External Interface Developer's Reference

VERSION 2018.0.01

jade

Contents

Contents v

Before You Begin

The *JADE External Interface Developer's Reference* is intended as a major source of information when you are using external interfaces to develop or maintain JADE applications.

Who Should Read this Reference

The main audience for the *JADE External Interface Developer's Reference* is expected to be developers of JADE application software products.

What's Included in this Reference

The *JADE External Interface Developer's Reference* has five chapters and two appendixes.

Chapter 1	Gives a reference to using external methods and external functions
Chapter 2	Gives a reference to external interfaces, including OCX, OLE, mainframe hosts, and ODBC
Chapter 3	Provides instructions for transforming an external relational database
Chapter 4	Provides instructions for importing and maintaining ActiveX control libraries and automation libraries
Chapter 5	Gives a reference to XML Metadata Interchange (XMI) support
Appendix A	Gives a reference to the Open Database Connectivity (ODBC) reserved words
Appendix B	Gives a reference to the JADE pointers in the RootSchema **jomos** external function library

Conventions

The *JADE External Interface 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.
Italic	Parameter values or placeholders for information that must be provided; for example, if instructed to enter *class-name*, 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.

Convention	Description
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 "Parameter Mapping to C Data Types" 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.

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 Developer's Reference	Developing or maintaining JADE applications
JADE .NET Developer's Reference	Developing applications using .NET class libraries exposed in JADE
JADE Database Administration Guide	Administering JADE databases
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 Report Writer User's Guide	Using the JADE Report Writer to develop and run reports

Title	Related to…
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

Chapter 1 Using External Methods and External Functions

This chapter covers the following topics.

Using External Methods

You can write external user-methods (or routines) for JADE classes. External routines can be written in any language that can create a Dynamic Link Library.

The .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.

For details, see "Writing External Methods", in the following section, and "JADE Application Programming Interface (API)", in Chapter 3 of the *JADE Object Manager Guide*. For details about low-level C API to obtain JADE initialization file information, directory information, and to convert JADE characters to an ANSI or Unicode string, see "C-Level Application Programming Interface (API)", in Chapter 6 of the *JADE Object Manager Guide*. For details about adding an external method to a class, see "Adding External Methods to Classes", in Chapter 8 of the *JADE Development Environment User's Guide*.

Notes Before you can create a new external method, the library containing your library file must already exist and you must select the class to which the method is to be added. You can use the same library to import both external methods and external functions, if required.

In JADE thin client mode, an external method call made from JADE logic executes on the application server workstation and not that of the presentation client.

Writing External Methods

The **demodll** folder in the JADE **examples** directory provides you with a demonstration schema, C++ external method (**demodll.cpp**), and the include files and library files used when writing external methods.

Sample .**vcproj** files are included for the current Microsoft Visual Studio versions. For details, see the **readme.txt** file in the **demodll** folder in the JADE **examples** directory.

Include Files

As a minimum, include the following header files in your library.

- jomobj.hpp
- jomtypes.h

Other include files that may be required are installed in the **include** subdirectory.

Library Files

When using Microsoft Visual C or C++ to write external methods, link to the **jom.lib** library file. If you need to call a JADE method that returns strings, you also need to link to the **jomutil.lib** library file.

Note Library files are available with JADE for the support of ANSI characters, Unicode characters, 32-bit, and 64-bit. Ensure that you link with the appropriate libraries for the type of project that you are building.

Microsoft Compiler Name Decoration

The Microsoft compiler uses the calling conventions used in the declaration of a function to identify the form of the name decoration (name mangling) that is used.

JADE assumes the Windows standard calling convention (**stdcall**) for external methods and functions.

By default, the Microsoft compiler decorates the exported name by adding a leading underscore character (_) and a trailing at sign (@), followed by a number that represents the number of bytes in the parameter list. External JADE methods must be exported using undecorated names. For more details, see the sample **vcproj** files or the Microsoft Linker documentation.

Supported Compiler Versions

You can create Windows libraries containing external JADE methods by using any compiler that supports the Windows "standard calling convention" (**stdcall**). However, if these libraries contain calls to the JADE Object Manager API, they should link to the relevant JADE Object Manager import libraries provided in the JADE **library** directory during installation or upgrade.

For details about the supported version of the Microsoft compiler, see "Software Requirements", in Chapter 1 of the *JADE Installation and Configuration Guide*.

Object Method Interface

The C++ function signature for the JADE Object Manager methods is:

```
extern "C" DllExport int JOMAPI
functionName (DskBuffer *pBuffer,
              DskParam  *pParams,
              DskParam  *pReturn)
```

The JADE Object Manager method parameters are listed in the following table.

Parameter	Description
pBuffer	A pointer to the object buffer for the receiver object.
pParams	Used to pass one or more parameters to the method. If there are no parameters, you can pass a **null** pointer.
pReturn	This pointer is used to pass a return value from the invoked method back to the sender.

The values listed in the following table are returned.

Value	Description
0	The function was successful.
<>0	The function failed and is converted by the system into a system exception.

The **jomtypes.h** and **jomdefs.h** header files, included in **jomobj.hpp**, contain details for the types and associated defines used by this interface.

Primitive Method Interface

The C++ function signature for the JADE Object Manager primitive methods is:

```
extern "C" DllExport int JOMAPI
functionName (UINT      primitiveNumber,
              DskParam  *primitiveValue,
              DskBuffer *pParams,
              DskParam  *pReturn)
```

The primitive method parameters are listed in the following table.

Parameter	Description
primitiveNumber	The unique number assigned to the type of the primitive in the schema.
primitiveValue	Contains the actual value of the primitive.
pParams	Used to pass one or more parameters to the method. If there are no parameters, you can pass a **null** pointer.
pReturn	This pointer is used to pass a return value from the invoked method back to the sender.

The values listed in the following table are returned.

Value	Description
0	The function was successful.
<>0	The function failed and is converted by the system into a system exception.

Passing Parameters Using the DskParam Structure

The **DskParam** structure consists of a header and a body. The header specifies the type of parameter being passed. Header values are defined in the **jomtypno.h** file. Valid parameter types are defined in the **jomtypes.h** file.

The body contains the parameter value and combines all of the C++ types that are currently supported. When a method contains more than one parameter, these C++ types are passed by the **pParams** parameter in the **parameterList** format of **DskParam**.

A **parameterList** is an array of pointers to **DskParam** structures, with one pointer for each parameter. If a parameter is not used or is not defined, a NULL pointer may be used; for example, when a method has no parameters or it does not return a value.

Use the following functions declared in **jomparam.hpp** to get and set parameter values in a **DskParam** structure.

Binary

```
int paramGetBinary(const ShortDskParam& param,
                   BYTE*& pValue,
                   Size& length)

int paramSetBinary(ShortDskParam& param,
                   const BYTE* pValue,
                   Size length,
                   PUsage usage = USAGE_CONSTANT)
```

Boolean

```
int paramGetBoolean(const DskParam& param,
                    Boolean& value)

int paramSetBoolean(DskParam& param,
                    Boolean value = FALSE,
                    PUsage usage = USAGE_CONSTANT)
```

Byte

```
int paramGetByte(const DskParam& param,
                 Byte& value)

int paramSetByte(DskParam& param,
                 Byte value = 0,
                 PUsage usage = USAGE_CONSTANT)
```

Character

```
int paramGetCharacter(const DskParam& param,
                      Character& value)

int paramSetCharacter(DskParam& param,
                      Character value = 0,
                      PUsage usage = USAGE_CONSTANT)
```

Date

```
int paramGetDate(const DskParam& param,
                 DskDate& value)

int paramSetDate(DskParam& param,
                 DskDate value = 0,
                 PUsage usage = USAGE_CONSTANT)
```

Decimal

```
int paramGetDecimal(const DskParam& param,
                    DskDecimal& value)

int paramSetDecimal(DskParam& param,
                    const DskDecimal& value,
                    PUsage usage = USAGE_CONSTANT)
```

Integer

```
int paramGetInteger(const DskParam& param,
                    Integer& value)

int paramSetInteger(DskParam& param,
                    Integer value = 0,
                    PUsage usage = USAGE_CONSTANT)
```

Integer64

```
int paramGetInteger64(const DskParam& param,
                      Integer64& value)

int paramSetInteger64(DskParam& param,
                      Integer64 value = 0,
                      PUsage usage = USAGE_CONSTANT)
```

MemoryAddress

```
int paramGetMemoryAddress(const DskParam& param,
                          DskMemoryAddress& value)

int paramSetMemoryAddress(DskParam& param,
                          const DskMemoryAddress& value,
                          PUsage usage = USAGE_CONSTANT)
```

Oid

```
int paramGetOid(const DskParam& param,
                DskObjectId& value)

int paramSetOid(DskParam& param,
                const DskObjectId& value,
                PUsage usage = USAGE_CONSTANT)
```

Point

```
int paramGetPoint(const DskParam& param,
                  DskPoint& value)

int paramSetPoint(DskParam& param,
                  const DskPoint& value,
                  PUsage usage = USAGE_CONSTANT)
```

Real

```
int paramGetReal(const DskParam& param,
                 Real& value)

int paramSetReal(DskParam& param,
                 Real value = 0,
                 PUsage usage = USAGE_CONSTANT)
```

Note The JADE **Real** primitive type equates to the C **double** type (an 8-byte real representation).

If you need to explicitly set or get float, double, or long double values, use the appropriate **paramGetFloat**, **paramGetDouble**, **paramGetLDouble**, **paramSetFloat**, **paramSetDouble**, or **paramSetLDouble** function.

String

```
int paramGetString(const DskParam& param,
                   Character* pValue,
                   Size length)

int paramSetString(DskParam& param,
                   Character* pValue,
                   Size length = UseNullTerminatedLength,
                   PUsage usage = USAGE_CONSTANT)
```

Note Within JADE, you can embed null characters in strings. If a string containing embedded **null** characters is passed as a usage **input** parameter to an external method, you should assume that the string returned by **paramGetString** ends at the first **null** character; that is, assume that the string has been truncated at the first **null** character.

To safely pass strings containing embedded **null** characters from JADE to external methods, define the string parameter in JADE as **io** usage.

StringUtf8

```
int JOMAPI paramGetString(const DskParam &param,
                          Character      *&pValue)

int paramSetStringUtf8(DskParam& param,
                       const StringUtf8Info& value,
                       PUsage usage = USAGE_CONSTANT)
```

Time

```
int paramGetTime(const DskParam& param,
                 DskTime& value)

int paramSetTime(DskParam& param,
                 DskTime value = 0,
                 PUsage usage = USAGE_CONSTANT)
```

TimeStamp

```
int paramGetTimeStamp(const DskParam& param,
                      DskTimeStamp& value)
```

```
int paramSetTimeStamp(DskParam& param,
                      const DskTimeStamp& value,
                      PUsage usage = USAGE_CONSTANT)
```

TimeStampInterval

```
int paramGetTimeStampInterval(const DskParam& param,
                              DskTimeStampInterval& value)

int paramSetTimeStampInterval(DskParam& param,
                              const DskTimeStampInterval& value,
                              PUsage usage = USAGE_CONSTANT)
```

TimeStampOffset

```
int paramGetTimeStampOffset(const DskParam& param,
                            DskTimeStampOffset& value)

int paramSetTimeStampOffset(DskParam& param,
                            const DskTimeStampOffset& value,
                            PUsage usage = USAGE_CONSTANT)
```

Parameter List

```
int paramGetParameter(const DskParam& param,
                      EntryCount index,
                      DskParam*& pValue)
```

Initializing the DskParam Structure

When your external method is called from a JADE method, the **DskParam** structure headers are initialized to contain the correct parameter and return types specified in your schema-defined method header.

If you are passing a **DskParam** structure to a method or function, it is your responsibility to initialize the **DskParam** header to contain the appropriate type for the associated parameter or return type. Use the **paramSet** functions in **jomparam.hpp** to do this.

String and Binary Parameters and Return Types

Use the following functions declared in **jomparam.hpp** to initialize a **DskParam** structure to contain a JADE language string or binary.

These functions allocate a buffer of the specified length from the JADE string pool, copy the string or binary value into this buffer, and then initialize the **DskParam** structure to contain a reference to this buffer.

```
int JOMAPI paramCreateJadeString(DskParam  *pJadeString,
                                 Character *pData,
                                 Size      length);

int JOMAPI paramCreateJadeBinary(DskParam *pJadeString,
                                 BYTE     *pData,
                                 Size     length);
```

The following functions that release the memory associated with the copy of the string or binary are also available.

```
int JOMAPI paramDeleteJadeString(DskParam  *pJadeString);

int JOMAPI paramDeleteJadeBinary(DskParam  *pJadeString);
```

Using the C++ Proxy Classes

The **DskObject** class is defined in **jomobj.hpp**. Use this class to make object-related API calls easier to use. In particular, the **DskObject** class methods listed in the following table are available.

JADE Object Manager API Call	DskObject Method
jomCreateObject	createObject
jomDeleteObject	deleteObject
jomGetProperty	getProperty
jomSetProperty	setProperty
jomLockObject	lockObject
jomUnlockObject	unlockObject
jomUpdateEdition	updateEdition
jomSendMsg	sendMsg
jomSendDbMsg	sendDbMsg
jomGetObject	getObject
jomClearBuffers	clearBuffer

For more details, see "JADE Application Programming Interface (API)", in Chapter 3 of the *JADE Object Manager Guide*.

Example of a C++ External Method

The following is an example of a C++ external method library.

```
#include "jomobj.hpp"

extern "C" DllExport int JOMAPI
PassOneParam(DskBuffer *pbuffer,
             DskParam  *pParams,
             DskParam  *pReturn)
{
    Character *myParam;
    int       result;

    // single parameter example
    result = paramGetString(*pParams, myParam);
    CHECK_RESULT;

    // return a 1 to JADE
    return paramSetInteger(*pReturn, 1);
}
```

```
extern "C" DllExport int JOMAPI
PassTwoParams(DskBuffer *pbuffer,
              DskParam  *pParams,
              DskParam  *pReturn)
{
    Character *myParam1;
    int       myParam2;
    int       result;
    DskParam  *pParam;

    // get multiple parameters
    result = paramGetParameter(*pParams, 1, pParam);
    CHECK_RESULT;
    result = paramGetString(*pParam, myParam1);
    CHECK_RESULT;

    result = paramGetParameter(*pParams, 2, pParam);
    CHECK_RESULT;
    result = paramGetInteger(*pParam, myParam2);
    CHECK_RESULT;
    // return a 2 to JADE
    return paramSetInteger(*pReturn, 2);
}
```

Using External Functions

An external function is a routine that is not necessarily associated with any specific class. For details about adding an external function to a class, see "Defining External Functions", in Chapter 8 of the *JADE Development Environment User's Guide*.

For details about low-level C API to obtain JADE initialization file information, directory information, and to convert JADE characters to an ANSI or Unicode string, see "C-Level Application Programming Interface (API)", in Chapter 6 of the *JADE Object Manager Guide*.

Notes JADE supports external functions only; utility functions written in the JADE language are not supported.

Only one External Functions Browser for the current schema can be open at any time in the JADE development environment. However, you can have concurrent open External Functions Browsers for different schemas in a development session.

Define external function calls for compatibility with the running JADE environment on which the calls are made. For example, calling Windows Application Programming Interfaces (APIs) that pass a Windows handle have different parameter definitions that depend on whether the client is running in 32-bit or 64-bit mode.

To cater for this, you need to establish two different external function call definitions with the required parameter types. Your calling JADE logic must then determine which architecture is being used by the client and which external function call it should therefore make.

External functions are exported from a library and can be called directly from a method written in the JADE language. This enables you to directly call entry points exported from existing third-party libraries or the operating system without having to write an external method wrapper. An external function defines a mapping between JADE parameters and the external parameters. Parameter mapping is made interpretively at run time when an external function is called.

Defining an external function is very much like defining an external method and a subset of the same information is captured. However, external functions are viewed and maintained from the External Functions Browser, accessed from the Browse menu at schema level, unlike external methods that are viewed and maintained from the Class Browser or the Primitive Types Browser.

External function calls are invoked with the JADE language **call** expression, to make a clear distinction between *function* calls (that are not invoked on an object) and object *method invocations*.

For details about external functions declared in the RootSchema **jomos** external function library that call Windows library functions as defined in the Microsoft Developer Network (MSDN), see Appendix B.

Parameter Mapping to C Data Types

The primitive types that can be used as parameters to external functions are a restricted subset of the JADE primitive types. External functions do not support mapping to **Decimal**, **Date**, **Time**, **TimeStamp**, **TimeStampInterval**, or **TimeStampOffset** primitive types.

The external function mapping of standard C types to simple JADE primitive types is listed in the following table.

C Data Type	Recommended JADE Primitive Type
int	Integer
int64	Integer64
void*	MemoryAddress
long	Integer
short	Integer
char (ANSI)	Character
wchar_t (Unicode)	Character
bool	Boolean
float	Real[4]
double	Real or Real[8]

The external function mapping of standard C types to structured JADE primitive types is listed in the following table.

C Data Type	Recommended JADE Primitive Type
char* containing null-terminated string	String
char* containing binary data	Binary

You can substitute other JADE primitive types to clarify the interface. For example, you can map a JADE **Boolean** primitive type to any of **int**, **long**, **short**, **char**, or **unsigned char** C data types, and so on.

The handling and interpretation of signed or unsigned values is left to the external function protocol and your use of the function.

Parameter Mapping to a Windows Data Type

The JADE **Point** primitive type can be used as a parameter to or the return value from an external function, as listed in the following table.

Windows Data Type	Stack Frame Size Bytes	JADE Primitive Type
LPPOINT	4	Point

Both the parameter and the return value must be passed by reference, regardless of the parameter usage (for example, **constant**, **input**, or **io**). The external function must therefore define the parameter or return type as a reference or pointer to a point structure. In Windows API calls, this corresponds to the **LPPOINT** typedef; that is, a pointer to **POINT**.

C++ Structure or Class Support

JADE does not provide a facility to define structured record types with explicitly named members, although many external functions take such structures as parameters.

To provide a mapping to structured parameter types, you can define a fixed-length **Binary** primitive type in the function signature, where the binary length matches the size of the structure.

It is your responsibility to manually access fields within the **Binary** structure. For example, the **GetTimeZoneInformation** WIN32 Application Programming Interface (API) returns information in a **TIME_ZONE_INFORMATION** structure and takes a pointer to this structure as a parameter. If the size and layout of the structure are known, you can define a binary of the correct fixed-length in the signature.

You can then use the bracket (**[]**) substring operator on the binary to obtain the contents of individual structure fields and use type casting to convert them to an appropriate primitive type.

You can use the **bufferAddress** method of the **String** or **Binary** primitive type to determine the actual memory address if the structure needs to contain a pointer to another structure. For details, see "String Type" or "Binary Type", in Chapter 1 of the *JADE Encyclopaedia of Primitive Types*.

Using Real Primitive Types in External Functions

As external functions can use any of the standard external floating point types, you must specify the size of the real in the parameter definition. A four-byte **Real** primitive type maps to a C float and an eight-byte **Real** primitive type maps to a C double.

Note As the JADE **Real** primitive type maps internally to a **double** (8-byte real), conversion to **real[4]** may result in loss of precision.

Parameter Passing Conventions

The Windows *Standard* calling convention only is supported. In the Standard calling convention, parameters are pushed from right to left.

The called function is responsible for cleaning the stack and the exact number of arguments expected by the called function must be passed.

Passing by Value or Reference

The "pass by value" and "pass by reference" modes of parameter passing are supported by most languages.

C++ and Pascal directly support "by reference" parameters and C directly supports only "by value" parameters. In C, "pass by reference" is emulated by passing the address of a variable to a parameter defined as a pointer to type.

In JADE external function support, the C++ "by reference" parameter definition of the *type*& form and the C pointer parameter definition of the *type** form are considered equivalent. As they are the same from the viewpoint of the caller, the address of a variable is passed to the called function in both forms.

Parameter Passing Rules

The JADE external function facility passes parameters using the following conventions.

