool, see “[Pool of JadeDBProcess Instances](#)”, later in this chapter.

A **JadeDBProcess** object represents an instance from a pool of processes that is assigned to a thread. Only one **JadeDBProcess** instance can be active on a thread at any time. The **JadeDBProcess** class also implements the *Dispose* pattern, the **Dispose** method being called explicitly or implicitly from a **using** statement. In this case, the **Dispose** method releases a pooled process from its current thread and places it back into the free pool for later use.

With the **using** statement in C#, a **JadeDBProcess** instance can be acquired and released, as shown in the following example.

```
using (JadeDBProcess process = connection.GetProcess())
{
    Company coy = Company.FirstInstance();
}
```

Signing Off from JADE

To sign a **JadeDBProcess** off from JADE, call the **JadeDBProcess** method **Dispose**. To disconnect from the JADE database, call the **JadeDBConnection** method **Dispose**.

When the **Dispose** method is called for a **JadeDBConnection** instance, the **Dispose** method is called automatically on any **JadeDBProcess** instances.

Pool of JadeDBProcess Instances

If the .NET application uses multiple threads to access JADE objects, a pool of **JadeDBProcess** instances can be made available with the pool size specified by the **numberProcesses** attribute of the **<application>** element in the configuration file.

If you run more threads than there are processes in the process pool, a **GetProcess** method call on the **JadeDBConnection** instance may be unable to satisfy the request, as all processes are currently assigned to a thread.

An overloaded form of the **GetProcess** method takes a **timeout** parameter, which enables the **GetProcess** method to wait for a specified number of milliseconds for a process to be released into the pool. For example, the **GetDBProcess** method in the following code waits for up to a second for a process to become available.

```
JadeDBProcess process = connection.GetDBProcess(1000);
using (process)
{
    Company coy = Company.FirstInstance();
}
```

Accessing JADE Objects

Exposed JADE classes have the following methods defined for them.

Method	Returns ...
FirstInstance	The first instance of a class
LastInstance	The last instance of a class
AllInstances	All the instances of a class

For example, the following typical sequence of C# calls returns the first instance of the **Company** class, iterates through a collection on the class, and uses references and methods of the objects in the collection.

```
Company company = Company.FirstInstance();
foreach (Client client in company.AllClients)
{
    Location clientLocation = client.myLocation;
    if (Company.checkLocation(clientLocation))
    {
        client.processOrders();
    }
}
```

You can use the **jomDotNet** defined **Type ObjectId** to locate a specific JADE instance. Using the JADE object identifier (oid) for the object, you can initialize an **ObjectId** instance and use the static method **FindJadeInstance** of the **JadeObject** class, as shown in the following example.

```
MyObj myObj = MyObj.FindJadeInstance(new ObjectId(3274, 1));
```

You can obtain the oids of the standard JADE objects by using the **GetExecInstances** method of the **JadeDBProcess** instance. The standard JADE objects are listed in the following table.

Standard Object	Description
global	The persistent object for a schema, which is shared by all applications running from that schema. This is often used to exchange information between applications or to retain information when an application closes.
app	The current transient application instance. This is often used to store application-specific data.
rootSchema	The JADE RootSchema .
currentSchema	Current user-defined schema.
system	The JADE architectural environment consisting of a group of nodes to which the current node belongs.
node	The workstation that hosts the execution of the current process. One node object exists for each <i>logical</i> workstation connected to the server node workstation. There is one fixed server node and none or many client nodes. A node represents a workstation that runs a number of processes.
process	The current thread in a workstation executing the current method.
currentSession	The current Web session