- Structured parameter (**String** and **Binary**) types
  - Always passed by reference; that is, the address of the **String** or **Binary** variable is taken and passed
  - When usage is **output** or **io** (which signifies the calling function may update the value), length *must* be specified in the parameter definition
  - A function that returns a **String** primitive type is assumed to return a zero-terminated string
  - A function that returns a **Binary** primitive type must have the length of the **Binary** specified in the return type
- Simple parameter (**Integer**, **Character**, **Real**, and **Boolean**) types
  - When the usage is defined as **constant** or **input**, the parameter is passed by value
  - When the usage is defined as **output** or **io**, the parameter is passed by reference
- Structured parameter (**Point**) type
  - Always passed by reference; that is, the address of the **Point** variable is taken and passed

Passing Null Pointer Values

It may be necessary under certain circumstances to pass a **null** pointer as a parameter value to an external function. You cannot pass a **null** pointer parameter value by specifying a value of **null** or **""** in the external function call, as these both produce a pointer to an empty string instead of a **null** pointer.

To pass a **null** pointer parameter value, specify a value of zero (**0**) in the external function call. For **String** and **Binary** parameters, a value of zero (**0**) yields a **null** pointer for the parameter value of the external function.

Unicode Awareness

When defining and using external functions, you must ensure that the correct versions of Unicode or ANSI functions are used in your function definitions, depending on the required internal **String** representation. For example, most WIN32 API functions that take string parameters have two versions: a Unicode (or wide character) version that is appended with a **W** character and an ANSI version that is appended with an **A** character.

For example, the two versions of the **IsCharAlphaNumeric** WIN32 API function are listed in the following table.

Version	WIN32 API Function
Unicode	IsCharAlphaNumericW
ANSI	IsCharAlphaNumericA

Calling External Functions from JADE Thin Clients

Use the **presentationClientExecution** or **applicationServerExecution** option in the function definition of an external function (that is, *not* in the **call** expression) to specify where the function call is made, as shown in the following example.

```
getProfileString(sectionName : String;
                 keyName     : String;
                 default     : String;
                 value       : String[100] output;
                 length      : Integer): Integer is GetProfileStringW in
                                        kernel32 applicationServerExecution;
```

External functions are called on the JADE thin client workstation by default; that is, **presentationClientExecution** is assumed.

Note The **presentationClientExecution** qualifier has no effect if JADE is not currently running in JADE thin client mode.

Handling Faults

The **Disabled** parameter in the [FaultHandling] section of the JADE initialization file specifies how a fault is handled. If the parameter is set to **true**, the fault is passed to the system default handler (for example, to Windows error reporting). If the parameter is set to **false**, the process is terminated after taking a process dump, where possible.

A JADE exception can be raised instead of terminating the process. This option is available for a function in a third-party DLL where any fault that occurred would never imperil the integrity of the executing process (for example, by corrupting memory or data).

For more details, see the **Disabled** parameter in Chapter 1 of the *JADE Initialization File Reference*.

Chapter 2 Using External Interfaces

This chapter covers the following topics.

- Connecting to Network Devices Using a Named Pipe
  - External Software Requirements
  - Using the NamedPipe Connection Class
  - InternetPipe Subclass
- Interfacing to the Internet

For considerations when using the Relational Population Service (RPS) to replicate a production JADE database to one or more Relational Database Management System (RDBMS) target databases, see Chapter 2, "Relational Population Service (RPS) Support", in the *JADE Synchronized Database Service (SDS) Administration Guide*.

External Database Coexistence

JADE external database coexistence provides:

- Mapping technology to enrich relational schemas with object model semantics
- Tables (base, derived, or predefined views) mapped to subclasses of **ExternalObject**
- Columns mapped to attributes
- Foreign key definitions are used to determine inverse relationships
- Tuples (or rows) map to instances of external classes
- Each external database is represented by a subclass of **ExternalDatabase** and a singleton instance
- Object identifier to primary key mapping enables JADE to uniquely identify a tuple and map this to a proxy object in JADE
- Inverse relationships can be defined and implemented using references to class types
- Multiple-valued properties can be implemented with **Dictionary**, **Set**, or **Array** types

JADE can coexist with existing legacy or heritage relational databases in a seamless fashion. JADE provides the External Schema Wizard, to enable you to map your relational database to a JADE object model. (For details about using the External Schema Wizard, see Chapter 3, "Transforming an External Relational Database".)

You can map one or more external relational databases into a JADE object model, by using a Global Conceptual Schema (GCS). The GCS provides a high-level integrated view of external relational databases as part of an existing JADE schema, hiding structural differences between the JADE object model and the relational model, giving the appearance of accessing a single unified database. The relationships between tables in the relational database are mapped to bi-directional relationships using inverse references.

Access relational objects in an external database in a procedural manner, by using the JADE language rather than constructing Structured Query Language (SQL) statements. The External Schema Wizard enables you to map:

- Tuples (which may be the result of a project/join or a view) to objects
- SQL types to JADE primitive types
- Joins to bi-directional relationships based on foreign and primary keys

The mapping is specified declaratively rather than programmatically, to reduce coding requirements and to increase flexibility.

The schema mapping technology is supported by Open Database Connectivity (ODBC), which provides the gateway to access relational databases. ODBC is used during the schema transformation phase to access catalog information, and at run time to populate virtual object instances using SQL queries. As ODBC uses SQL as its database access language, a single application can access different database management systems with the same source code. The JADE query engine calls functions in the ODBC interface, which are implemented in *drivers*; that is, database-specific modules.

ODBC Requirements for External Database Coexistence

To support external database coexistence, the ODBC Data Source Name (DSN) must be defined on any workstation that is accessing an external relational database. If you are running JADE in thin client mode, the DSN must be defined on the application server.

External Database Coexistence ODBC Requirements under Windows

The Microsoft Data Access Components (MDAC) installed with your Windows operating system is sufficient for external database coexistence.

When running JADE on a 64-bit machine under Windows in 32-bit mode, you must define the DSN by using the 32-bit version of the Microsoft Data Source Administrator (that is, do not use the default 64-bit version).

To run the 32-bit version, execute:

```
<\windows directory>\SysWOW64\odbcad32.exe
```

When running JADE on a 64-bit machine, the bit version of the JADE node where the JADE external database code is executed must match the bit version of the external ODBC driver that is being used and the bit version in which the DSN is defined. For example, if accessing Microsoft Access, which supplies only a 32-bit driver, the JADE node must be executing in 32-bit and the DSN must be defined in the 32-bit ODBC Data Source Administrator.

Semantic Schema Enrichment

The semantic schema enrichment process used to take an existing relational database schema and map this into a JADE object model makes use of existing JADE object concepts and language constructs, to access the external database in a unified manner.

The schema enrichment process of the External Schema Wizard is not a simple deterministic mapping from a pure relational schema to a JADE object model, as a relational database schema does not generally provide the additional higher level semantic information required for deriving useful relationships between classes.

The transformation of a relational schema into a JADE object model by using the External Schema Wizard is therefore a partially automated process that requires you to specify additional information and for you to have some knowledge of the semantics of the relational database that is being mapped.

In many cases, JADE can derive certain useful information directly from the relational catalog; for example, primary and foreign key specifications. Often, foreign key relationships can also be derived from naming conventions; for example, if **dept-id** is the primary key of the **department** table, the External Schema Wizard deduces **dept-id** in the **employee** table is probably a foreign key.

Usage Example

The following example of a partial reference schema definition shows the capability of the mapping to a relational database provided by JADE.

```
CREATE TABLE customer
(
    id           integer  NOT NULL,
    fname        char(15) NOT NULL,
    lname        char(20) NOT NULL,
    address      char(35) NOT NULL,
    city         char(20) NOT NULL,
    state        char(2)  NOT NULL,
    phone        char(12) NOT NULL,
    company_name char(35) NULL,

    PRIMARY KEY ("id")
)
CREATE TABLE employee
(
    emp_id     integer,
    manager_id integer,
    dept_id    integer,
    emp_fname  char(20),
    emp_lname  char(20),
    salary     numeric(20,3) NOT NULL,
    start_date date NOT NULL,
    birth_date date NULL,
    sex        char(1) NULL,

    PRIMARY KEY ("emp_id")
)

CREATE TABLE department
(
    dept_id      integer,
    dept_name    char(40),
    dept_head_id integer,

    PRIMARY KEY ("dept_id")
)
```

The schema enrichment process provided by the External Schema Wizard enables this schema to be mapped to the following JADE schema definition.

```
typeHeaders

    E_Central subclassOf ExternalDatabase;

    // External proxy classes
    E_Customer subclassOf ExternalObject  transient;
    E_Department subclassOf ExternalObject  transient;
    E_Employee subclassOf ExternalObject  transient;

    // External proxy collections
    E_CustomerDict subclassOf ExternalDictionary transient;
```

```
    E_DepartmentDict subclassOf ExternalDictionary transient;
    E_EmployeeSet subclassOf ExternalSet transient;

typeDefinitions

    // Represents external database and is also our 'root object' class
    E_Central
    (
    referenceDefinitions
        customers:   E_CustomerDict implicitMemberInverse;
        departments: E_DepartmentDict implicitMemberInverse;
    )

    E_Customer
    (
    attributeDefinitions
        address:     String[36];
        city:        String[21];
        companyName: String[36];
        firstName:   String[16];
        lastName:    String[21];
        phone:       String[13];
    )

    E_Department
    (
    attributeDefinitions
        name:       String[50];
    referenceDefinitions
        // SELECT * FROM employee
        // WHERE employee.emp_id = department.dept_head_id
        manager:    E_Employee explicitInverse;
        // SELECT * FROM employee
        // WHERE employee.dept_id = department.dept_id
        employees:  E_EmployeeSet explicitInverse;
    )

    E_Employee
    (
    attributeDefinitions
        birthDate:  Date;
        firstName:  String[20];
        lastName:   String[20];
        salary:     Decimal[20,3];
        sex:        Character;
        startDate:  Date;
    referenceDefinitions
        department: E_Department explicitInverse;
        manager:    E_Employee explicitInverse;

    // the following methods were added after the mapping phase using
    // the standard JADE class browser
    jadeMethodDefinitions
        age():             Real;     // compute using birthDate
        lengthOfService(): Real;
    )
```

```
    inverseDefinitions
employees of Department automatic parentOf department of E_Employee manual;
```

Note Tables are not simply mapped to classes, and columns to attributes. The classes contain properties that are references to other external SQL proxy classes; that is, **E_Department** and **E_Employee**. In addition, the **E_Department** class defines a multiple-valued property, **employees**, whose type is a collection.

Object Identifier Mapping

The mapping of an object identifier to primary key (or special columns) mappings directly maps a proxy instance to a row in the relational database. The **ObjectIdentifier** property of the proxy denotes the primary key of the relational entity that the proxy is modeling.

External Proxy Classes

The External Schema Wizard process creates the external proxy class, which acts as the mediator between the JADE world and the relational world. Proxy classes are derived from a common abstract subclass of **Object** in the hierarchy: **ExternalObject**. The proxy classes that comprise the transformation schema have two components: an interface and a mapping (or query) specification.

The interface defines the properties and any methods. The mapping, using an SQL-like query, defines how to populate a proxy object with instances from the external relational database. The *tuples*, or instances retrieved by this query, are virtual instances of the proxy object, enabling you to deal with transient instances of the SQL proxy classes defined in the schema.

An SQL proxy class definition can be mapped from one of the following relational entities.

- A base table
- A relational database view
- A query (for example, a join)

The properties of a proxy class can be attributes or virtual references. Attributes are simple types whose domain is one of the supported JADE primitive types. The type of an external reference can be another external class type or an external collection. External references are always defined as explicit inverse references. These references are implemented using mapping methods, and employ an external object or an external collection method to retrieve an instance.

The JADE query engine provides methods on the external schema entities used by the External Schema Wizard to import catalog information from an external relational schema. These methods employ the relevant ODBC catalog functions. As some drivers or data sources do not support all required functions, the External Schema Wizard has less information to use for production of a default mapping.

The JADE query engine interrogates driver capability, to determine whether a required function is available before using it; for example, catalog functions, block mode, and scrollable cursors.

The following entities are imported from external relational catalogs and are stored as persistent meta information in your JADE schema.

- Tables
- Columns
- Primary and foreign keys
- Indexes

The catalog import functions create new entities in the JADE schema or update existing definitions if they already exist, to enable the External Schema Wizard to provide schema upgrade capabilities.

JADE provides the classes listed in the following table, to encapsulate the behavior of your external database schema.

Class	Description
ExternalArray	Represents the rows in a result set generated from an SQL query containing a sort specification
ExternalCollection	Provides the common protocol for external collection classes
ExternalDatabase	Represents a connection to an external database
ExternalDictionary	Represents the rows in a result set generated from an SQL query with an **ORDER BY** sort specification
ExternalIterator	Encapsulates behavior required to sequentially access elements of a collection
ExternalObject	Base class for all external database classes
ExternalSet	Represents the rows in a result set generated from an SQL query that has no sort specification
ExternalSet	Represents the rows in a result set generated from an SQL query that has no sort specification
ODBCException	Defines behavior for exceptions that occur as a result of ODBC communicating with external databases

For details, see Chapter 1 of the *JADE Encyclopaedia of Classes*.

Mapping an SQL Data Type to a JADE Primitive Type

The following table lists the ANSI SQL data types and the JADE primitive types to which they are mapped by default.

ANSI SQL Data Type	JADE Primitive Type
BINARY, VARBINARY, LONGVARBINARY	Binary
BIT	Boolean
CHAR(1), TINYINT	Character
CHAR, VARCHAR, LONGVARCHAR	String
DECIMAL and NUMERIC (precision and scale factor)	Decimal (precision and scale factor)
REAL, FLOAT, DOUBLE	Real
INTEGER, SMALLINT	Integer
BIGINT (64-bit integer)	Decimal[20, 0]
DATE	Date
TIME	Time
TIMESTAMP	TimeStamp

Some external databases may have restrictions in the use of specific SQL data types, which the user may need to be aware of when changing data in those databases.

SQL Server Restrictions

If inserting character strings into a SQL Server database, trailing spaces may be added, depending on the type of the **SqlServer** field and the **Use ANSI Null, Paddings and Warnings** setting. For details, see the SQL Server documentation.

MS Access 2007 Restrictions

Multi-valued fields are mapped to **String** by the Microsoft (MS) Access ODBC driver. An update of a *Multi-valued* field in JADE code results in an error from MS Access.

Multi-valued fields that are marked as indexes have collections created by default in the JADE import, even though use of these collections results in an error from MS Access.

In both cases, the ODBC interface does not distinguish *Multi-valued* fields from *Memo* fields.

Using the External Schema Wizard

The External Schema Wizard automates the process of transforming a relational database model into a JADE object model.

Use the External Schema Wizard to access the catalog information from selected external data sources and generate classes. For details, see Chapter 3, "Transforming an External Relational Database".

Note Some data sources (ODBC drivers) may not support all catalog functions and some schemas may not contain all of the information required to derive or suggest possible relationships.

The External Schema Wizard enables you to:

- Select data sources configured on the development workstation
- Import information from a relational catalog, including tables, columns, primary and foreign keys, and indexes
- Select tables for transformation
- Map base tables or views, as well as join derived tables to classes
- Select columns, and column to attribute mapping
- Override default names suggested for all entities
- Automatically map SQL types to JADE primitive types
- Override the default JADE primitive types and length mappings
- Define external collection types, based on suggested potential dictionaries using primary key and index information obtained from the catalog
- Define relationships between derived classes, based on suggested potential relationships using foreign key information

Accessing an External Relational Database from JADE

You can access the relational database using the JADE language and by navigating relationships using inverse references; for example, you can code the following style of iterative access.

```
foreach emp in dept.employees do
    emp.display;
endforeach;
```

This code uses the external iterator and an external collection. The external collection issues the required SQL query to create a virtual collection of instances; that is, the rows of the **employee** table, where **employee.dept_id = department.dept_id**. The **dept_id** foreign key is obtained from the virtual **department** instance, which is the parent of the virtual **employee** dictionary.

The **ExternalIterator** class implements the standard iterator protocol; that is, the **next** and **back** methods, and so on.

The external iterator and external collections enable constructs like the **foreach** instruction to function in a transparent fashion.

Accessing Objects in Your External Database

The **ExternalIterator** class, the **ExternalCollection** class, and its subclasses provide methods with behavior similar to that of the methods provided by the corresponding JADE classes. For details, see Chapter 1 of the *JADE Encyclopaedia of Classes*.

Tip You do not have to explicitly open your external database, but if you do so, you must also close it explicitly.

You can access objects (or records) in your external database by using the **getAtKey** method of the **ExternalDictionary** class, as shown in the following example.

```
getEmployee;
vars
    ed  : EmployeesByLastNameDict;  // an external dictionary
    emp : Employee;                 // an external object
begin
    create ed;
    emp := ed.getAtKey("D");
    write emp;
epilog
    delete ed;
end;
```

In this example, **EmployeesByLastNameDict** is an index in the relational database that has been described to JADE by the External Schema Wizard. You can also use the **ExternalDictionary** and **ExternalIterator** class dictionary and iterator methods and the **foreach** instruction to access entries in an external collection sequentially.

The following example shows the use of the **foreach** instruction.

```
vars
    ed  : EmployeesByLastNameDict;  // an external dictionary
    emp : Employee;                 // an external object
begin
    create ed;
    foreach emp in ed do
        listBoxEmployees.addItem(emp.employeeName);
```

```
    endforeach;
end;
```

The following example shows the use of an iterator.

```
vars
    ed   : EmployeesByLastNameDict; // an external dictionary
    emp  : Employee;                // an external object
    iter : ExternalIterator;
begin
    create ed;
    iter := ed.createIterator;
    while iter.next(emp) do
        listBoxEmployees.addItem(emp.employeeName);
    endwhile;
epilog
    delete ed;
    delete iter;
end;
```

To add objects (or rows) to your external database, use the **ExternalCollection** class **createObject** method and the **ExternalObject** class **update** method, as shown in the following example.

```
createEmployee;
vars
    ed  : EmployeesByLastNameDict; // an external dictionary
    emp : Employee;                // an external object
begin
    create ed;
    emp := ed.createObject;        // creates a reference to the object
    emp.employeeName := "Sid";
    emp.update;                    // updates collection with the new object
epilog
    delete ed;
end;
```

To update an object, use the **ExternalObject** class **update** method, as shown in the following example.

```
updateEmployee;
vars
    ed  : EmployeesByLastNameDict; // an external dictionary
    emp : Employee;                // an external object
begin
    create ed;
    emp := ed.last;
    if emp.isUpdatable then
        emp.region := "ZZ";
        emp.update;
    else
        write "not updateable";
    endif;
epilog
    delete ed;
end;
```

The **beginExternalTransaction** and **commitExternalTransaction** methods provided by the **ExternalDatabase** class enable you to start an external database transaction if your external database supports transactions.

You can use this method at the start of a series of updating operations (that is, creates, deletes, or updates) that must be applied atomically to the target database to ensure consistency and the ability to recover. (Updates are committed immediately, by default, if they are not within a **beginExternalTransaction** and **commitExternalTransaction** pair of instructions which delays the commitment of updates until the **commitExternalTransaction** method is called.)

Summary of the Properties and Methods Provided by External Database Classes

This subsection contains a summary of the properties and methods provided by external database classes. For details, see the appropriate class in Chapter 1 of the *JADE Encyclopaedia of Classes*.

The **ExternalArray** class encapsulates the behavior of an ordered virtual collection that represents the rows in a result set generated from an SQL query containing a sort specification; that is, the **ORDER BY** clause. Instances of this class occur in the order determined by the **ORDER BY** clause. It does not explicitly provide any properties or methods. It inherits those provided by the **ExternalCollection** superclass.

The **ExternalCollection** class provides the common protocol for external collection classes. External collection classes represent the result set of a selection from an external data source. External collections provide operations for direct and relative key access, and may be used in collaboration with external iterators to access rows in a result set. They are read-only; that is, operations such as add, remove, clear, and purge are not supported.

The properties provided by the **ExternalCollection** class are summarized in the following table.

Property	Description
database	Contains a reference to the external database instance
filterExpression	Contains the filter that enables you to select specific rows
sortExpression	Contains the expression that controls how instances in the collection are ordered

The methods provided by the **ExternalCollection** class are summarized in the following table.

Method	Description
at	Returns the entry at a specified index in the collection
canCreate	Returns **true** if member type instances can be created
createIterator	Creates the external iterator for an external collection
createObject	Creates a new instance of the external object
first	Returns the first entry in the collection
getSQL	Returns the SQL statement of the receiver
includes	Returns **true** if the collection contains the specified object
last	Returns the last entry in the collection
maxSize	Returns the maximum number of entries that the external collection can contain
size	Returns the current number of entries in the collection

The **ExternalDatabase** class encapsulates the behavior required to access entries in an external database. This class represents a connection to an external database, and provides properties and methods that operate on the data source.

The properties provided by the **ExternalDatabase** class are summarized in the following table.

Property	Description
connectionString	Contains parameters required to connect to a data source
name	Contains the name of the external database
password	Contains the password required by the data source
serverName	Contains the name of the server defined for the data source
userName	Contains a user id used to establish a connection

The methods provided by the **ExternalDatabase** are summarized in the following table.

Method	Description
abortExternalTransaction	Rolls back the changes made during the current transaction
beginExternalTransaction	Starts a database transaction
canTransact	Returns **true** if the external database supports transactions
close	Closes the connection to an external database
commitExternalTransaction	Commits a transaction
executeSQL	Directly executes an SQL statement
getFileDSN	Returns the file data source name
getLastError	Returns the last ODBC exception when the **isSQLValid** method returns **false**
getMachineDSN	Returns the machine data source name
importStoredProcedures	For internal use only
isOpen	Returns **true** if the external database is currently open
isSQLValid	Checks the syntax of an SQL statement
isUpdatable	Returns **true** if the external database can be updated
loadProcedure	Reserved for future use
open	Opens a connection to an external database
setFileDSN	Programmatically sets the file data source name
setMachineDSN	Programmatically sets the machine data source name

The **ExternalDictionary** class encapsulates the behavior of an ordered virtual collection containing keys that represents the rows in a result set generated from an SQL query containing a sort specification; that is, the **ORDER BY** clause. External dictionaries provide direct key access to an external object instance; that is, random access to a row or tuple in the relational database.

The methods provided by the **ExternalDictionary** class are summarized in the following table.

Method	Description
getAtKey	Returns the object at the specified key

Method	Description
getAtKeyGeq	Returns the object with a key greater than or equal to the specified key
getAtKeyGtr	Returns the object with a key greater than the specified key
getAtKeyLeq	Returns the object with a key less than or equal to the specified key
getAtKeyLss	Returns the object with a key less than the specified key
includesKey	Returns **true** if the receiver contains an entry at the specified key
startKeyGeq	Sets a start position within a collection for an external iterator object
startKeyGtr	Sets a start position within a collection for an external iterator object at the next object after the specified key
startKeyLeq	Sets a start position within a collection for an external iterator object at the object equal to or before the specified key
startKeyLss	Sets a start position within a collection for an external iterator object at the object before the specified key

The **ExternalIterator** class encapsulates the behavior required to sequentially access elements of an external collection. An external iterator instance sequentially accesses the virtual instances of the collection, in a forward or a reverse direction. The methods provided by the **ExternalIterator** class are summarized in the following table.

Method	Description
back	Accesses entries in reverse order in the collection to which the external iteration is attached
getCollection	Returns the external collection associated with the receiver
isValid	Returns **true** if the receiver is a valid external iterator
next	Accesses successive entries in the collection to which the external iterator is attached
reset	Initializes the external iterator
startAtIndex	Sets the starting position of the external iterator to a specified row in the result set

The **ExternalObject** class provides a superclass for all external class subclasses, and defines the behavior specific to external proxy classes. The query engine uses the **ExternalObject** class at run time to populate virtual proxy instances. Each external class contains the SQL query required to populate a class extent or to do a **join** query for a single valued reference.

The methods provided by the **ExternalObject** class are summarized in the following table.

Method	Description
deleteSelf	Deletes the external object
isUpdatable	Returns **true** if instances can be updated (that is, create, delete, or update instances)
update	Completes a create or update operation and saves changes to the external database

The **ExternalSet** class encapsulates the behavior of an unordered virtual collection that represents the rows in a result set generated from an SQL query that has no sort specification; that is, it has no **ORDER BY** clause. The order in which instances are retrieved is dependent on your data-source. The method provided by the **ExternalSet** class is summarized in the following table.

Method	Description
includes	Returns **true** if the virtual external collection or result set contains the specified object

Object Lifetimes

The lifetime of external objects is conceptually persistent, as they are retrieved from a persistent database. However, the external proxy objects are implemented as shared transient objects in JADE.

You can create and delete rows or records in an external database, by using methods provided by the relevant classes. The creation and deletion of proxy objects is automatic. A proxy can have one current object only for each collection reference, which corresponds to the current-row in a cursor operation.

Caution When using proxy objects, do not retain references to an external object beyond the last access for a specific collection; that is, the last direct or relative key access or the last object fetched using a **foreach** instruction or the **next** or **back** method of the **ExternalIterator** class.

Using External Collections

External collections implement non-updating collection methods; for example, the **first**, **last**, and **size** methods.

External dictionaries provide direct key access to instances of an external object; that is, random access to a row or tuple in the external relational database. For example:

```
department := E_Company.departments[name];
```

Relative key access is provided by the dictionary key methods; for example, **getAtKeyGtr** or **startKeyLss**.

SQL server locks are released when the cursor used to perform the query is closed. JADE manages cursors automatically, and closes them under the following conditions.

- Cursors used to support an iteration are closed when the external iterator is deleted (or at the end of a **foreach** instruction).
- Cursors used for collection-level access methods such as **first**, **last**, **getAtKey**, and so on, are closed when the collection is used for a different query or the collection is deleted.
- All cursors are closed when the database connection is explicitly closed.

ADHOC Query Extensions

ADHOC queries are supported, enabling you to explicitly set filtering (by using the **WHERE** clause) and sort specifications (by using the **ORDER BY** clause) on external collections at run time, overriding the canned query associated with the collection.

The **filterExpression** and **sortExpression** properties provided by the **ExternalCollection** class enable you to explicitly set external collections at run time. You can set these properties before a collection is used in a **foreach** instruction or in conjunction with an iterator. The **filterExpression** property allows the SQL **WHERE** clause associated with a collection to be specified or overridden and the **sortExpression** property allows the SQL **ORDER BY** clause to be specified or overridden. These expressions are combined with the column list and table selection expressions to build an SQL query.

For example, you could use these properties to create a shared external collection instance, set the filter and sort expressions, and then use an iterator or **foreach** instruction to fetch the instances, as shown in the following example.

```
findRichCustomers();
vars
    accounts : CustomerAccountSet;
    account  : Account;
begin
    create accounts;
    accounts.filterExpression := "account.balance > 100000";
    accounts.sortExpression   := "account.balance";
    foreach account in accounts do
        display(account.number, account.name);
    endforeach;
end;
```

For more details, see "ExternalCollection Class", in Chapter 1 of the *JADE Encyclopaedia of Classes*.

Sequential Access

External iterators implement the standard iterator protocol, and you can use them directly.

Standard JADE **foreach** instruction syntax is fully supported, including the **reversed** and **where** clauses.

Exception Handling

The **ODBCException** class provides additional information specific to the ODBC interface for external database access.

Changes to the external relational database schema that may become inconsistent with the schema mapping are detected and handled.

Schema Mismatch Exceptions

Special consideration is given to detecting and handling changes to the relational schema that may become inconsistent with the schema mapping. In some cases, changes are handled transparently. For example, when a column is added or the column type is changed and a valid type mapping still exists, the query engine creates a proxy, mapping the actual columns to the attributes defined in JADE.

Some types of changes to the relational schema render mappings or queries invalid; for example, when a table that is mapped to an external class is deleted.

When a mapping or query that is no longer valid is referenced, the appropriate ODBC exception is raised.

When the schema mapping becomes so out of date that it is unusable, you must return to the External Schema Wizard to update the mapping information.

Optimizing the *where* Clause

The JADE **foreach** instruction optimizes special cases when the predicate following the **where** clause consists of a conditional key expression, including when iterating with a **foreach** instruction over an external proxy collection. (For details, see "*where* Clause Optimization", in Chapter 1 of the *JADE Developer's Reference*.)

A special case occurs when the predicate following the **where** clause contains operands that are all attributes of the collection member type. In this case, the **where** predicate from the **foreach** instruction can *become* the **where** predicate in the SQL statement or it can be joined with an existing **where** predicate in the query specification associated with the external collection.

Updating External Databases

Positioned (cursor-based) or non-cursor updates are supported by external collection methods to create, update, or delete a current external proxy object, as follows.

- Non-cursor updates

  The **ExternalDatabase** and **ExternalObject** classes provide transaction support and direct SQL execution. The **executeSQL** method of the **ExternalDatabase** class enables you to perform searched updates.

- Cursor-based updates

  Positioned (cursor-based) updates are supported by **ExternalCollection** class methods to create, update, or delete a current proxy object.

Use the colon and equals symbols (:=) assignment operator to update the attributes of a proxy. (For details, see "Assignments", in Chapter 1 of the *JADE Developer's Reference*. See also "Accessing Objects in Your External Database", earlier in this chapter.)

Security at Run Time

Your external relational database can have table-level or column-level security enforced.

If a user does not have the required access rights for selected columns in a table, these columns are initialized to JADE **null** values in the external proxy class.

If a specific query is not possible for security reasons, an ODBC exception is raised.

Obtaining a Relational View of Your JADE Database

The JADE ODBC (Open Database Connectivity) standard driver and thin client driver are provided to enable you to use SQL statements to access a relational view of your JADE database. You can use a JADE ODBC driver with any tool that accesses databases using ODBC (for example, MS Query or Crystal Reports).

The JADE Relational Views Wizard enables you to create relational views of your JADE database. For details about using this wizard to define relational views, see Chapter 9, "Defining ODBC Inquiry Relational Views and Ad Hoc Indexes", in the *JADE Development Environment User's Guide*.

For guidelines about query optimization, soft attributes, and thin client and application server queries, see the *Relational Queries Using ODBC* white paper on the JADE Web site at https://www.jadeworld.com/developer-center/resource-library/white-papers.

The JADE ODBC drivers are installed as part of the JADE installation process. When you have created a relational view of your JADE database, you must then configure an ODBC data source in order to access your relational view. For details, see "Configuring an ODBC Driver", later in this chapter.

The JADE ODBC drivers are invoked by another application, often a third-party generic tool (for example, a report writer) to access data in the JADE database. A JADE ODBC driver differs from other JADE invocations in that it has no command line arguments passed to it and it has no control over the directory from which it is invoked. It must therefore obtain all of its session parameters from the data source configuration or environment variables.

The JADE ODBC thin client establishes a TCP/IP connection with a user-defined ODBC server application running in a JADE node. The JADE ODBC standard (fat) client runs as a JADE client node, establishing communication with a JADE database server in the same way as any other JADE standard client.

The JADE ODBC driver is a Core Level implementation of an ODBC Version 3.51 driver. The ODBC driver accepts only SQL keywords that are all uppercase characters (for example, **LIKE**), or optionally keywords with an initial capital letter (for example, **Like**, but not **like** or **liKe**) or keywords that are all lowercase characters (for example, the SQL Server keywords). The case of identifier characters used in a query must be distinct from those in keywords.

The JADE ODBC drivers are available in 32-bit and 64-bit versions. If running on a 64-bit machine, the driver used must match the third-party tool being used; for example, it may be necessary to install 32-bit JADE ODBC drivers for use with 32-bit tools.

Notes The **OidFieldSeparator** parameter in the [JadeOdbc] section of the JADE initialization file on the server enables you to customize the OID field separator with a single punctuation or similar character (for example, **@**). This can be useful when the default OID String values that are produced are misinterpreted by a third-party tool as a different ODBC type such as a Decimal.

Time attributes are handled differently in Access 97 (where they are retrieved as a character string) and Access 2000 (where they are retrieved and displayed as a time stamp). This causes differences in the display of time fields.

The ODBC driver supports SQL Minimum Grammar, including the following.

- **=**, **>**, **<**, **<=**, and **>=** comparison operators
- **LIKE** pattern matching operator
- **NOT**, **AND**, and **OR** boolean operators
- **SELECT**, **FROM**, **WHERE**, and **ORDER BY** clause

In addition to the SQL Minimum Grammar, the ODBC driver supports the following.

- **GROUP BY** and **HAVING** clauses
- The following aggregate functions
  - **COUNT**(*) and **COUNT**([ALL|DISTINCT] columnName)
    - columnType is **SQL_BIGINT**
  - **SUM**([ALL | DISTINCT] columnName) and **AVG**([ALL | DISTINCT] columnName)
    - Valid for numeric types only
    - columnType is dependent on type of columnName
    - **SQL_TINYINT** or **SQL_INTEGER** : columnType is **SQL_INTEGER**
    - **SQL_BIGINT** : columnType is **SQL_BIGINT**
    - **SQL_DECIMAL**(p, s) : columnType is **SQL_DECIMAL**(23, s)
  - **MIN**([ALL | DISTINCT] columnName) and **MAX**([ALL | DISTINCT] columnName)
    - Valid for all types except **SQL_BIT**
    - columnType is the same as columnName type
- Simple Joins and left and right outer Joins, but it does *not* support Nested Queries

- **UNION** clause, which combines the results of two queries when the queries have the same number of columns and the selected columns are of the same data type
- **SELECT** clause **AS<*column-name*>** column aliasing predicate that can be used in the **ORDER BY** clause

For SQL Server queries, you must enclose all table and field names that conflict with SQL keywords with double quotation (**""**) characters. (See the Query Analyzer help for details.) The third sheet of the System DSN wizard in the ODBC Data Source Administrator has the configurable **Use ANSI quoted identifiers** check box, which enables RPS to interrogate the RPS ODBC data source that it uses to determine if to quote SQL identifiers (that is, table, column, and procedure names). This allows reserved T-SQL keywords to be used in SQL statements, which is useful if you want a property to have the same name as a T-SQL reserved word.

Notes When importing tables from JADE using Microsoft Access, do not select any field as the **Unique Record Identifier**. Although the object identifier (oid) seems the natural choice for this, Access performs validation that causes unnecessary problems and it does not work at all with Access 2000 and Windows 98. The Unique Record Identifier is useful to Access only when updating tables, which is not allowed in the JADE database.

When using the JADE ODBC standard client and setting the **TerminateProcessOnDisconnect** parameter in the [JadeClient] section of the JADE initialization file to **true**, unwanted side-effects may occur when the connection to the JADE database server is disconnected, as the underlying program that opened the connection is also terminated.

When using the JADE ODBC standard client, we recommend that cache coherency is used in the ODBC node, to ensure use of the latest edition.

The output of query execution tracing to the **jommsg.log** file is switched on or off using the **QueryExecutionTraceOn** parameter in the [JadeOdbc] section.

The values returned by the JADE ODBC driver for a **Date** attribute are listed in the following table.

Attribute Value	Returned Value
Valid date	Valid date
Value not initialized	Null value
Value invalid	Current date (with non-fatal warning)

The values returned by the JADE ODBC driver for a **Time** attribute are listed in the following table.

Attribute Value	Returned Value
Valid time	Valid time
Value not initialized	00:00:00 (default value == 0)
Value invalid	00:00:00 (with non-fatal warning)

The values returned by the JADE ODBC driver for a **TimeStamp** attribute are listed in the following table.

Attribute Value	Returned Value
Valid date/time	Valid date/time
Value not initialized	Current date 00:00:00
Date not set/time set	Current date and time (as in ODBC specifications)
Date set/time not set	Valid date/00:00:00
Invalid values	Current date 00:00:00 (with non-fatal warning)

Configuring a JADE ODBC Driver

The JADE ODBC drivers are installed as part of the JADE installation process. The following subsections explain how to configure a JADE ODBC driver.

As the JADE ODBC driver is invoked by third-party applications, it is unlikely that your current working directory will identify the path for the JADE libraries.

If your JADE libraries are not located in your current working directory, the libraries are loaded from the same directory as the JADE ODBC driver. If the libraries are not found in that directory, exception 8327 is raised.

Windows ODBC drivers are run from the directory specified by the following registry values.

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\OBDCINST.INI\JADE ODBC Driver\Driver

HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\OBDCINST.INI\JADE ODBC Driver\Setup
```

Each data source may also store where to find its driver directly, as follows.