A typical use of the **app** object is to store a reference to a *root* object, which is a singleton persistent object that contains collections of all the objects in a class. For example, in a banking application the root object would represent the bank itself and would have a collection of all the customers of the bank. The following code would initialize a reference to the root object.

```
ObjectId global;
ObjectId app;
ObjectId rootSchema;
ObjectId currentSchema;
ObjectId system;
ObjectId node;
ObjectId process;
ObjectId currentSession;
dbProcess.GetExecInstances (out ObjectId global,
                            out ObjectId app,
                            out ObjectId rootSchema,
                            out ObjectId currentSchema,
                            out ObjectId system,
                            out ObjectId node,
                            out ObjectId process,
                            out ObjectId currentSession);
CSharp applic = CSharp.FindJadeInstance (app) as CSharp;
Bank bank = applic.GetFirstBank () as Bank;
```

Creating Objects

To create a new instance of a JADE class, use the class constructor, specifying the persistence of the object using the **ClassPersistence** parameter, and then assign values to the properties of the class instance, as shown in the following example.

```
process.BeginTransaction (JomTransactionType.Persistent);
Client client = new Client (ClassPersistence.Persistent);
client.name = "John Smith ";
process.EndTransaction (JomTransactionType.Persistent);
```

If you use the constructor without a parameter, you must call the **CreateJadeInstance** method to create a JADE object.

Deleting Objects

To delete an instance of a JADE class, use the **DeleteJadeInstance** method.

Handling JADE Lock Exceptions

The **JadeDBProcess** class provides the **AddLockExceptionHandler** method, which registers an object to handle lock exception retries in .NET code, as follows.

```
process.AddLockExceptionHandler (handler);
```

The class of the **handler** parameter is **JadeLockExceptionHandler**, which is defined as follows.

```
public delegate JadeLockExceptionActions
JadeLockExceptionHandler (JadeExceptionInfo info);
```

This method returns a **JadeLockExceptionActions** enum value of **Abort**, **Continue**, **PassBack**, or **RetryOperation**.

When a lock exception is raised, the most-recently registered handler method is called. The **info** parameter holds details of the lock exception, and provides methods to examine this information.

The handler method can deal with the exception and return **Abort** or **Continue**, or it can return **PassBack** to let the next most-recently armed handler deal with it.

If none of the handlers in the registered stack handle the exception (that is, there are no registered handlers or they all return **PassBack**), a regular JADE exception is raised.

The **JadeDBProcess** class provides the **RemoveLockExceptionHandler** method, which unregisters an object as a lock exception handler. It also provides the **GetLockExceptionHandlers** method, which returns an array of currently armed lock exception handlers.

Notifications

The **JadeProcess** class provides methods to interface with user and system JADE notifications.

To receive JADE notifications:

- Register with JADE for events in which you are interested, by calling the **BeginNotification** method.
- Provide a notification handler method to be called when the notification occurs. Do this by adding your handler to the **SystemNotification** or **UserNotification** property, as appropriate. In C#, this is normally done as follows.

```
company.UserNotification += company_UserNotification;
```

Use the **CauseEvent** method to raise an event. Any .NET application or pure JADE application registered for this event will receive a notification.

Notifications are *not* raised on the same thread as that used to register the notification. In .NET (both Windows forms and WPF), all access to a control must be made on the thread that created the control. If a notification handler needs to access a control, the **Invoke** method for the control must be used on the correct thread.

The following example shows methods that subscribe, unsubscribe, and respond to notifications.

```
void beginSystemNotification()
{
    Company company = Company.FirstInstance();
    Client client = Client.FirstInstance();
    company.SystemNotification += company_SystemNotification;
    company.BeginNotification(client,
                             NoteEventType.ALL_JOM_EVENTS,
                             NoteResponseType.Continuous,
                             99);
}

void endSystemNotification()
{
    Company company = Company.FirstInstance();
    Client client = Client.FirstInstance();
    company.SystemNotification -= company_SystemNotification;
    company.EndNotification(client,
                             NoteEventType.ALL_JOM_EVENTS,
                             NoteResponseType.Continuous,
                             99);
}
```

```
void company_SystemNotification(object sender, JadeSystemNotificationArgs e)
{
    // Process the notification.
}
```

Exceptions

The JADE class library can raise the following three types of exception.