```
HKEY_CURRENT_USER\SOFTWARE\ODBC\OBDC.INI\Data-Source\Driver
```

Tips If the operating system error 126 (*ERROR_MOD_NOT_FOUND*), error 1157 (*ERROR_DLL_NOT_FOUND*), or JADE error 8327 *(Directory location of the Jade ODBC library in the DSN is not the Jade Install Directory*) is raised, check that the ODBC driver has been installed by accessing the Control Panel **ODBC** applet to determine if the JADE driver is listed on the **ODBC Drivers** sheet. If not, install the driver by using the JADE installation medium or **odbcinst**.

If the JADE driver exists but it is listed as **Not Marked** or it is the incorrect version, the registry settings specified earlier in this section are pointing at the wrong directory and need to be corrected. If the version is correct, check that the directory contains valid JADE library files.

Adding a Data Source

To add a new data source

1. Run the Microsoft Data Source Administrator program from the Control Panel **Administrative Tools** option **Data Sources (ODBC)** applet.

   If you are running JADE on a 64-bit machine under Windows in 64-bit mode and you are configuring a 32-bit tool, run the following program:

   ```
   <\windows-directory>\SysWOW64\odbcad32.exe
   ```

   This runs the 32-bit version of the Microsoft Data Source Administrator program rather than the 64-bit version.

2. To add a JADE data source for all but SQL Server, select the **User DSN** tab. For SQL Server, select the **System DSN** tab.

3. Click the **Add** button. The Create New Data Source dialog is then displayed. The list box displays the following JADE ODBC drivers:

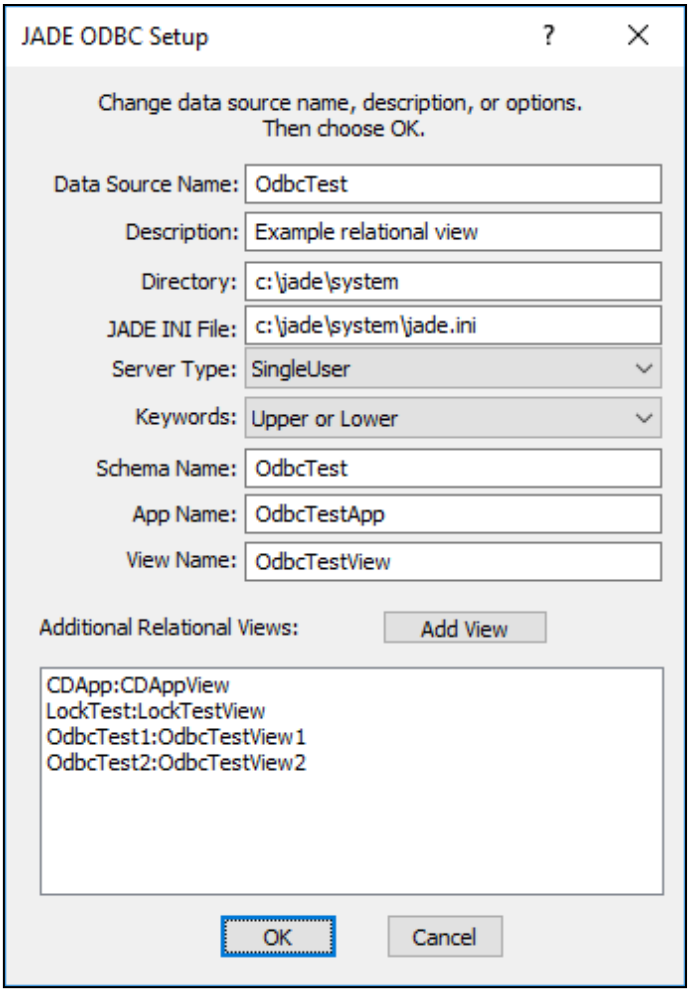
   - A generic standard driver that is updated with each JADE release
   - A version-specific standard driver that is specific to a release of JADE
   - The thin client driver

4. Select the appropriate JADE ODBC driver.
5. Click the **Finish** button.

You can then configure the JADE ODBC standard client or thin client driver. For details, see "Configuring the JADE ODBC Standard Client Driver" and "Configuring the JADE ODBC Thin Client Driver", in the following subsections.

Configuring the JADE ODBC Standard Client Driver

The JADE ODBC Setup dialog, shown in the following image, is displayed when you have added a new data source for a JADE ODBC standard client driver on the Create New Data Source dialog.



To configure the JADE ODBC standard client data source

1. In the **Data Source Name** text box, specify your data source name. You can specify any name that you choose, to identify this data source.
2. In the **Description** text box, specify a description of the data source. This description is for your own documentation for the data source.
3. In the **Directory** text box, specify the directory that is the location of your JADE database. In multiuser mode,

this is the location of the database on the server.

Tip If you have the server workstation mapped to a different drive, you would still specify the drive that is seen by the server workstation as the database location, regardless of the drive mapping of individual client workstations. This is the same directory specified in the **path** parameter when running a standard client on that machine.

4. In the **JADE INI File** text box, specify the full path and file name of the JADE initialization file on the client node. The initialization file defaults to **jade.ini**, located in your database (system) path.
5. In the **Server Type** list box, select the required server type. The default server type is **SingleUser**.
6. In the **Keywords** list box, the **All Upper** option is selected by default and specifies that only reserved words (or keywords) that are all uppercase characters are recognized; that is, they adhere to the ODBC standard and minimize reserved word conflicts with your user table or column names. (For details about the ODBC reserved words, see "ODBC Reserved Words", in Appendix A.)

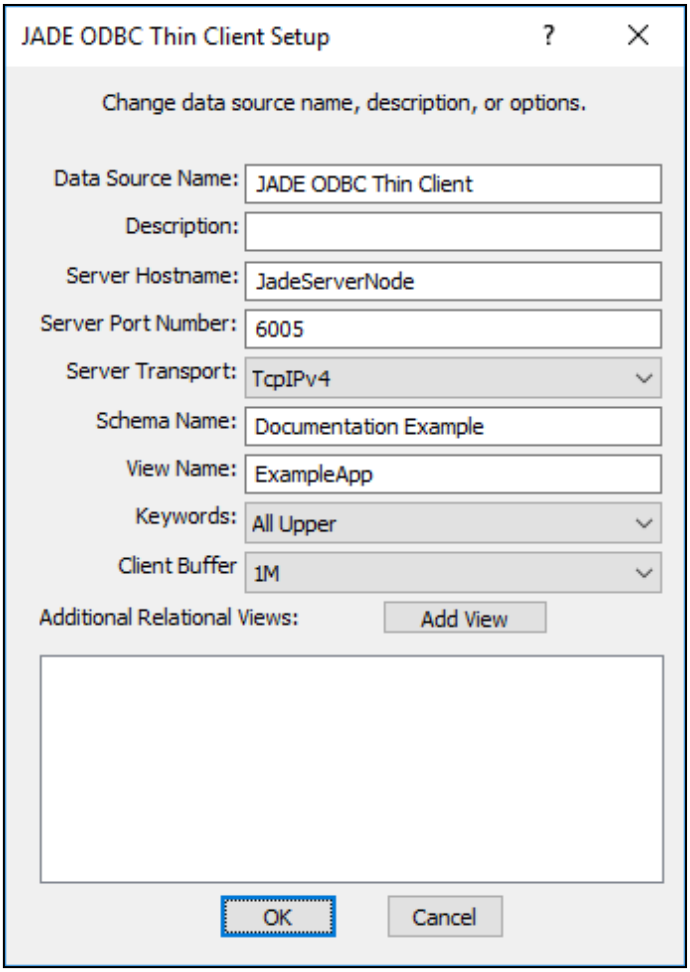
   Select the **Initial Upper** option if you want all ODBC reserved words converted to an initial capital letter so that words such as **Like** (which is used by some applications) are then recognized.

   Select the **Upper or Lower** option for SQL Server, whose keywords are sent to the ODBC driver in lowercase format.
7. In the **Schema Name** text box, specify the name of the schema from which the data for your relational view is to be sourced.
8. In the **App Name** text box, specify the name of the application from which the data for your relational view is to be sourced.
9. In the **View Name** text box, specify the name of the required relational view. The relational view that you specify must already exist. For details, see "Defining Your Relational View", in Chapter 9 of the *JADE Development Environment User's Guide*.
10. If you want to define multiple schema and relational view pairs for a DSN, click the **Add View** button. For details, see "Specifying Additional Relational Views", later in this chapter.
11. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

You can now use the DSN to access data defined in the relational view.

Configuring the JADE ODBC Thin Client Driver

The JADE ODBC Thin Client Setup dialog, shown in the following image, is displayed when you have added a new data source for a JADE ODBC thin client driver on the Create New Data Source dialog.



To configure the JADE ODBC thin client data source

1. In the **Data Source Name** text box, specify your data source name. You can specify any name that you choose, to identify this data source.
2. In the **Description** text box, specify a description of the data source. This description is for your own documentation for the data source.
3. In the **Server Hostname** text box, specify the name of the host that is used to connect to a JADE ODBC service application.
4. In the **Server Port Number** text box, specify the port number for which the ODBC service application accepts connections.

5. In the **Server Transport** combo box, the **TcpIPv4** option is selected by default, and specifies the transport type to connect to the server node. The server transport values available for your ODBC thin client connection are:

   - **TcpIPAny**, which connects using IP version 4 or 6.
   - **TcpIPv4**, which connects using IP version 4.
   - **TcpIPv6**, which connects using IP version 6.

   For the **TcpIPAny** transport type, the client will first attempt to connect via IP version 6 and then IP version 4 protocol on the provided IP addresses. Each connection failure will be logged, and the next available combination tried. For details about the TCP/IP transport, see "TCP/IP Transport", in Chapter 2 of your *JADE Installation and Configuration Guide*.

6. In the **Schema Name** text box, specify the name of the schema from which the data for your relational view is to be sourced.

7. In the **View Name** text box, specify the name of the required relational view. The relational view that you specify must already exist. For details, see "Defining Your Relational View", in Chapter 9 of the *JADE Development Environment User's Guide*.

8. In the **Keywords** combo box, the **All Upper** option is selected by default and specifies that only reserved words (or keywords) that are all uppercase characters are recognized; that is, they adhere to the ODBC standard and minimize reserved word conflicts with your user table or column names. For details about the ODBC reserved words, see "ODBC Reserved Words", in Appendix A.

   Select the **Initial Upper** option if you want all ODBC reserved words converted to an initial capital letter so that words such as **Like** (which is used by some applications) are then recognized.

   Select the **Upper or Lower** option for SQL Server, whose keywords are sent to the ODBC driver in lowercase format.

9. In the **Client Buffer Size** combo box, the **1M** option is selected by default and specifies the size of the client buffer. The size of the thin client buffer determines the maximum data transfer size for the result set from the ODBC service application to the presentation client.

   The actual maximum size is the minimum of the selected client buffer size and the value of the **ServerResultSetBufferSize** parameter in the [JadeOdbc] section of the JADE initialization file.

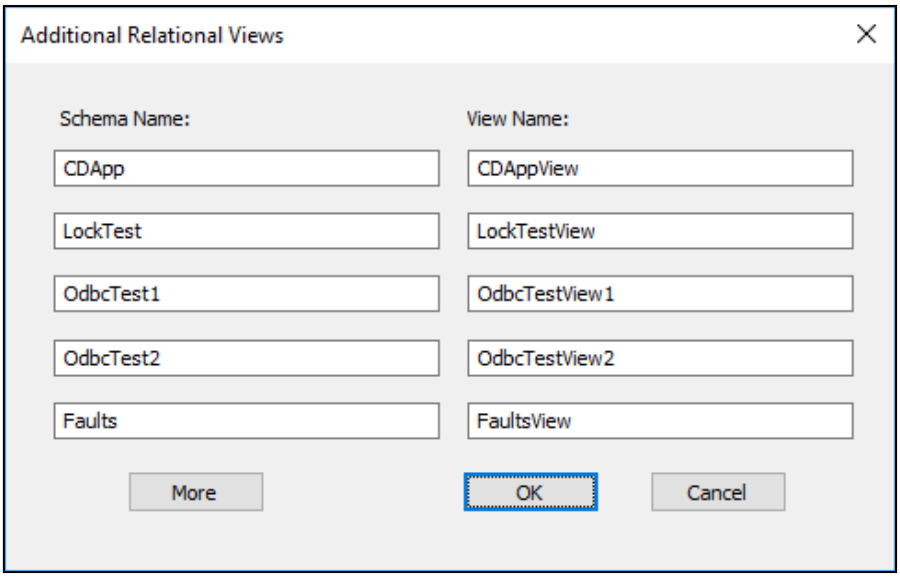
   If you want to change the size of the client buffer, select the required value from the **1M** (the default), **2M**, **5M**, and **10M** items in the drop-down list.

10. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

You can now use the DSN to access data defined in the relational view.

Specifying Additional Relational Views

If you want to define multiple schema and relational view pairs for a DSN and you clicked the **Add View** button on the JADE ODBC Setup dialog, the Additional Relational Views dialog, shown in the following image, is then displayed.



To specify additional relational views

1. In the **Schema Name** and **View Name** pairs of text boxes, enter the schema and relational view names of each pair that you require for the DSN.

   When you have defined all additional schema and relational view pairs that you require, click the **OK** button to return to the JADE ODBC Setup dialog. All additional relational views for the DSN are then displayed in the **Additional Relational Views** list box.

   The tables in all relational views are available to the caller. It is your responsibility to ensure that there are no name conflicts in the included relational views, as there is no check for the same table name included in multiple relational views. If there is a table name conflict, an exception is raised if the table is accessed.

   To avoid table name conflicts across multiple views and schemas, you can use the **Add Class Prefix** item in the **Modifier** list box and specify the prefix in the **Value** box on the ninth sheet of the Relational View Wizard to prefix table names with a unique code.

   The ODBC driver logs into JADE using the top-level schema name and application name defined in the DSN (that is, specified in the **Schema Name** and **App Name** text boxes on the JADE ODBC Setup dialog). Any methods called that are not in the application schema may raise an exception if they access the **global** or **app** system variable.

   The log-in application security method called, if any, is defined in the top-level relational view for tables in all schemas.

2. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Using the Data Source in a Connection String

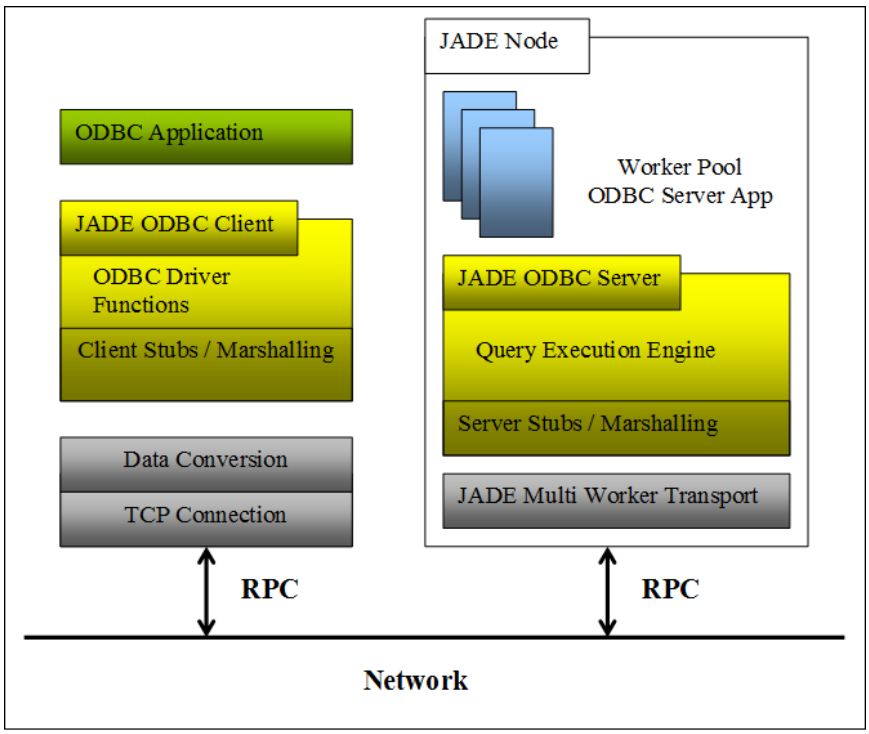
In some cases, you may be required to specify an ODBC connection string. The syntax of the connection string is as follows.

```
DSN=DSN-name;UID=userid;PWD=password;
```

JADE ODBC Thin Client Driver

The JADE ODBC thin client driver must be installed and configured on the machine on which the ODBC tool is to be run. This driver (**jadodbc_c.dll**) must be installed with all JADE libraries included in a thin client installation.

The server side of the ODBC thin client driver is a user-defined ODBC service application that can be run in any JADE node.



JADE ODBC Service Application

The JADE ODBC service application is a user-defined non-GUI application that provides TCP/IP connection to the JADE database for the JADE ODBC thin client driver. The node providing the service can be a standard client, application server, or database server node.

The server-side network transport and connection management are provided by the JADE multiple-worker TCP transport mechanism. The multi-worker infrastructure enables a number of client connections to be supported by a smaller number of worker processes. ODBC queries are executed by a worker process selected from a pool of worker processes.

User-Defined ODBC Service Application

To define an ODBC service application:

- Define a non-GUI application in the schema that defines the relational view that is being used in the ODBC DSN.
- The application must call the following method from its **initialize** method.

```
Application::odbcWorkerInitialize
```

- The application must call the following method from its **finalize** method.

```
Application::odbcWorkerFinalize
```

The **odbcWorkerIntialize** method processes the ODBC server configuration parameters, starting additional copies of the application as required.

When ODBC initialization has completed successfully, the service is ready to accept client connections on the configured TCP listen port. Each worker application joins the transport group associated with the ODBC service for a schema and relational view combination.

Configuring an ODBC Service

The following subsections cover configuring an ODBC service.

Defining the Configuration in the JADE Initialization File

The [JadeOdbcServer] section of the JADE initialization file contains the ODBC server configuration parameters; for example:

```
[JadeOdbcServer]
ListenHostName          = localHost
ListenPort              = 53099
ReadTimeout             = 0
MinWorkers              = 1
MaxWorkers              = 1
QueueDepthLimit         = 1
QueueDepthLimitTimeout  = 2
WorkerIdleTimeout       = 120
```

To support multiple schema and relational view combinations, the [JadeOdbcServer] section of the JADE initialization file can include the **ApplicationConfigFile** parameter containing the name of an XML configuration file, which can define the configuration parameters for multiple JADE ODBC service applications; for example:

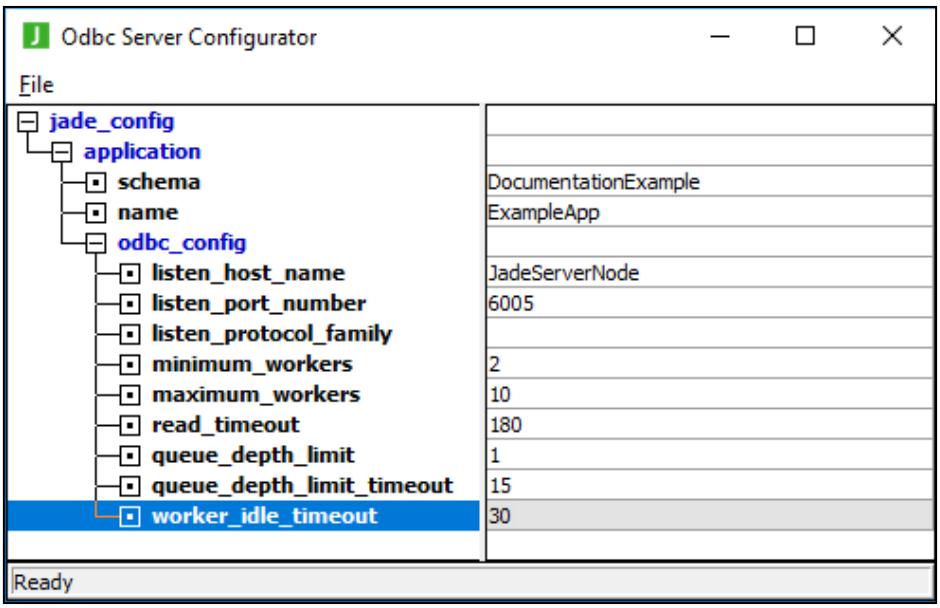
```
[JadeOdbcServer]
ApplicationConfigFile=D:\jadetest\database\A6300\odbc_config1.xml
```

Creating and Maintaining the Configuration File

Use the JADE Monitor schema **OdbcServerConfigurator** application to create and maintain the XML configuration file, as shown in the following example.

```
jade.exe path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeMonitorSchema
app=OdbcServerConfigurator
```

The **OdbcServerConfigurator** application provides the Odbc Server Config dialog, shown in the following image, which enables you to define sections in the XML file for multiple schema and application name combinations, where the application name matches the name of your user-defined ODBC service application.



To configure the ODBC server

1. In the File menu, select the:
   - **New** command, if you want to add a new ODBC server configuration. The dialog then displays in black the application and ODBC configuration parameter values. When you add and save an ODBC server configuration, a file with a default name of **config<*n*>.xml** is output to the path that you specify. (The **<*n*>** value is incremented with each subsequent configuration file.)
   - **Open** command, if you want to maintain an existing ODBC server configuration. The common Open dialog is then displayed, to enable you to select the **config<*n*>.xml** ODBC configuration file that you required. When you have selected the appropriate file from its destination and clicked the **Open** button, the Odbc Server Config file dialog is then populated with the parameter values from that file.
   - **Append New** command, if you want to add a new ODBC server configuration to the end of an existing configuration file. (This command is enabled only when a new or an existing ODBC server configuration is displayed in the dialog.)
2. In the **schema** and **name** rows, specify the name of the schema and application from which the data for your relational view is to be sourced, respectively.
3. Click on the **odbc_config** node to expand it, and then specify the following optional parameter values to meet your requirements.
   a. The optional **listen_host_name** parameter specifies the host name or IP address of the local interface card on which to listen. If you do not specify this parameter or set it to **0.0.0.0**, the ODBC service listens on all interfaces.
   b. The **listen_port_number** parameter specifies the listener port number for which the ODBC server transport accepts connections. This must be a unique port number in the range 1024 through 65534 that does not conflict with any other TCP service running on the machine.
   c. The **read_timeout** parameter specifies in seconds the length of time to wait before a network read

request is terminated. The default value is **120** seconds.

The value of this parameter is used as the **JadeMultiWorkerTcpConnection** class **timeout** property value. The parameter value is translated from seconds in the **read_timeout** parameter to milliseconds in the **timeout** property for the TCP/IP connection between the thin client ODBC driver and the ODBC server application.

d. The following dynamic worker pool parameters control the minimum and maximum number of ODBC applications workers, when new workers are started, and when idle workers are terminated.

- **minimum_workers**
- **maximum_workers**
- **queue_depth_limit**
- **queue_depth_limit_timeout**
- **worker_idle_timeout**

When the number of outstanding requests queued for an ODBC worker application exceeds the number specified in the **queue_depth_limit** parameter for the time period specified by the **queue_depth_limit_timeout** parameter and fewer than the value specified in the **maximum_workers** parameter copies are running, an additional worker will be started.

When an ODBC worker application has been idle for the number of seconds specified in the **worker_idle_timeout** parameter and there are more than the number of copies specified in the **minimum_workers** parameter, an idle worker will be terminated.

4. In the File menu, select the **Save** command, if you want to save the configuration file. The common Save As dialog is then displayed, to enable you to select the file location and name. The default name of **config<*n*>.xml** is output to the path that you specify.

5. In the File menu, select the **Save As** command, if you want to save the configuration file to another location or name. The common Save As dialog is then displayed, to enable you to select the file location or a different name.

6. In the File menu, select the **Exit** command, to close the Odbc Server Config dialog the JADE Monitor.

XML Configuration for an ODBC Server Application

The configuration file for ODBC server applications has the following format. For details about these values, see steps 2 and 3 under "Creating and Maintaining the Configuration File", in the previous section.

```
<?xml version="1.0" ?>
<jade_config>
  <application schema="DocumentationExample" name="ExampleApp">
    <odbc_config>
      <listen_host_name>JadeServerNode</listen_host_name>
      <listen_port_number>6005</listen_port_number>
      <minimum_workers>2</minimum_workers>
      <maximum_workers>10</maximum_workers>
      <read_timeout>180</read_timeout>
      <queue_depth_limit>0</queue_depth_limit>
      <queue_depth_limit_timeout>15</queue_depth_limit_timeout>
      <worker_idle_timeout>30</worker_idle_timeout>
    </odbc_config>
  </application>
</jade_config>
```

Installing the JADE ODBC Thin Client Driver

The JADE ODBC thin client driver is installed as part of the standard JADE installation.

To install the driver (**jadodbc_c.dll**) manually, run **odbcinst.exe**, supplying the following arguments.

```
registryKey=<> binDir=<> dllName=<> [deinstall]
```

The **registryKey** argument is the name of the ODBC driver that is displayed in the **Data Source Administrator Driver** tab of the ODBC Data Source Administrator dialog, the **binDir** argument is the directory in which the driver DLL is located, the **dllName** argument is the name of the driver DLL, and the optional **deinstall** argument is an argument to deinstall the driver.

The following is an example of the command line.

```
odbcinst registryKey="Jade ODBC Thin Client" binDir=d:\jade\bin dllName=jadodbc_
c.dll
```

User Authentication

When a new ODBC client connection is opened in a JADE ODBC service application, the **Global** class **isUserValid** method is invoked with the login user code and password passed as parameters.

If the method returns **false**, the client connection is refused.

Starting and Ending an ODBC Thin Client Session

The following **Application** class methods can be reimplemented, if required.

- startOdbcSession(rv: RelationalView; username: String);

  This method is called in an ODBC service application after the **isUserValid** method of the **Global** class to indicate a new session has been established.

- endOdbcSession(sessionObject: Object);

  This method is called in an ODBC service application when a session is closed. The value of the **sessionObject** parameter is the session object set by the application using the **setOdbcSessionObject** method or else it is null.

Initialize and Finalize Query Callback

The following methods in the **Application** class can be reimplemented, if required.

- initializeOdbcSelect(rv: RelationalView; username: String);

  This method is called in an ODBC service application before the execution of an SQL query. The **rv** parameter specifies the relational view currently in use and the **username** parameter specifies the user code of the logged-on user. This method can be reimplemented as required; for example, to re-establish application state for the user executing the query.

- finalizeOdbcSelect(rv: RelationalView; username: String);

  This method is called in an ODBC service application at the end of the execution of an SQL Query. The **rv** parameter specifies the relational view currently in use and the **username** parameter specifies the user code of the logged-on user. This method can be reimplemented as required; for example, to clean up any transient objects left from the execution of the query.

User Impersonation

When an ODBC worker process is calling user methods to satisfy a query in the execution phase, the value of the **Process** class **userCode** property is set to the user code with which the connected user logged on with.

Session State

The following methods defined in the **Application** class can be called in an ODBC service application to set and get a reference to an application-maintained session object.

- setOdbcSessionObject(object: Object);

  This method would typically be called in the **Application** class **startOdbcSession** method, to save an application-defined context object for the user. The ODBC service application maintains this object on behalf of the client that is currently executing a query.

- getOdbcSessionObject(): Object;

  This method returns the last saved session object. Obtaining a saved session object reference using the **getOdbcSessionObject** method in an ODBC server application is analogous to dereferencing the **currentSession** environmental variable in a Web application.

Server/Client Version Checking

The ODBC thin client:

- Checks the validity of the versions of **jadodbc_c**, **jomos**, **jomjdi**, **jadcnet**, and **jomutil** across the client-server at start up. An error is raised if the versions do not match.
- Is updated as part of the JADE thin client automatic download. If no thin client connectivity is available, the files must be updated by the system administrator.

JADE ODBC Driver Soft Entities and Attributes

The JADE ODBC driver enables you to:

- Include user-defined (or *soft*) entities in a relational view
- Include user-defined (or *soft*) attributes of real classes or of entities in a relational view
- Implement a query provider that, given a class or soft entity and a search predicate expression (or **WHERE** clause), can return a result set of objects that will be used by the ODBC driver

To achieve a soft exposure to your application:

1. Implement the following **RootSchema** interfaces.
   - **JadeRelationalEntityIF**
   - **JadeRelationalAttributeIF**
   - **JadeRelationalQueryProviderIF** (optional)
2. Call methods in the **RelationalView** class, to add entities and attributes to the view.

JadeRelationalEntityIF Interface

The **JadeRelationalEntityIF** interface instance is passed to the **addUserTable** method of the **RelationalView** class to add a soft entity to a relational view.

The values returned for the **getJadeClass**, **getSQLName**, and **callIFAllInstances** methods at the time of the method call are stored in the schema meta data for the relational view. If these values change, you must remove the attribute and then add it again to update the meta data.

The **allInstances**, **getPropertyValue**, **getQueryProvider**, and **isAttributeValid** interface methods are called at run time to access the data in the JADE database.

JadeRelationalAttributeIF Interface

To expose soft attributes, implement **RootSchema** **JadeRelationalAttributeIF** interface.

The **JadeRelationalAttributeIF** interface instance is passed to the **addUserAttribute** method of the **RelationalView** class to add a soft attribute to a user-defined entity. The values returned at the time of the method call are stored in the schema meta data for the relational view. If these values change, you must remove the attribute and then add it again to update the meta data.

JadeRelationalQueryProviderIF Interface

To provide a search implementation that optimally finds and filters instances of a soft entity, implement the **JadeRelationalQueryProviderIF** interface. The interface implementation is used if the **JadeRelationalEntityIF** interface returns a reference to it in the **getQueryProvider** method.

The query provider interface provides methods to map the SQL search expression to an application representation of the expression. The resultant expression is passed back to the application to execute the query.

SQL Examples

The following subsections give some examples of SQL statements that can be applied to relational views that have been created from a fictitious JADE database.

Accessing Classes and Subclasses Using SQL

If you were to create a table from a class called **Book**, the following SQL statement lists all books.

```
SELECT * FROM Book
```

If **Book** has subclasses **Fiction** and **Nonfiction**, and these are included in your view, the following SQL statement lists all fiction books.

```
SELECT * FROM Fiction
```

Accessing Relationship Views Using SQL

The following examples show SQL statements required to access views of JADE classes that have relationships to each other.

One-to-One Relationship

In this example, the **Account** and **Password** classes have been included in the view. The **Account** class has a **password** property.

The SQL statements in the following example access account number 12345.

```
SELECT * FROM Account, Password
    WHERE Account.password = Password.oid
    AND Account.code = '12345'
```

One-to-Many Relationship

In this example, the **Customer** and **Invoice** classes have been included in the view. The **Customer** class has an **invoices** property that is a collection. The **Invoice** class has a **customer** property.

The SQL statements in the following example access customer 123456.

```
SELECT * FROM Invoice, Customer
    WHERE Invoice.customer = Customer.oid
    AND Customer.code = '123456'
```

Many-to-Many Relationship

In these examples, the **Book** and **Author** classes have been included in the view. The **Author** class has an **allBooks** property that is a dictionary. The **Book** class has an **authors** property that is an array. These examples also create the **Book_authors** table, which contains **author_oid**, **book_oid**, and **number**.

The SQL statements in the following example list a book and all its authors.

```
SELECT * FROM Book, Book_authors, Author
    WHERE Book.oid = Book_authors.book_oid
    AND Book_authors.author_oid = Author.oid
    AND Book.name = 'War of the Worlds'
```

The SQL statements in the following example list the first author of all books.

```
SELECT * FROM Book, Book_authors, Author
    WHERE Book.oid = Book_authors.book_oid
    AND Book_authors.author_oid = Author.oid
    AND Book_authors.number = 1
```

The SQL statements in the following example list an author and its books.

```
SELECT * FROM Author, Author_allBooks, Book
    WHERE Author.oid = Author_allBooks.author_oid
    AND Author_allBooks.book_oid = Book.oid
    AND Author.name = 'HG Wells'
```

Collections

In these examples, the **Password** class has a **prevPasswords** string array property. A relational table is created called **Password_prevPasswords**, with **password_oid**, **value**, and **number** data items.

The SQL statements in the following example list the previous passwords for a current password of **abcd**.

```
SELECT * FROM Password, Password_prevPasswords
    WHERE Password.oid = Password_prevPasswords.password_oid
    AND Password.currentvalue = 'abcd'
```

The SQL statements in the following example list the most-recent previous password for the **abcd** password.

```
SELECT * FROM Password, Password_prevPasswords
    WHERE Password.oid = Password_prevPasswords.password_oid
```

```
AND Password.currentvalue = 'abcd'
AND Password_prevPasswords.number = 1
```

Collection Methods

You can create a derived table from a method that returns a collection of objects. The parameters (if any) of such a method must be JADE primitive types whose usage is **input** or **constant**.

Notes If the collection method returns a transient collection, that collection is deleted when the query has completed.

A table derived from a method returning a collection has two columns of object identifiers (oids): one maps to the instance of the object on which the methods is defined and the other maps to instances of the collection returned by the method.

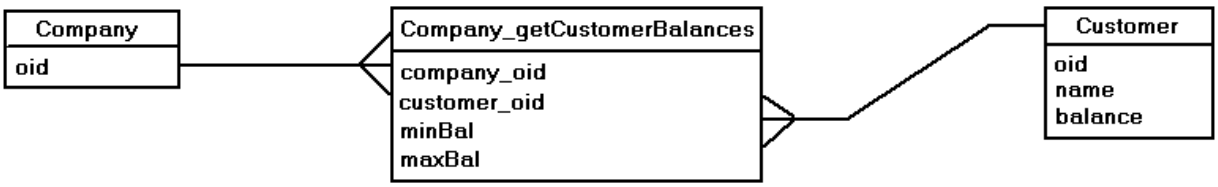
In addition, a defined column corresponds to each of the parameters in the methods. For example, a **Company** class method **getCustomerBalances(minBal: Decimal; maxBal: Decimal);** that returns a list of customers whose outstanding balance is in the range **minBal** less than or equal to **bal** less than or equal to **maxBal** creates a derived table called **Company_getCustomerBalances** that can have the following four columns.

- company_oid
- customer_oid
- minBal
- maxBal

The **OidFieldSeparator** parameter in the [JadeOdbc] section of the JADE initialization file on the server enables you to customize the OID field separator with a single punctuation or similar character (for example, **@**). This can be useful when the default OID String values that are produced are misinterpreted by a third-party tool as a different ODBC type such as a Decimal.

The derived table represents a virtual collection of customers whose outstanding balance is in the range **minBal** through **maxBal**. The values of **minBal** and max **Bal** are included in the **WHERE** clause of the select statement and are used as the parameter values passed to the **getCustomerBalances** method.

The logical relationship between **Company**, **Company_getCustomerBalances**, and **Customer** is shown in the following Entity-Relationship Diagram (ERD).



The following SQL statement selects customers with a balance in the range $500 through $5000.

```
SELECT Customer.name, Customer.balance
FROM Company, Customer, Customer_getCustomerBalances
WHERE Company.oid=Company_getCustomerBalances.company_oid AND
    Customer.oid=Company_getCustomerBalances.customer_oid AND
    Company_getCustomerBalances.minBal = 500 AND
    Customer_getCustomerBalances.maxBal = 5000
```

When specifying table columns in the **WHERE** clause that correspond to the parameters of a collection method, any comparison must be for equality (that is, the equals symbol only) and used in an **AND** operation. (An error is raised if this is not the case.) If a parameter is not specified, the method is passed a NULL value for that parameter.

Exceptions raised by methods that are mapped to columns are logged if the **LogUserMethodExceptions** parameter in the [JadeOdbc] section of the JADE initialization file is set to **true**. The result of a method column that raises an exception is by definition a NULL value. Using a NULL value in a comparison always returns a FALSE result.

Tip For methods that are mapped to ODBC columns, it is better to handle exceptions within your user code and return a valid value rather than propagating the error out to the ODBC driver code.

Although the following SQL statement would achieve the same result as the previous example, using the **getCustomerBalances** method may significantly reduce the io required to complete the query. (This would be particularly relevant if **getCustomerBalances** were returning an existing collection and not building one on each invocation of the method.)

```
SELECT Customer.name, Customer.balance
FROM Company, Customer
WHERE Customer.company=Company.oid AND
    Customer.balance >= 500 AND
    Customer.balance >= 5000
```

Mapping Your JADE Database to a Relational Database

The Relational Population Service (RPS) provides automatic replication of objects from a production JADE database to one or more relational databases when running an RPS node.

The Relational Population Service wizard provides facilities that enable you to map classes and properties to the required relational tables and columns.

All mapping meta data becomes part of the system specification in the JADE database, and can be extracted, loaded, versioned, and so on.

The supported RPS relational target databases are Microsoft SQL Server 2008 and higher.

An RPS system consists of:

- An RPS mapping defined in the JADE production database, which maps classes and attributes of a JADE schema to relation database tables and columns.
- An RPS node (or multiple RPS nodes), which runs as a relational SDS database and executes the **Datapump** application to pump the JADE data to the Relational Database Management System (RDBMS) target.
- An RDBMS target database for each RPS node, which contains the tables and columns defined in the RPS mapping.
- The RPS Manager utility, which enables you to create the RDBMS tables and columns from the RPS mapping, load the data instances from the JADE production database, and control the execution of the **Datapump** application.

For details about mapping your JADE database to a relational database, see Chapter 15, "Using the Relational Population Service (RPS) to Replicate a JADE Database", in the *JADE Development Environment User's Guide*.

For considerations when using RPS, see Chapter 2, "Relational Population Service (RPS) Support", in the *JADE Synchronized Database Service (SDS) Administration Guide*.

Interfacing to OLE 2.0

JADE enables you to interface to Object Linking and Editing (OLE 2.0) object images.

Data stored for a persistent **OleObject** object is compressed automatically, which significantly reduces the required storage overhead when large numbers of **OleObject** objects are stored. Transient **OleObject** objects are created for runtime use of **OleControl** objects. These transient objects are not compressed, as that would create an unnecessary overhead for that runtime object.

JADE provides the **OleObject** system class, which is used to store the Object Linking and Editing (OLE 2.0) object images for the **OleControl** class. The **OleObject** class can also be used to store programmatically controlled OLE images that are not attached to a control. (For more details, see "OleObject Class", in Chapter 1 of the *JADE Encyclopaedia of Classes*.)

The **OleControl** system class provided by JADE allows attachment of objects that are actually edited or controlled by other applications. For example, a Word document, an Excel spread sheet, a video clip, or a Paintbrush picture can be stored in JADE, but editing or playing the object invokes the controlling application. (For more details, see "OleControl Class", in Chapter 2 of the *JADE Encyclopaedia of Classes*.)

The **OleControl** class handles any object for which an OLE Server is registered in the database operating system registry. An application that is capable of being an OLE Server has a ***.reg** file associated with it. To load the registration information, double-click on the **.reg** file in the Windows Explorer.

Notes Neither the **OleObject** nor the **OleControl** class is available in a JADE application running on a server node.

The JADE extract process extracts a definition of the **OleControl** object (as it does with other controls painted at design time when the form definitions are extracted) but it does not extract the user data associated with the control. It is your responsibility to extract the user data and load it into a deployed database, if required.

For example, the following objects can be handled by the **OleControl** class.

- Word for Windows
- Excel
- Media Player (for sound and videos)
- Sound Recorder
- Paintbrush
- Word Art 2
- Equation 2
- MS Graph

You can use OLE 2.0 and ActiveX controls in JADE only on the currently supported Windows operating systems. (For details, see "Software Requirements" under "Operational Requirements", in the *JADE Installation and Configuration Guide*.)

Embedding and Linking Objects

The types of attachment provided by the **OleControl** class are listed in the following table.

Attachment	Description
Embedding	A new object is constructed and the data is copied into the JADE database.
	An existing file is used to build the object and that object is then copied into the JADE database. In most cases, the original file is then no longer referenced or updated. The file type must be recognized as belonging to a defined OLE Server.
Linking	A link to a file only is stored in the JADE database.

The data that is stored in the JADE database for an embedded object is determined by the application server that is being used. In most cases, the entire embedded object is stored. For Media Player objects, only the initial image and a link to the rest of the data on the CD-ROM are stored.

What the OleControl Shows

When a loaded object is not active in the control, an image similar to a picture box is displayed; for example:

- The first frame of the inserted video
- The image of a bitmap
- The first page of a Word for Windows document
- An image of an Excel spread sheet

The OLE control provides scroll bars if the image is bigger than the control, to enable the whole image to be viewed. This image is defined by the OLE Server, which is just an icon in some cases.

Creating an OLE Control

The JADE Painter handles the OLE control as it does for any other control, although it is displayed like an empty picture box, for example.

An object can be inserted or edited in the control in Painter, or programmatically at run time.

In Painter, to initiate a dialog form that displays a folder with the options listed in the following table, click on the **OleControl** palette button or the **oleObject** property in the **Specific** sheet of the Properties dialog.

Option	Action
Create New	Initiates the application and creates a new object
Embed from file	Builds an object based on a copy of the file
Link from File	Builds an object based on the file that must be available for subsequent access

Activating the OLE Application at Run Time

An object that is present in the control can be activated at run time, usually by the user double-clicking the image. When an object is activated, the application controlling the object is initiated to edit or play the object.

The activation occurs in the following ways, depending on its capabilities and on controllable options.

- In place

  If the application is in-place capable and the control is large enough, it runs in the control itself. The menu bar of the form may be replaced by the initiated menu of the application. The toolbar of the application may also be inserted into the control. For example, Word for Windows 6 replaces the menu bar of the form with its own menu bar (but many options are disabled). It also inserts the Word for Windows toolbar at the top of the control. The user may then edit the document that is displayed within the control.

  When the user clicks on any other control on the form other than the displayed OLE object, the OLE application is deactivated. The menu bar of the form is restored, the inserted toolbar is hidden, and the image is replaced by the static object version. Any changes made to the object are stored in the **OleObject** of the control.

  As two applications are running simultaneously, the deactivation process is very complex and can result in the loss of mouse messages after the initial **mouseDown** event. When clicking to deactivate, the object being clicked may not respond (except to cause the **deactivate** event).

- Open activation

  If the application does not support in-place activation, the application is opened in normal mode as though the application had been initiated from the program folder or shortcut. The control image is drawn through with diagonal lines, to show that the server is active.

  The application's menus may include different options applicable to OLE use. These options can include:

  - **update OleControl** (save changes into JADE now)
  - **exit & Return to OleControl** (shut down and save changes)

  When shutting down an application, you are usually asked whether you want to update the OLE control. This is not universal, however, and in some cases, changes that were made are discarded when closing using the Control-Menu unless the **exit & Return to OleControl** option is selected.

For more details, see "OleControl Class", in Chapter 2 of the *JADE Encyclopaedia of Classes*.

Connecting to Network Devices Using TCP/IP

You can connect to external (JADE or non-JADE) systems from JADE by using Transmission Control Protocol / Internet Protocol (TCP/IP) as the transport provider.

By specifying the host connection details to TCP/IP, the transfer of information between JADE and an external system is achieved simply by applying methods that are implemented by the **TcpIpConnection** class.

The **TcpIpConnection** class also provides the **JadeInternetTCPIPConnection** subclass, which implements the interface defined by the **TcpIpConnection** class specifically for the Internet TCP/IP API. The methods defined in the **JadeInternetTCPIPConnection** class are listed in the following table.

Method	Description
openPipeCallback	Initiates an asynchronous read of the opened TCP/IP connection
readBinary	Reads binary data from the Internet connection and returns when the specified number of bytes has been read or when a block of data is received
readDataWithLength	Reads data from the Internet TCP/IP connection and returns when the specified length of data is read
readPipeCallback	Performs Web session evaluation processing

Method	Description
sendReply	Sends the formatted HyperText Markup Language (HTML) page to the opened Internet TCP/IP connection
writeBinary	Writes binary data to the Internet TCP/IP connection and returns when the operation is complete

External Software Requirements

To connect to network devices using TCP/IP, you must have the TCP/IP protocol installed from your host operating system installation medium. For more details, see "Software Requirements", in Chapter 1 of the *JADE Installation and Configuration Guide*.

For details about configuring and maintaining your TCP/IP communications protocol, see "Configuring Your Network Protocol", in Chapter 2 of the *JADE Installation and Configuration Guide*.

Using the TcpIpConnection Class

Within JADE, you establish a connection and send or receive data programmatically. The **Connection** class defines all of the methods that you require for external communications. The **TcpIpConnection** class is a subclass of the **Connection** class and is specifically for external communication through TCP/IP.

Note As you can create a **TcpIpConnection** object as a shared transient object, you can pass it to another JADE process on the same JADE node, if required.

When using the **Connection** class, note the following restrictions.

- The **Connection** class hides underlying implementation and transport-dependent details.

  Any exception conditions result in an exception of type **ConnectionException** being raised.

- Connection objects must be transient.

  If an attempt is made to create a persistent **Connection** object, an exception is raised.

- Asynchronous methods have a receiver object and a message (method name) specified as parameters.

  When the method completes, the specified (callback) method of the specified object is called. The callback method is assumed to match the signature required by the calling asynchronous method.

Properties Provided by the TcpIpConnection Class

The properties defined in the **TcpIpConnection** class are summarized in the following table.

Property	Description
authenticationLibrary	Contains the name of the library that contains the authentication method
cryptLibrary	Contains the name of the library that contains the encryption and decryption methods
decryptMethod	Contains the name of the decryption method in the encryption and decryption library
encryptMethod	Contains the name of the encryption method in the encryption and decryption library

Property	Description
genAuthChallengeMethod	Contains the name of the method used to generate the authentication challenge
genAuthResponseMethod	Contains the name of the method used to generate the response from the authentication challenge
localIpAddress	Contains the local IP address
localInterface	Contains the interface name or IP address of a local network interface
localPort	Contains the local service port
networkProxy	Contains a reference to the object identifier (oid) of the proxy in the **TcpIpConnection** class object
port	Contains the target service port or listen port
remoteIpAddress	Contains the remote host IP address
remoteName	Contains the remote host name
remotePort	Contains the port number used on the remote node
resolveRemoteName	Specifies whether the remote host name is to be located
verifyAuthResponseMethod	Contains the name of the method used to verify the response to the authentication challenge

Properties Provided by the Connection Class

The following table summarizes the properties defined in the **Connection** class and inherited by the **TcpIpConnection** class.

Property	Function
fillReadBuffer	If set to **true**, the **readBinary** and **readBinaryAsynch** methods do not return or notify you until the requested length of data has been received. If set to **false**, these methods return or notify you as soon as any data is received.
name	Contains a generic name that specifies to what the connection object is to connect.
state	A read-only property that returns the state of the connection. The states are disconnected (**0**), connecting (**1**), connected (**2**), and disconnecting (**3**).
timeout	Contains the number of milliseconds after which a synchronous or asynchronous listen (including continuous), read, or write operation times out.

For details about the properties defined in the **Connection** class and inherited by the **TcpIpConnection** class, see Chapter 1 of the *JADE Encyclopaedia of Classes*.

Methods Provided by the TcpIpConnection Class

The following table summarizes the methods defined in the **Connection** class and inherited by the **TcpIpConnection** class.

Method	Description
close	Closes a connection to a remote application and returns when the connection is closed

Method	Description
closeAsynch	Closes a connection to a remote application and returns immediately
getMaxMessageSize	Returns the maximum message size that can be sent or received at one time
listen	Waits for a remote application to connect to its port and returns when the connection is established
listenAsynch	Waits for a remote application to connect to its port and returns immediately
listenContinuous	Waits for a remote application to connect to its port and returns the new connection on a new instance of the **TcpIpConnection** class while the original instance is still available for listening on subsequent calls
listenContinuousAsynch	Waits for remote applications to connect to its port and returns immediately
open	Establishes a connection to a remote application and returns when the connection is established
openAsynch	Establishes a connection to a remote application and returns immediately
readBinary	Reads binary data from the connection and returns when the specified number of bytes has been read or when a block of data is received
readBinaryAsynch	Reads binary data from the connection and returns immediately
readUntil	Reads data from the connection and returns when the specified delimiter is found in the data stream
readUntilAsynch	Reads data from the connection until the specified delimiter is found in the data stream and returns immediately
writeBinary	Writes binary data to the connection and returns when the operation is complete
writeBinaryAsynch	Writes binary data to the connection and returns immediately

For details about the methods defined in the **Connection** class and inherited by the **TcpIpConnection** class, see Chapter 1 of the *JADE Encyclopaedia of Classes*.

Connection Authentication

Connection authentication is initiated on the listener end of the connection by generating an authentication challenge and sending it to the connection initiator. The connection initiator generates an authentication response from the received challenge, and returns this to the listener to verify. If the verification is not successful, the listener closes the connection.

Authentication data is encrypted or decrypted if data encryption has been set up.

You must specify the name of the user-supplied authentication library in the **authenticationLibrary** property of the **TcpIpConnection** class before an **open**, **openAsynch**, **listen**, or **listenAsynch** method operation.

You must specify the name of the challenge generation method in the **genAuthChallengeMethod** property. The challenge generation method must have the following C++ signature.