- **JadeJomException** in the **JadeWorld.Jade.Interop** namespace
This exception is raised when an unhandled exception occurs in JADE logic. In this situation, the method is always aborted along with any outstanding transaction.
- **JadeInteropException** in the **JadeWorld.Jade.Interop** namespace
This exception is raised when there is an error in the .NET to JADE interface layer and it normally indicates improper use of the JADE .NET class library.
- **JadeConnectionException** in the **JadeWorld.Jade.Configuration** namespace
This exception is raised when there is an error or inconsistency in the configuration and connection with JADE.

These exceptions can be caught in the normal manner, as shown in the following example.

```
void catchAnException()
{
    Client c = Client.FirstInstance();
    try
    {
        // Should generate 'update outside transaction' exception
        c.Name = c.Name;
    }
    catch (JadeJomException e)
    {
        writeLine(String.Format("Jade Exception: {0}, {1}", e.ErrorCode,
                                e.ErrorText));
    }
}
```

Creating Web Services Using JADE Objects

ASP.NET, **jomdotnet.dll**, and your exposed class library can be used to create a Web service application.

In an ASP.NET application, methods are made accessible through the Web by adding the **WebMethod** attribute to your method, as shown in the following example.

```
[WebMethod]
public string HelloWorld()
{
    return "Hello World";
}
```

JADE Sign On

A Web service requires a pool of JADE processes to service incoming requests. This pool of processes can be provided by the **MultiProcessJadeSignOn** method.

You can use **JadeInitialize** and **MultiProcessJadeSignOn** methods to create a log-on method, but to avoid having a Web user call a log-on method, insert the following line in the **httpModules** section of the **web.config** file for your ASP.NET application, as shown in the following example. This provides automatic log-on and log-off functionality.

```
<httpModules>
  <add name="JadeConnectModule"
        type="JadeWorld.Jade.Interop.WebServices.JadeConnectModule,
             jomDotNet"/>
</httpModules>
```

In the **appSettings** section, add connection values, as shown in the following example.

```
<appSettings>
  <add key="JadeProcesses" value="2"/>
  <add key="JadeConnectionFile" value="erewhon.xml"/>
  <add key="JadeExposureAssemblies" value="Erewhon"/>
</appSettings>
```

Adjust the key and value attributes listed in the following table, as required.

Key	Value
JadeProcesses	Number of processes to start
JadeConnectionFile	XML file containing connection attributes
JadeExposureAssemblies	Semicolon-separated list of exposure class libraries

Connecting to a JADE Process

You can add another line to the **web.config** file to provide automatic attaching to a floating process, as shown in the following example.

```
<add name="ProcessHijackModule"
      type="JadeWorld.Jade.Interop.WebServices.ProcessHijackModule,
           jomDotNet"/>
```

You should also insert this line in the **httpModules** section. As an alternative, JADE provides a SOAP extension to perform the same function. Use of this extension does not require the **httpModules** section. Instead, you can use the attribute on the method, as shown in the following example.

```
[JadeProcessHijackAttribute(Timeout = 2000)]
```

This has the advantage of acquiring a JADE process only when one is actually required. With the **httpModules** section, a process is acquired and released on every request. However, the SOAP extension does not work when coming from a browser.

The following example shows a Web method that uses an **httpModules** section.

```
[WebMethod]
public String GetClient (string clientName)
{
    Company coy = Company.FirstInstance();
    Client client = coy.AllClients.GetAtKey(clientName);
    if (client == null)
        throwException("Client " + clientName + " not found!");
    return client.Name + " " + client.Address1;
}
```

The following example shows a corresponding Web method that uses a SOAP extension.

```
[WebMethod]
[JadeProcessHijackAttribute(Timeout = 2000)]
public String GetClient_usingHttpModule(string clientName)
{
    Company coy = Company.FirstInstance();
    Client client = coy.AllClients.GetAtKey(clientName);
    if (client == null)
        throwException("Client " + clientName + " not found!");
    return client.Name + " " + client.Address1;
}
```

You can implement your own HTTP modules or SOAP extension methods in place of those provided by JADE.