```
int JOMAPI generateChallenge (BYTE **ppChallenge,
                              Size *pChallengeSize);
```

You must specify the name of the response generation method in the **genAuthResponseMethod** property. The response generation method must have the following C++ signature.

```
int JOMAPI generateResponse (BYTE *pChallenge,
                             Size challengeSize,
```

```
                                    BYTE **ppResponse,
                                    Size *pResponseSize);
```

You must specify the name of the response verification method in the **verifyAuthResponseMethod** property. The response verification method must have the following C++ signature.

```
int JOMAPI verifyResponse (BYTE *pChallenge,
                           Size challengeSize,
                           BYTE *pResponse,
                           Size responseSize);
```

The authentication methods return one of the values listed in the following table.

Value	Description
0	The response was successful
Non-zero	The response failed

pChallenge or ppChallenge

Use the **pChallenge** or **ppChallenge** input parameter for a randomly generated challenge or the challenge received from the server-side of the connection.

challengeSize or pChallengeSize

Use the **challengeSize** or **pChallengeSize** input parameter for the size of the challenge received from the server-side of the connection.

The maximum challenge size is 1,024 bytes.

pResponse or ppResponse

Use the **pResponse** or **ppResponse** output parameter for the response generated from the challenge or returned from the client side of the connection.

responseSize or pResponseSize

Use the **responseSize** or **pResponseSize** output parameter for the size of the response received from the server-side of the connection.

The maximum response size is 1,024 bytes.

Data Encryption

Data is encrypted before any physical data is written and it is decrypted after any physical data has been read.

You must specify the name of the user-supplied encryption library in the **cryptLibrary** property before an **open**, **openAsynch**, **listen**, or **listenAsynch** method operation.

You must specify the name of the encryption method in the **encryptMethod** property. The encryption method must have the following C++ signature.

```
int JOMAPI encrypt (BYTE *pDataIn,
                    Size dataInLength,
                    BYTE **ppDataOut,
                    Size *pDataOutLength);
```

You must specify the name of the decryption method in the **decryptMethod** property. The decryption method must have the following C++ signature.