Index

.NET

- accessing JADE database from, 7-8
 - connecting to JADE, 8
 - developing applications, 19-27
 - exposure definition, 9-10
 - JADE class library, 7-8
 - namespace, 10
 - object model, 8
 - persisting objects, 9
 - project directory for, 13
 - runtime environment for, 13
 - types, 15
 - using JADE classes from, 19-27
- .NET persistence in JADE API, 9
- <application> element, 12, 21
- <classExposureAssemblies> element, 12
- <codebase> element, 13
- <configSections> element, 11
- <configuration> element, 13
- <connection> element, 12
- <jomdotnet.logging> element, 12
- <runtime> element, 12-13

A

- accessing objects, 9
- accessor
 - property, 15-16
- app object in JADE, 22-23
- Application class, 9, 17
- application configuration file for exposure, 10-13
- applications
 - developing .NET, 19-27
 - Web, 13
 - Web service, 25
- assembly
 - jomdotnet, 19-20
 - proxy class, 7

C

- C#
- class definitions, 7
 - class files, 9
 - generating source files, 7
 - project file for, 10, 20
 - sequence of calls from, 22-23
- caching JADE objects, 9

calls

- sequence of C#, 22-23

class

- accessing exposed JADE, 22-24
 - Application, 9, 17
 - Collection, 17-18
 - exposed JADE, 14
 - exposing JADE, 7
 - ExtKeyDictionary, 9
 - Global, 9, 17
 - JadeConnectionException, 25
 - JadeConnectionParams, 20
 - JadeDBConnection, 9, 12, 20
 - JadeDBProcess, 9, 21
 - JadeInteropException, 25
 - JadeJomException, 25
 - JadeObject, 14
 - MemberKeyDictionary, 9
 - Node, 9
 - ObjectArray, 9
 - ObjectSet, 9
 - Process, 9
 - proxy, 7-8
 - System, 9
 - using JADE, 19-27
- class library
- JADE .NET, 7-8
 - Class-name.cs* file, 10, 17
 - Collection class, 17-18
 - concept of a root object, 23
 - configuration file, 11-13
 - connecting a .NET application to JADE, 8
 - creating JADE objects, 23
 - currentSchema object in JADE, 22-23
 - currentSession object in JADE, 22-23

D

- deleting JADE objects, 23
- deleting objects, 9

E

- element
- <application>, 12, 21
 - <classExposureAssemblies>, 12
 - <codebase>, 13
 - <configSections>, 11

- element (*continued*)
 - <configuration>, 13
 - <connection>, 12
 - <jomdotnet.logging>, 12
 - <runtime>, 12-13
- environment
 - .NET runtime, 13
- events
 - registering for JADE, 24
- exception handlers
 - registering, 25
 - unregistering, 25
- exceptions
 - handling, 23-25
 - registering a handler for, 25
 - unregistering a handler for, 25
- exposed class file, 10, 14-18
- exposed JADE class, 14
- exposed JADE class access, 22-24
- exposed JADE properties, 15-16
- exposing JADE classes, 10, 14-18
- exposing JADE constants, 14-16
- exposing JADE methods, 10, 14-16
- exposing JADE properties, 10, 14-16
- exposure
 - application configuration file, 10-13, 20
 - JADE classes to .NET, 7
 - project file, 10
- Exposure-name* namespace, 17
- Exposure-name.config* file, 10
- ExtKeyDictionary class, 9

F

- files
 - C# class, 9
 - C# project, 10
 - C# source, 7
 - Class-name.cs*, 10, 17
 - configuration, 11-13
 - Exposure-name.config*, 10
 - Exposure-name.csproj*, 10
 - jomdotnet.dll, 8-10, 13

G

- Global class, 9, 17
- global object in JADE, 22-23

H

- handling exceptions, 23-25
- HTTPS modules implementation, 27

I

- IEnumerable interface, 17

- instances
 - pool of JadeDBProcess, 21
- interface
 - IEnumerable, 17

J

- JADE
 - .NET class library, 7-8
 - accessing, 21
 - application name, 12
 - exposing classes, 10
 - exposing classes to .NET, 7
 - exposing methods, 10
 - exposing properties, 10
 - object caching, 9
 - object model, 8
 - RootSchema, 9
 - schema name, 12
 - signing off from a .NET application, 21
 - signing on from a .NET application, 8, 20
 - sign-on parameters, 12
 - JADE .NET class library, 7-8
 - JADE class exposure, 14
 - JADE collections, 9
 - JADE database
 - accessing from .NET, 7-8
 - JADE DLL file location, 13
 - JADE language, 16
 - JADE objects
 - app, 22-23
 - creating, 23
 - currentSchema, 22-23
 - currentSession, 22-23
 - deleting, 23
 - global, 22-23
 - methods for accessing, 22-27
 - node, 22-23
 - process, 22-23
 - rootSchema, 22-23
 - standard, 22-23
 - system, 22-23
 - JADE process
 - connecting to a, 26-27
 - JADE properties
 - exposed, 15-16
 - JADE types, 15
 - JadeConnectionException class, 25
 - JadeConnectionParams class, 20
 - JadeDBConnection class, 9, 12, 20
 - JadeDBProcess class, 9, 21
 - JadeInteropException class, 25
 - JadeJomException class, 25
 - JadeObject class, 14
 - JadeProcess class, 24
 - JadeWorld namespace, 9, 19-20
 - jomdotnet assembly, 7-9, 13, 19-20

jomdotnet.dll file, 8-10, 13

L

library

JADE .NET class, 7-8

location of JADE DLL file, 13

lock exception handlers

registering, 24

unregistering, 24

lock exceptions

handling, 23-24

registering a handler for, 24

unregistering a handler for, 24

M

MemberKeyDictionary class, 9

methods for accessing JADE objects, 22-27

modules

HTTP, 27

multiple threads

accessing JADE using, 21

mutator

property, 15-16

N

namespace

.NET, 10

Exposure-name, 17

JadeWorld, 9, 19-20

Node class, 9

node object in JADE, 22-23

notifications

causing, 24

handling, 24

interfacing to JADE, 24

registering for, 24

unregistering for, 24

O

object model

.NET, 8

JADE, 8

ObjectArray class, 9

objects

accessing, 21-27

creating, 23

deleting, 23

ObjectSet class, 9

P

parameters

method, 16

parameters (*continued*)

syntax of, 16

usage, 16

Usage.Constant, 16

Usage.Input, 16

Usage.IO, 16

Usage.Output, 16

persistence, 9

persisting .NET objects, 9

process

connecting to a JADE, 26-27

timing out, 21

Process class, 9

process object in JADE, 22-23

project

C#, 20

project directory

.NET, 13

project file for exposure, 10

project files

C#, 10

properties

exposed JADE, 15-16

proxy class access, 7-8

R

registering exception handlers, 25

registering lock exception handlers, 24

root object concept, 23

RootSchema, 9

rootSchema object in JADE, 22-23

runtime environment

.NET, 13

S

service

creating a Web, 25

signing off to JADE from a .NET application, 21

signing on to JADE, 12

signing on to JADE from a .NET application, 8, 20

syntax

method parameter, 16

System class, 9

system object in JADE, 22-23

T

threads

accessing JADE using multiples, 21

timing out a process, 21

transaction control, 9

types

.NET, 15

JADE, 15

U

- unregistering exception handlers, 24, 25
- Usage.Constant method parameter, 16
- Usage.Input method parameter, 16
- Usage.IO method parameter, 16
- Usage.Output method parameter, 16

V

- Visual Studio
 - building .NET project, 7
 - reference to JADE assembly, 7

W

- Web application, 13
- Web service creation, 25
- Web service applications using JADE objects, 25