```
int JOMAPI decrypt (BYTE *pDataIn,
                    Size dataInLength,
                    BYTE **ppDataOut,
                    Size *pDataOutLength);
```

Caution If character data is passed in or out, your routines must be aware of the ANSI or Unicode character size (that is, 1 byte for ANSI and 2 bytes for Unicode).

The encryption and decryption methods return the following values.

Value	Description
0	The encryption or decryption routine was successful
Non-zero	The encryption or decryption routine failed

pDataIn

Use the **pDataIn** input parameter to specify the input data stream for encryption or decryption.

dataInLength

Use the **dataInLength** input parameter to specify the length of the input data stream.

ppDataOut

Use the **ppDataOut** output parameter to specify the location of the output data stream after encryption or decryption.

pDataOutLength

Use the **pDataOutLength** output parameter to specify the length of the encrypted or decrypted data stream.

TCP/IP Proxy Servers

The **Object** class provides the transient **JadeTcpIpProxy** subclass, which implements TCP/IP network proxy support that enables you to open a TCP/IP network connection through a proxy server.

If you cannot establish a direct TCP/IP connection because of physical network layouts or restrictions (for example, the use of a firewall), you may have to establish a connection through a proxy server by using the functionality provided by this class.

Proxies can be used as part of a firewall solution, as they sit between the client application and the server application and do not permit the client to connect directly to the server. The client is required to connect to the proxy, and asks the proxy to connect to the server on the client's behalf. The proxy may also require authentication from the client before it allows the connection to the server. There are a number of different types of proxies, the two major types being HyperText Transfer Protocol (HTTP) and SOCKS. From the perspective of the client, the difference between the types of proxy is the protocol (that is, the type and format of messages) used between the client and the proxy. The other issue for the client is determining the type of proxy and where it is running.

The **TcpIpConnection** class **networkProxy** property contains a reference to the **JadeTcpIpProxy** class. If this reference contains a non-null value, the **JadeTcpIpProxy** class **connect** method is executed, which in turn calls the **TcpIpConnection** class **open** or **openAsynch** method for each attempt to connect to the proxy. If the value of the **networkProxy** property value is **null**, the **TcpIpConnection** class **open** or **openAsynch** method is executed.

For details, including code examples and issues to consider when reimplementing **JadeTcpIpProxy** class functionality, see Chapter 1 of your *JADE Encyclopaedia of Classes*. See also "Firewall for the JADE Internet Environment", in Chapter 4 of the *JADE Installation and Configuration Guide*.

Constants Provided by the JadeTcpIpProxy Class

The constants provided by the **JadeTcpIpProxy** class are listed in the following table.

Constant	Integer Value	Constant	Integer Value
BrowserType_Auto	3	ProxyType_Auto	0
BrowserType_InternetExplorer	1	ProxyType_Direct	5
BrowserType_Netscape	2	ProxyType_Http	1
BrowserType_None	0	ProxyType_Https	4
ProxyType_Socks4	2		

Properties Provided by the JadeTcpIpProxy Class

The properties defined in the **JadeTcpIpProxy** class are summarized in the following table.

Property	Contains the …
browserType	Browser type whose proxy host configuration settings are used
domain	Domain name to log in to the host
host	Name or IP address of the host
password	Password that is to complete the log in to the host
port	Port number used to connect to the host
proxyType	Communications protocol used to connect to the proxy host
userName	User name that logs in to the host

Method Provided by the JadeTcpIpProxy Class

The method defined in the **JadeTcpIpProxy** class is summarized in the following table.

Method	Description
connect	Establishes a connection to the target host through the specified network proxy

Multiple Worker TCP/IP Connections

Multiple worker TCP/IP connections provide an interface for sharing the messages arriving on client sockets among a pool of worker server JADE applications, using a single listen TCP/IP socket. This enables server-style applications to share a workload generated by a large number of clients among a small number of server processes.

No master or manager application is required, and all worker processes share exactly the same initialization and message processing logic.

Note All JADE worker processes must reside in the same JADE node as the TCP listen socket.

A client connection can be bound to a single worker process. Events for a bound connection are directed to that bound worker. Events for unbound connections are handled by any worker. JadeMultiWorkerTcpConnection

For details, see the **JadeMultiWorkerTcpTransport** and **JadeMultiWorkerTcpConnection** classes in Volume 1 of the *JADE Encyclopaedia of Classes*.

JadeMultiWorkerTcpConnection Class

The **JadeMultiWorkerTcpConnection** class provides an interface for sharing the messages arriving on client sockets among a pool of worker server JADE applications.

JadeMultiWorkerTcpConnection Class Constants

The constants provided by the **JadeMultiWorkerTcpConnection** class are listed in the following table.

Constant	Integer Value	Description
open	3	Connection is open

JadeMultiWorkerTcpConnection Properties

The properties defined in the **JadeMultiWorkerTcpConnection** class are summarized in the following table.

Property	Description
connectionId	Contains the unique number assigned to the client connection when it opens
fillReadBuffer	Specifies whether the read buffer is filled
idleTimeout	Contains the maximum number seconds to wait for idle connections to have no input before causing a **WorkerIdleTimeout** management connection event
keepAssigned	Specifies whether the connection is kept assigned after exiting from the callback method
localAddress	Contains the address of the local connection
localPortnumber	Contains the port number of the local connection
remoteAddress	Contains the address of the remote connection
remotePortnumber	Contains the port number of the remote connection
state	Contains the current state of the connection
timeout	Contains the number of seconds to wait for a **readBinary** or **writeBinary** method call to complete
userObject	Contains connection-related information between event callbacks
userState	Contains the state value between event callbacks

JadeMultiWorkerTcpConnection Methods

The methods defined in the **JadeMultiWorkerTcpConnection** class are summarized in the following table.

Method	Description
bindToAssignedWorker	Binds this connection to the currently assigned worker (that is, to the current JADE process)
bindToWorkerId	Attempts to bind the connection to the worker process with the specified worker identifier
causeUserEvent	Queues a user event for the connection with the specified tag value
close	Requests that the connection be closed locally
getAssignedWorkerId	Returns the worker to which the connection is assigned
getBoundWorkerId	Returns the worker identifier (the current or another worker process) to which the connection is bound
getFullName	Returns a string containing the full name of the associated **JadeMultiWorkerTcpTransport** object
getGroupId	Returns the identifier of the transport group to which the connection belongs
getLocalHostname	Returns a string containing the generic host name associated with the **localAddress** property
getRemoteHostname	Returns a string containing the generic host name associated with the **remoteAddress** property
readBinary	Reads binary data from the connection and returns when the specified number of bytes has been read or when a block of data is received
readUntil	Reads data from the connection and returns when the specified delimiter is found in the data stream
unbind	Unbinds the connection if it is currently bound
writeBinary	Writes binary data to the connection and returns when the operation is complete
writeBinaryAndFile	Writes binary data and the file to the connection, writes the specified section of the file, and then returns when the operation is complete

JadeMultiWorkerTcpTransport Class

The **JadeMultiWorkerTcpTransport** class encapsulates behavior required for multiple user TCP/IP connections between JADE systems.

JadeMultiWorkerTcpTransport Class Constants

The constants provided by the **JadeMultiWorkerTcpTransport** class are listed in the following table.

Constant	Integer Value	Occurs…
Notify_Continuous	0	Continuously until disabled by a call to the **cancelNotify** method
Notify_OneShot	1	Once only

JadeMultiWorkerTcpTransport Properties

The properties defined in the **JadeMultiWorkerTcpTransport** class are summarized in the following table.

Property	Contains the …
listenHostname	Host name of the listening connection
listenPortnumber	Port number of the listening connection
listenState	State of the listen connection
queueDepthLimit	Connection ready queue size required to trigger the **QueueDepthExceeded** management event
queueDepthLimitTimeout	Number of seconds that the connection ready queue size must remain greater than the value of the **queueDepthLimit** property before triggering the **QueueDepthExceeded** management event
statisticsLogInterval	Number of seconds at which worker and group to-date statistics are written to the **jommsg.log** file
userGroupObject	Reference to an object that you can associate with the transport group
workerId	Unique sequential number assigned on the first **beginListening** method call on this **JadeMultiWorkerTcpTransport** object
workerIdleTimeout	Number of seconds after the last event handled by this process that the **WorkerIdleTimeout** management event is queued for this process

JadeMultiWorkerTcpTransport Methods

The methods defined in the **JadeMultiWorkerTcpTransport** class are summarized in the following table.

Method	Description
beginListening	Attaches to the transport group and creates a new group if the specified group does not exist
cancelNotify	Stops callbacks being queued to this process
causeUserEventOnConnId	Queues a user event for the specified connection
countAssignedConnections	Returns the number of connections currently assigned to this worker
countBoundConnections	Returns the number of connections currently bound to this worker
countIdleWorkers	Returns the number of idle worker processes sharing the pool
countQueuedConnections	Returns the number of connections that are unassigned and have queued events
countWorkers	Returns the number of worker processes sharing the pool
getFullName	Returns the full name of this transport group (that is, the values of the **listenHostname** and **listenPortnumber** properties, separated by a colon character)
getGroupId	Returns the unique number assigned to each transport group when it is created by the first worker to call the **beginListening** method with a new full name
getGroupStatistics	Populates the specified **JadeDynamicObject** with properties containing the to-date values of statistics captured by this transport group

Method	Description
getMyStatistics	Populates the specified **JadeDynamicObject** with properties containing the to-date values of statistics captured by the receiving worker
notifyEventsAsync	Notifies the implementing **JadeMultiWorkerTcpTransportIF** receiver of asynchronous events
stopListening	Closes the listen connection to prevent further client connections
validateServerProcess	Validates the server process

JadeMultiWorkerTcpTransportIF Interface

The **JadeMultiWorkerTcpTransportIF** interface provides TCP/IP multiple worker connection event callback methods.

You can view the **JadeMultiWorkerTcpTransportIF** interface and its constants and methods only in the Interface Browser of a user schema that has an implementation mapping to this **RootSchema** interface. (For details about implementing the **JadeMultiWorkerTcpTransportIF** interface for a class selected in the Class Browser of a user schema, see "Implementing an Interface", in Chapter 14, "Adding and Maintaining Interfaces", of the *JADE Development Environment User's Guide*.)

JadeMultiWorkerTcpTransportIF Interface Constants

The constants provided by the **JadeMultiWorkerTcpTransportIF** interface are listed in the following table.

Constant	Integer Value
ConnEvType_IdleTimeout	1
MgmntEvType_QueueDepthExceeded	1
MgmntEvType_WorkerIdleTimeout	2

JadeMultiWorkerTcpTransportIF Interface Callback Method Signatures

The signatures of callback methods provided by the **JadeMultiWorkerTcpTransportIF** interface are summarized in the following table.

Method	Callback method for the …
closedEvent	**ClosedEvent** connection event
connectionEvent	**ConnectionEvent** connection event
managementEvent	**ManagementEvent** transport event
openedEvent	**OpenedEvent** connection event
readReadyEvent	**ReadReadyEvent** connection event
userEvent	**UserEvent** connection event

Connecting to Network Devices Using a Secure Sockets Layer (SSL)

You can connect to external systems from JADE by using an SSL library protocol instead of the TCP/IP protocol when the **TcpIpConnection** class **sslContext** property contains a reference to a **JadeSSLContext** transient object.

JadeSSLContext connections use digital certificates in X509 format, which must exist on disk in Privacy-Enhanced Electronic Mail (PEM)-encoded certificate (PEM) format.

The transient **JadeX509Certificate** class stores the digital certificates in X509 format.

SSL is a secure communication protocol on top of an already established TCP/IP connection.

SSL libraries are generated from publicly available third-party sources, maintained by the OpenSSL Group (http://www.openssl.org). Jade supports TLS version 1, TLS version 1.1, and TLS version 1.2.

For details about the **JadeSSLContext** and **JadeX509Certificate** classes and the properties and methods defined in these classes, see Chapter 1 of the *JADE Encyclopaedia of Classes*.

Connecting to Network Devices Using a Named Pipe

You can connect to external systems from JADE by using the Windows NamedPipe feature, which is available on all supported Windows operating systems, to establish a two-way communication channel between a JADE process and another JADE or non-JADE process.

One process must offer the server end of the NamedPipe channel and another process can then connect to the client end of the channel. After the connection is made and while it remains valid, both sides of the pipe have equal status (that is, the terms *server* and *client* do not apply).

Multiple instances of the pipe can be opened, by running multiple copies of the JADE application from the same **jade.exe** executable program, where each application opens the same pipe name. For details, see "NamedPipe Class", in Chapter 1 of the *JADE Encyclopaedia of Classes*.

Using the NamedPipe Connection Class

Within JADE, you establish a connection and send or receive data programmatically. The **Connection** class defines all of the methods that you require for external communications. The **NamedPipe** class is a subclass of the **Connection** class and is specifically for external communication through the Windows NamedPipe facility. When using the **Connection** class, note the following restrictions.

- The **Connection** class hides underlying implementation and transport-dependent details. Any exception conditions result in an exception of type **ConnectionException** being raised.
- Connection objects must be transient. If an attempt is made to create a persistent **Connection** object, an exception is raised.
- Asynchronous methods have a receiver object and a message (method name) specified as parameters. When the method completes, the specified (callback) method of the specified object is called. The callback method is assumed to match the signature required by the calling asynchronous method.

The **NamedPipe** class supports both synchronous and asynchronous operations, as follows.

- As the functionality of the **timeout** property in the **Connection** class is not supported in the **NamedPipe** class, synchronous methods that time out will wait forever for completion.

- Asynchronous methods have a receiver object and a message (method name) specified as parameters. When the method (I/O) completes, the specified (callback) method of the object is called. The callback method must match the signature required by the calling asynchronous method.

Only one synchronous or asynchronous read operation can be in effect at each end of each instance of the pipe. Multiple asynchronous write operations can be in effect.

Opening the server end of the pipe waits until the other end of the pipe is connected. Opening the client end of the pipe fails immediately if the server end of the pipe has not been offered.

NamedPipe Class Property

The property defined in the **NamedPipe** class is summarized in the following table.

Property	Description
serverName	Specifies the name of the server workstation

NamedPipe Class Methods

The methods defined in the **NamedPipe** class are summarized in the following table.

Method	Description
close	Closes a connection to a remote application
closeAsynch	Closes a connection to a remote application and returns immediately
getMaxMessageSize	Gets the maximum message size that can be sent or received at one time
listen	Offers a connection to a remote application and returns when established
listenAsynch	Offers a connection to a remote application and returns immediately
open	Attempts to open the client end of a named pipe connection
openAsynch	Attempts to open a connection to a named pipe and returns immediately
readBinary	Reads binary data from the connection and returns when the data has been read
readBinaryAsynch	Initiates a read of binary data from the connection and returns immediately
writeBinary	Writes binary data to the connection and returns when the operation is complete
writeBinaryAsynch	Initiates a write of binary data to the connection and returns immediately

As the **listenContinuous** and **listenContinuousAsynch** methods inherited from the **Connection** superclass are not supported for the **NamedPipe** class, an exception is raised if you attempt to use these methods.

InternetPipe Subclass

The **InternetPipe** class, a subclass of the **NamedPipe** class, provides an interface for communicating with JADE applications from the Internet through an Internet server.

Note This class is available only under a Windows operating system that supports services. To access your JADE applications from the Internet, the JADE server node and the workstation running the JADE application must be running a Windows operating system that supports services.

To communicate with the **jadehttp** file on the Internet server using the pipe channel, the JADE application creates a transient instance of the **InternetPipe** class and then offers the named pipe for opening with the name of the JADE application. When the pipe is connected, it waits for Internet requests to be sent over the pipe. If no named pipe is open, the Internet user is advised that the service is not available. Multiple instances of the pipe can be opened, by running multiple copies of the JADE application from the same **jade.exe** executable program, where each application opens the same pipe name.

InternetPipe Class Methods

The methods defined in the **InternetPipe** class are summarized in the following table.

Method	Description
openPipeCallback	Initiates an asynchronous read of the opened pipe
readPipeCallback	Performs Web session evaluation processing
sendReply	Sends the formatted HyperText Markup Language (HTML) page to the opened pipe

For details, see "InternetPipe Class", in Chapter 1 of the *JADE Encyclopaedia of Classes*.

Interfacing to the Internet

You can use the World Wide Web (Internet) to access your JADE applications, using the HTML thin client mode. Web browsers, such as Netscape Navigator or Microsoft Explorer, provide a convenient client interface to JADE data on distributed servers.

Notes You cannot run Unicode Web applications.

Both the JADE server node workstation and the workstation that is running the JADE application must be running under a Windows operating system that supports services when accessing your JADE applications from the Internet.

When using the HTML thin client mode to access a JADE application from the Internet, your JADE application and Microsoft Internet Information Server (IIS) 2.0 must reside on the same workstation.

The JADE HTML thin client interface to the Internet provides the following features.

- Automatic generation of the Web interface
- Session management
- Web browsers
- Web server
- JADE clients
- Security

For details, see "Implementing Web Applications", in Chapter 1 of the *JADE Web Application Guide*. See also "Firewall for the JADE Internet Environment", in Chapter 2 of the *JADE Installation and Configuration Guide*.

Chapter 3 Transforming an External Relational Database

This chapter covers the following topics.

Overview

JADE provides the External Schema Wizard, which enables you to map your relational database to a JADE object model. The External Schema Wizard, installed with your JADE software, automates the process of transforming a relational database model into a JADE object model and enables you to add and configure an external database schema for your schema.

Note Before you can create, maintain, or access an external database schema, you must first install an ODBC driver. For details, see "ODBC Requirements for External Database Coexistence", in Chapter 2.

You can map one or more external relational databases into a JADE object model. Use the External Schema Wizard to access the catalog information from selected external data sources and generate proxy classes. For an overview of external database coexistence, see "External Database Coexistence", in Chapter 2.

Defining Your External Database Schema

The External Databases Browser, accessed from the Schema Browser, enables you to select and operate an external database schema.

Use the External Databases Browser to create, maintain, or access an external database schema for the current JADE schema. You can configure more than one external database to coexist with JADE.

» To open an External Databases Browser

- Select the **External Databases** command from the Browse menu in the Schema Browser.

An External Databases Browser window is then opened. If you have not yet created an external database schema, nothing is displayed in the External Databases Browser.

Only one External Databases Browser for the current schema can be open at any time. However, you can have concurrent open External Databases Browsers for different schemas in a development session. You can change your default browser options, if required, by using the **Browser** sheet, accessed from the Options menu **Preferences** command.

Using the Databases Menu

The External Databases Browser provides the Databases menu, containing the commands listed in the following table, to enable you to define and maintain your external database schema.

Command	Action
Add	Displays the External Schema Wizard, to enable you to add and configure a new external database schema
Remove	Deletes the selected external database schema from the schema
Change	Displays the External Schema Wizard, to enable you to maintain the selected external database schema
View	Displays the External Schema Wizard in read-only mode
Print	Outputs a full description of the external database schema to the printer
Extract	Extracts the selected external database schema
Load	Loads an external database schema from a file

Adding an External Database Schema

Add an external database schema to the current JADE schema from the Databases menu in the External Databases Browser.

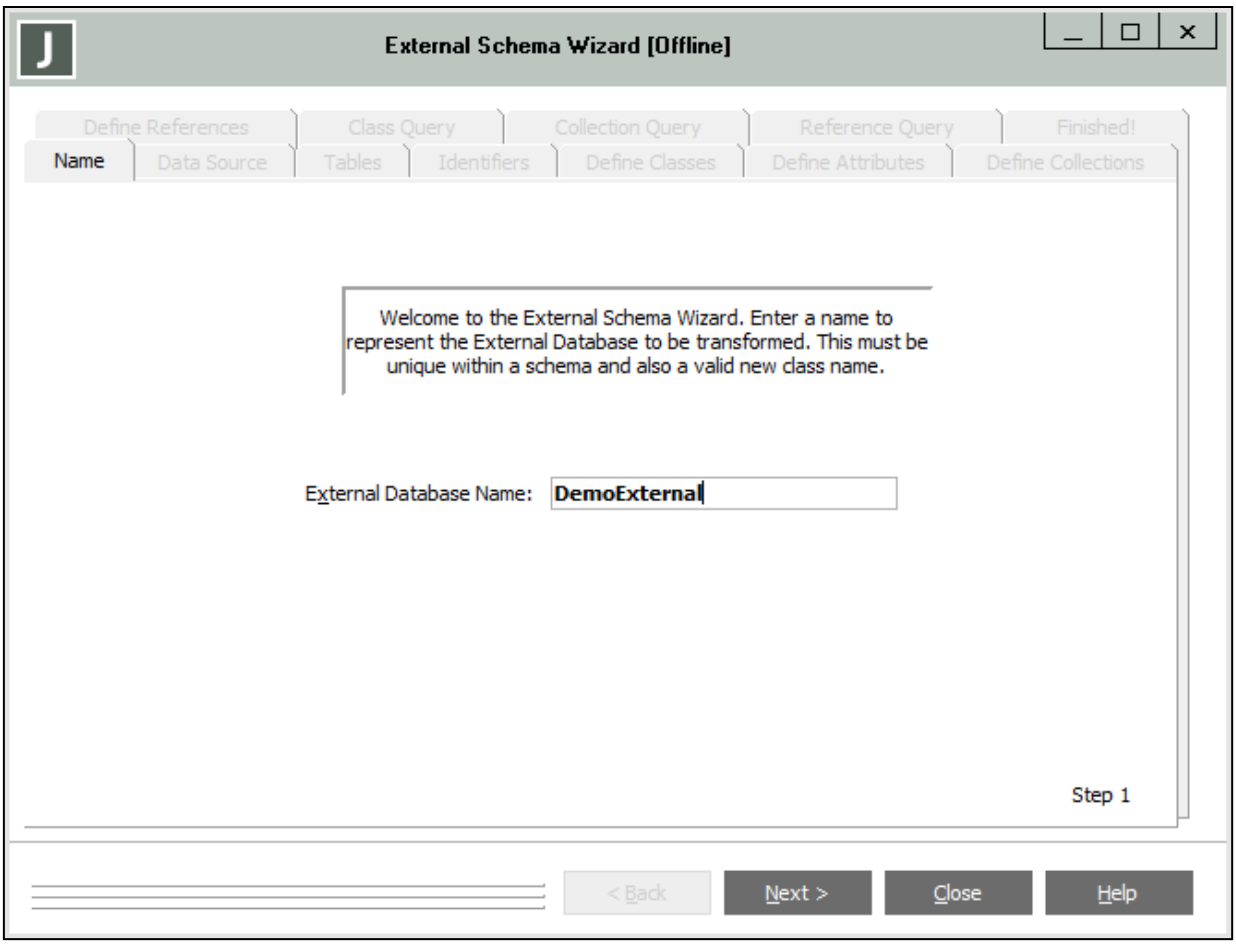
Notes When adding a new external database to a versioned schema in the development environment, add the external database in the *current version*. You can specify connection and subsequent database modifications in the latest version.

Use the **+** and **-** label in the upper-left column of a table header to include or exclude rows (that is, toggle the selection of rows).

To add an external database definition

- Select the **Add** command from the Databases menu.

The External Schema Wizard, shown in the following image of the first sheet, is then displayed.



Notes The External Schema Wizard is a wizard-style dialog that consists of 12 steps, each represented by a sheet of the dialog. Use the **Next >** and **< Back** buttons to navigate forwards or backwards through the steps, or explicitly to a step by tabbing the sheet tab. No step is enabled until the previous step has been completed.

As the changes are committed at each step, you can click the **Close** button on any sheet to close the wizard and resume your external database transformation later, if required.

Specifying a Name for Your External Database Schema

The **Name** sheet of the External Schema Wizard is displayed when the dialog is first opened, to enable you to define the name of your new external database schema.

To specify a name for your external database schema

1. In the **External Database Name** text box, specify the name of your new external database schema; for example, **ExtDb**. The name must be unique to the current JADE schema and must be a valid unique class name. JADE converts the first character to uppercase, if required.

   The name is used as a subclass of the JADE **ExternalDatabase** class. When you have defined the name for your external database schema and proceeded to the next step in the schema transformation, you cannot change this name.

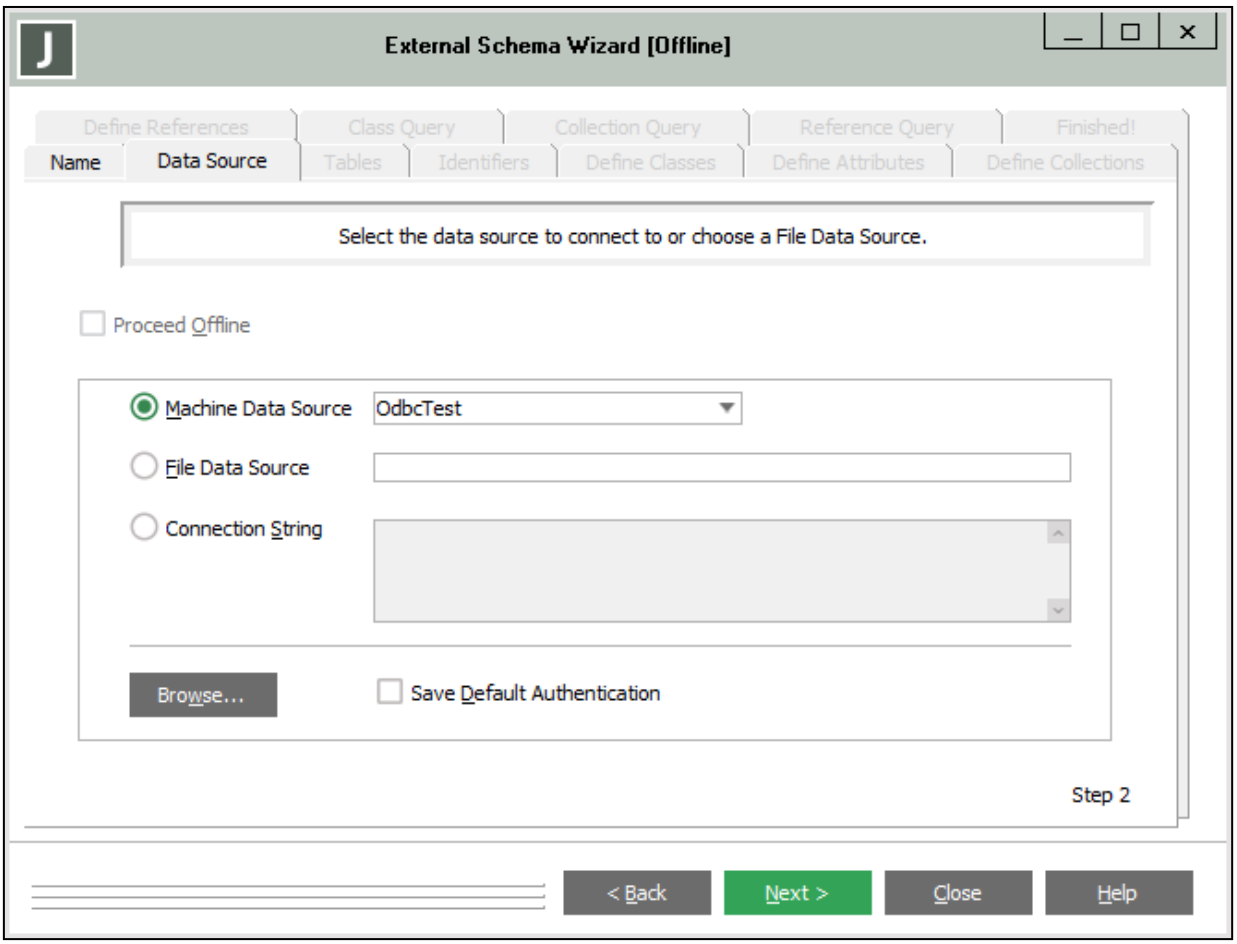
2. Click the **Next >** button when you have defined your external database schema name.

   Alternatively, click the **Close** button to abandon the schema transformation.

When you click the **Next >** button, the External Schema Wizard then enables you to establish the connection to the external database.

Establishing the Connection to the External Database

When you have specified the name to be associated with your external database schema, the **Data Source** sheet of the External Schema Wizard, shown in the following image, then enables you to specify the information required to establish the connection to the external database using ODBC.



Your data source connection can be a machine or a file data source.

To specify your data source

1. Check the **Proceed Offline** check box if you want to use the External Schema Wizard without connecting to the data source every time. This check box is enabled only after a successful connection to a data source has been established at least once, and the necessary catalog information has been read. If the External Schema Wizard is used offline, it cannot detect any changes that have occurred in the catalog information of the data source since the last connection was established.

   By default, this check box is unchecked; that is, the External Schema Wizard attempts to connect to the data source every time that it is used.

2. The **Machine Data Source** option button is selected by default if it is a new data source. If a data source has been specified previously, the last data source that was used is displayed. Select the required data source from the drop-down list box if you want to connect to a machine data source.

Alternatively, select the **File Data Source** option button and specify the data source name in the corresponding text box if you want to specify a file data source.

3. If you know the name of your data source, click the **Connection String** option button and then specify the source in the **Connection String** text box; for example:

```
DSN=TestDataSource;AutoStop=yes
```

Note For security reasons, your user code and password are not displayed.

4. If you do not know the data source, click the **Browse** button.

   The ODBC Driver Managers Data Source Browser is then displayed, to enable you to select an existing data source from the **Machine** sheet or the **File** sheet of the ODBC Driver Managers dialog. When you have selected the required data source and clicked the **OK** button, the selected data source is then displayed in the **Connection String** text box (without the user code and password).

   Caution If you later change the data source to access a database with a different schema, this marks all existing class definitions as obsolete and causes their deletion.

5. Check the **Save Default Authentication** check box if you want to store the authentication part of the connection string; that is, the user code and password. When this check box is unchecked (the default), no administrative-level password is stored in an unsecured system and any existing authentication that is stored is deleted. If no authentication is stored, the ODBC driver prompts you for it.

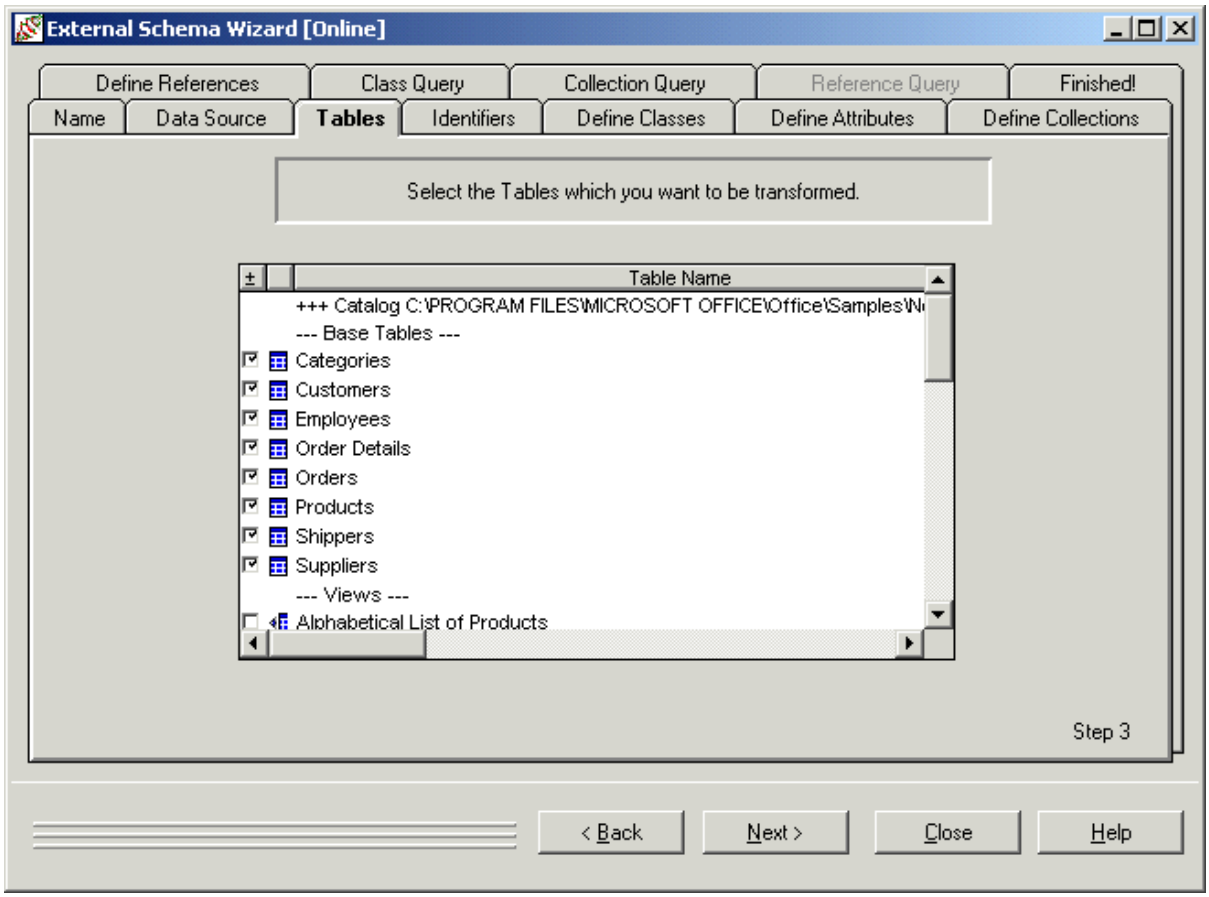
6. Click the **Next >** button when you have specified your data source. Alternatively, click the **Close** button to close the wizard.

When you click the **Next >** button, the External Schema Wizard then enables you to select the tables for exclusion from your external database schema.

For details about the JADE methods that enable you to set or return the external database machine or file data source, see the ExternalDatabase Methods, in Chapter 1 of the *JADE Encyclopaedia of Classes*.

Selecting Tables for Exclusion from the Schema

When you have established the connection to the external database, the **Tables** sheet of the External Schema Wizard, shown in the following image, then enables you to select the tables that are to be excluded from the external database schema transformation.



By default, all base tables and views are included. You can select a table for exclusion at any stage of the schema transformation, as long as it is not being referenced.

Tables are grouped by table type into base tables, views, and system tables. Bitmaps indicate each of the table types, and they are used throughout the External Schema Wizard. If a remark or description exists for a table or view in the external relational database, it is displayed at the right of the table or view name.

» To select tables for exclusion from your external database schema

1. In the **Table Name** list box, click the check box at the left of each table or view that you want to exclude.

   The check box is then unchecked and that table is no longer included in your external database schema.

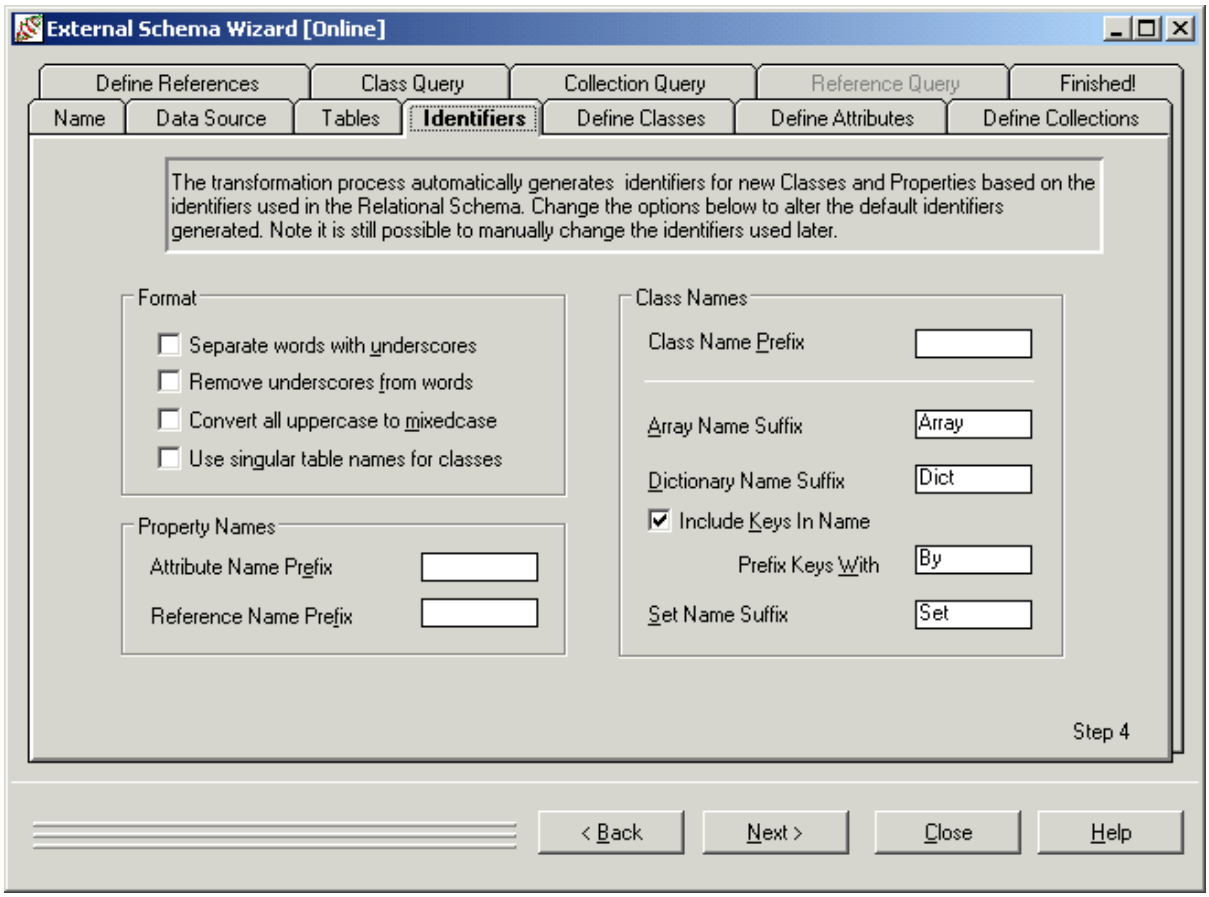
   When you access this sheet again, only those tables previously selected are checked. If one or more selected tables are subsequently unchecked, existing class definitions using that table are then marked as obsolete and are deleted.

2. Click the **Next >** button when you have selected the tables or views to exclude from your external database schema. Alternatively, click the **Close** button to close the wizard.

When you click the **Next >** button, the External Schema Wizard then enables you to select identifiers for class and property names.

Specifying Class and Property Name Identifiers

When you have selected the tables that you want excluded from your external database schema, the **Identifiers** sheet of the External Schema Wizard, shown in the following image, then enables you to specify the identifiers that are automatically generated for new classes and properties, based on the identifiers used in the external relational database.



You can manually change these identifiers later in the schema transformation process, if required.

To specify class and property name identifiers for your external database schema

1. Check the **Separate words with underscores** check box in the Format group box if you want to insert an underscore character between words when constructing an identifier from several words.

   By default, underscore characters are not inserted; that is, this check box is unchecked. When you select this check box and an underscore character already exists in the identifier, no underscore character is inserted.

2. Check the **Remove underscores from words** check box in the Format group box if you want to remove underscore characters from identifiers in the relational database schema and change the letter following the removed underscore character to uppercase.

   By default, underscore characters are not removed; that is, this check box is unchecked.

3. Check the **Convert all uppercase to mixedcase** check box in the Format group box if you want to convert identifiers in the relational database schema that are all uppercase characters to a mixed-case representation. By default, no uppercase character identifiers are converted to a mixed case representation; that is, this check box is unchecked.

   Tip As JADE enforces property names to start with a lowercase character, check this box when a relational database uses only uppercase characters for identifiers.

4. Check the **Use singular table names for classes** check box in the Format group box if you want to convert table names in the plural form in the relational database schema to the singular form before using them to construct class names. By default, the plural forms of table names are not converted to the singular form; that is, this check box is unchecked.

5. In the **Attribute Name Prefix** text box in the Property Name group box, specify an optional prefix for all attribute properties, if required. By default, no prefix is added to attribute names. The attribute prefix must start with a lowercase alpha character.

6. In the **Reference Name Prefix** text box in the Property Name group box, specify an optional prefix for all reference properties, if required. By default, no prefix is added to reference names. The reference prefix must start with a lowercase alpha character.

7. In the **Class Name Prefix** text box in the Class Names group box, specify an optional prefix for all class names including collection class names, if required. By default, no prefix is added to class names. The prefix must start with an uppercase alpha character.

   If you checked the **Separate words with underscores** check box, an underscore character is inserted between the prefix and the root of the class name.

   Tip Use this feature to implement a naming convention to distinguish between JADE and external class names.

8. In the **Array Name Suffix** text box in the Class Names group box, specify an optional suffix for Array class names, if required. By default, a suffix of **Array** is appended to **Array** class names.

   If you checked the **Separate words with underscores** check box, an underscore character is inserted between the root of the array class name and the suffix.

9. In the **Dictionary Name Suffix** text box in the Class Names group box, specify an optional suffix for dictionary class names, if required. By default, a suffix of **Dict** is appended to dictionary class names.

   If you checked the **Separate words with underscores** check box, an underscore character is inserted between the root of the dictionary class name and the suffix.

10. Uncheck the **Include Keys In Name** check box in the Class Names group box if you do not want dictionary class names to include the names of properties used as keys for the dictionary. By default, dictionary class names include the names of properties used as keys; that is, this check box is checked.

11. In the **Prefix Keys With** text box in the Class Names group box, specify a prefix for the keys part of a dictionary class name, if required. By default, a prefix of **By** is added to dictionary class names that include the names of properties used as keys for the dictionary.

    This text box is enabled only when the **Include Keys In Name** check box is checked.

12. In the **Set Name Suffix** text box in the Class Names group box, specify an optional suffix for set class names, if required. By default, a suffix of **Set** is appended to set class names.

    If you checked the **Separate words with underscores** check box, an underscore character is inserted between the root of the set class name and the suffix.

13. Click the **Next >** button when you have specified your class and property name identifiers. Alternatively, click the **Close** button to close the wizard.

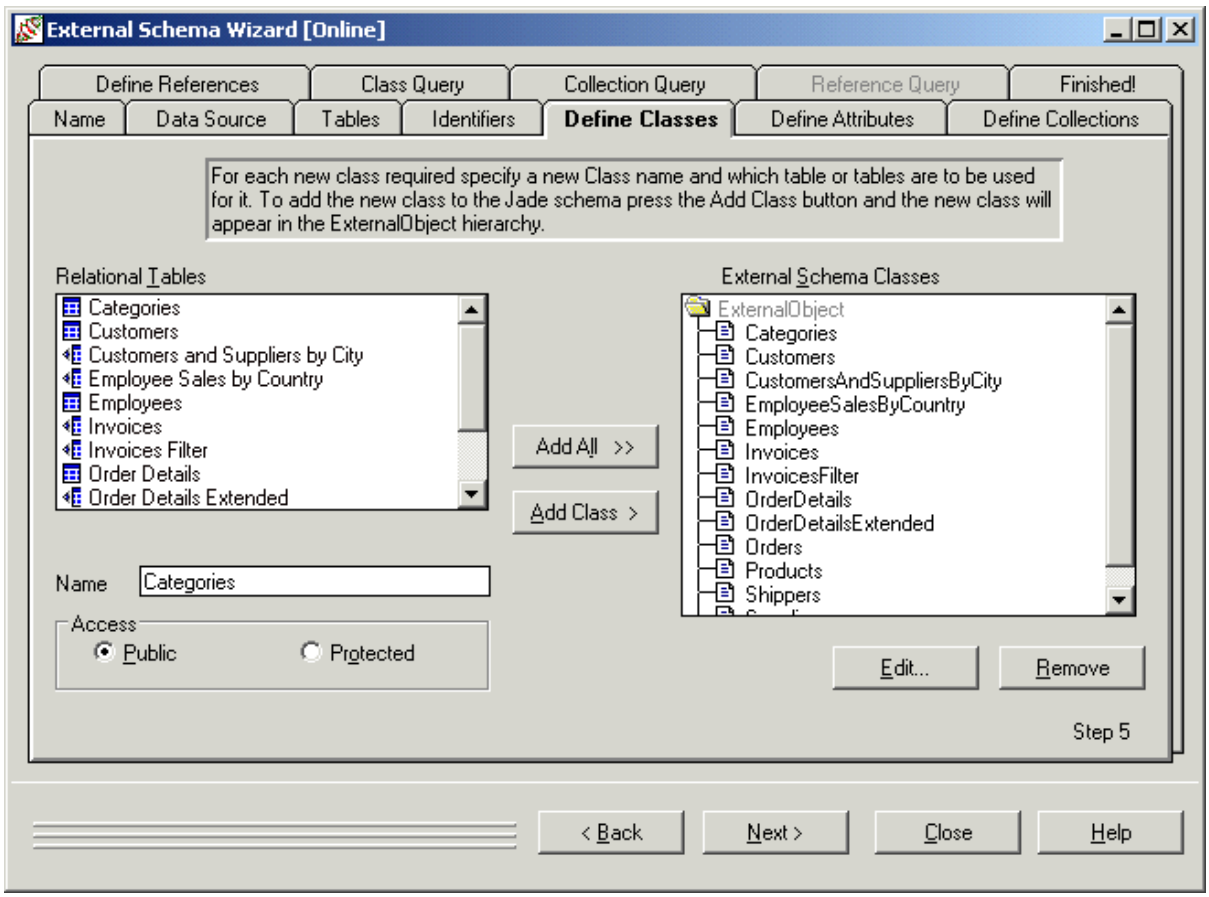
When you click the **Next >** button, the External Schema Wizard then enables you to define the classes that are to be created from relational database tables.

Creating Classes from Tables

When you have specified your class and property name identifiers, the **Define Classes** sheet of the External Schema Wizard then enables you to define the classes that are to be created from relational database tables.

Note You can map multiple tables into one class.

An example of the **Define Classes** sheet of the External Schema Wizard is shown in the following image.



The **Relational Tables** list box alphabetically lists all tables specified in the **Tables** sheet for inclusion in the schema transformation.

A bitmap at the left of each table indicates the type of table; that is, whether it is a base table, a view, or a system table. By default, the first unused table in the list is selected. A table listed in black (that is, one that is not grayed) already has at least one defined class using that table.

The **Name** text box displays the class name to be generated for the selected relational table. If no relational table is selected, this text box is empty and the **Add Class** button is disabled.

The **External Schema Classes** list box lists all the classes defined for this external database schema; that is, all subclasses of the **ExternalObject** class for this external database.

» To define classes from relational tables

1. In the **Relational Tables** list box, select one or more tables that are to be used in your class. (A class that is based on more than one table is called a *join*.)

   As the selection in the **Relational Tables** list box automatically moves to the next unused table in the list, you can add a default class name for each table in the list by successively clicking the **Add Class** button. Alternatively, click the **Add All** button to initiate the creation of a new class for each relational table that is not used by a class.

2. A default class name is generated for the selected table or tables and is displayed in the **Name** text box. Specify a different name for the class, if required.

   The name must be a valid class name that is unique to the **ExternalObject** class.

3. In the Access group box, select the appropriate option button to specify the type of access you require for the currently selected class; that is, protected or public. (The default access type is determined by the user profile.)

4. Click the **Add Class** button to add your class to the **ExternalObject** hierarchy and update the JADE database. The next unused table in the **Relational Tables** list box is then selected.

   Alternatively, click the **Add All** button to initiate the creation of a new class for each relational table that is not used by a class.

5. To delete an existing class that you no longer require in your external database schema, in the **External Schema Classes** list box, select the class or classes that you want to delete, and then click the **Remove** button.

6. To edit the type of access of a class, select the class in the **External Schema Classes** list box that you want to change and then click the **Edit** button.

   The Define Class dialog is then displayed. Controls and sheets that are not valid for the changing of an external database schema class are disabled. For details about using this dialog, see "Defining a Class", in Chapter 3 of the *JADE Development Environment User's Guide*.

7. Click the **Next >** button when you have defined your external database schema classes. Alternatively, click the **Close** button to close the wizard.

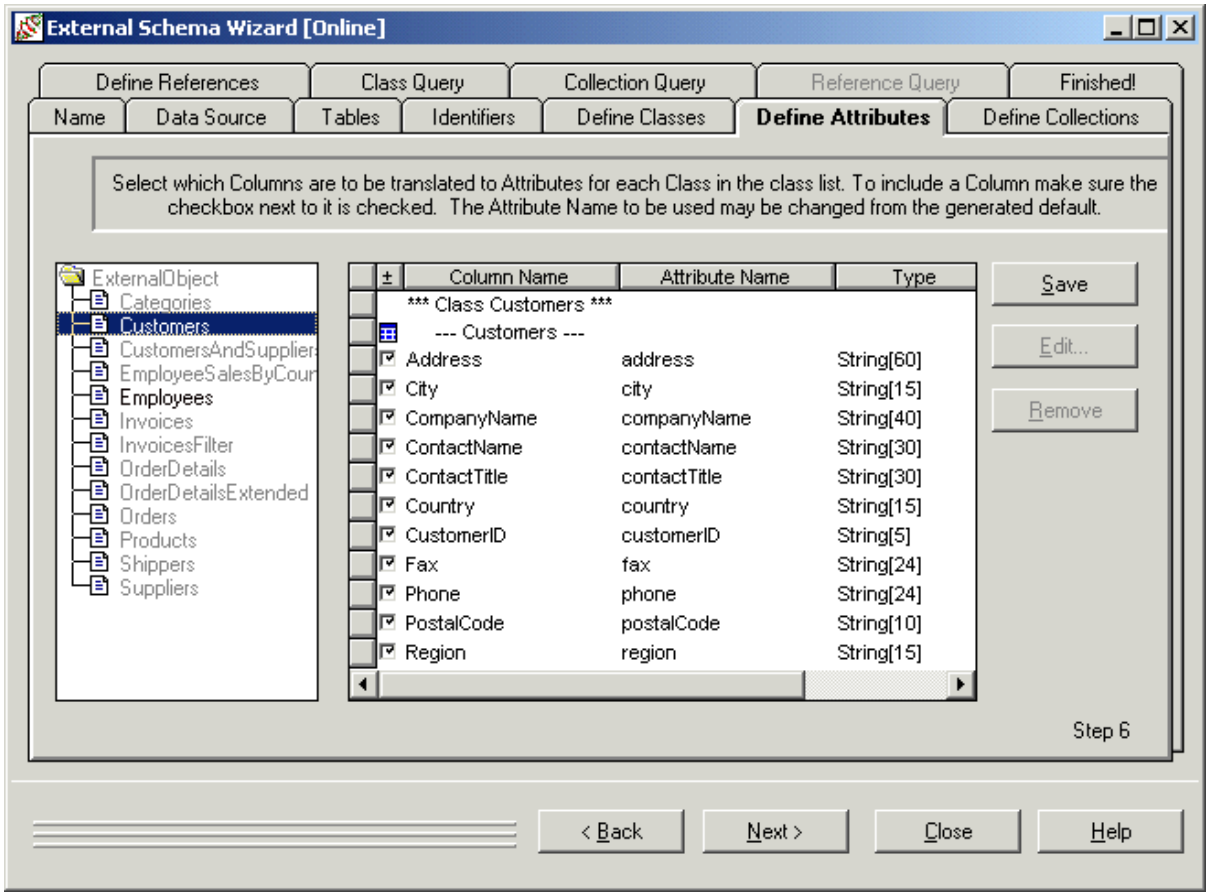
When you click the **Next >** button, the External Schema Wizard then enables you to define attributes for a class.

Defining Attributes for a Class

When you have created classes from tables, the **Define Attributes** sheet of the External Schema Wizard then enables you to add attributes to a class, based on columns of the tables defined in the **Define Classes** sheet.

The creation of class attributes is a prerequisite for the definition of dictionaries.

An example of the **Define Attributes** sheet of the External Schema Wizard is shown in the following image.



The **Classes** list box lists all of the subclasses of the **ExternalObject** class defined for this external database schema. The **Attributes** table displays all of the potential and defined attributes for the class selected in the **Classes** list box.

The rows of the **Attributes** table are grouped into sections, with a separate section for each table included in the class. Each row of the table displays a check box at the left of a table column of one of the tables included in the class and a generated attribute name and default JADE primitive type for that attribute, based on the description of the table column from the relational database.

When you first access this sheet and no attributes are defined for a class, all the attributes in the **Attributes** table are checked. When you have added one or more attributes to a class, only those attributes are checked.

Rows in the **Attributes** table that correspond to an existing attribute for the selected class are displayed in black. Rows that do not correspond to an existing attribute are displayed in gray and enable you to change the attribute name by specifying the required name in an edit text box in the table. A changed attribute name is then displayed in black.

To define attributes for a class

1. In the **Classes** list box, select the **ExternalObject** subclass whose attributes you want to define.

   The **Attributes** table is then automatically populated with all of the possible attributes that can be added to that class. If you have not yet defined any attributes for the selected class, all of the attributes in the **Attributes** table are checked.

If you have added one or more attributes to that class, only those attributes are checked; that is, your previous selections are retained and displayed when you next access the **Define Attributes** sheet of the Wizard.

Note Any attributes that correspond to primary key columns are always checked and added to the selected class, as these are required for references and update operations.

2. To exclude a table column for transformation to an attribute, uncheck the check box at the left of that column in the **Attributes** table.
3. When you have made the required changes to the attributes for the selected class, click the **Save** button. The attributes for that class are then saved.

   The selection in the **Classes** list box automatically moves to the next **ExternalObject** subclass that has no attributes defined for it in the **Attributes** table, enabling you to define attributes for each class.
4. Repeat steps 1 through 3 for each **ExternalObject** subclass whose attributes you want to define.
5. To change a generated attribute, select the attribute name and then click the **Edit** button. The Define Attributes dialog is then displayed. Options that are not valid for the changing of an external database schema attribute are disabled.

   You can change the attribute type, length, scale factor, precision, and access mode. For details, see "Adding an Attribute Property", in Chapter 4 of the *JADE Development Environment User's Guide*.

   The JADE primitive type of the attribute can be changed only if a reasonable conversion exists. Symmetric conversions only are allowed. If a data type conversion produces a truncation or that type is not defined for your ODBC driver, an exception is raised at run time. For details, see "Mapping an SQL Data Type to a JADE Primitive Type", in Chapter 2.
6. To delete a defined attribute, uncheck its check box and then click the **Save** button. Alternatively, select the required attribute in the **Attributes** table and then click the **Remove** button. (The **Remove** button is disabled if you have not selected an attribute or you have selected an attribute based on a primary key.)

   If the attribute is used anywhere in your schema, you are prompted to confirm that it is to be deleted.
7. Click the **Next >** button when you have defined the attributes for your classes. (Alternatively, click the **Close** button to close the wizard.)

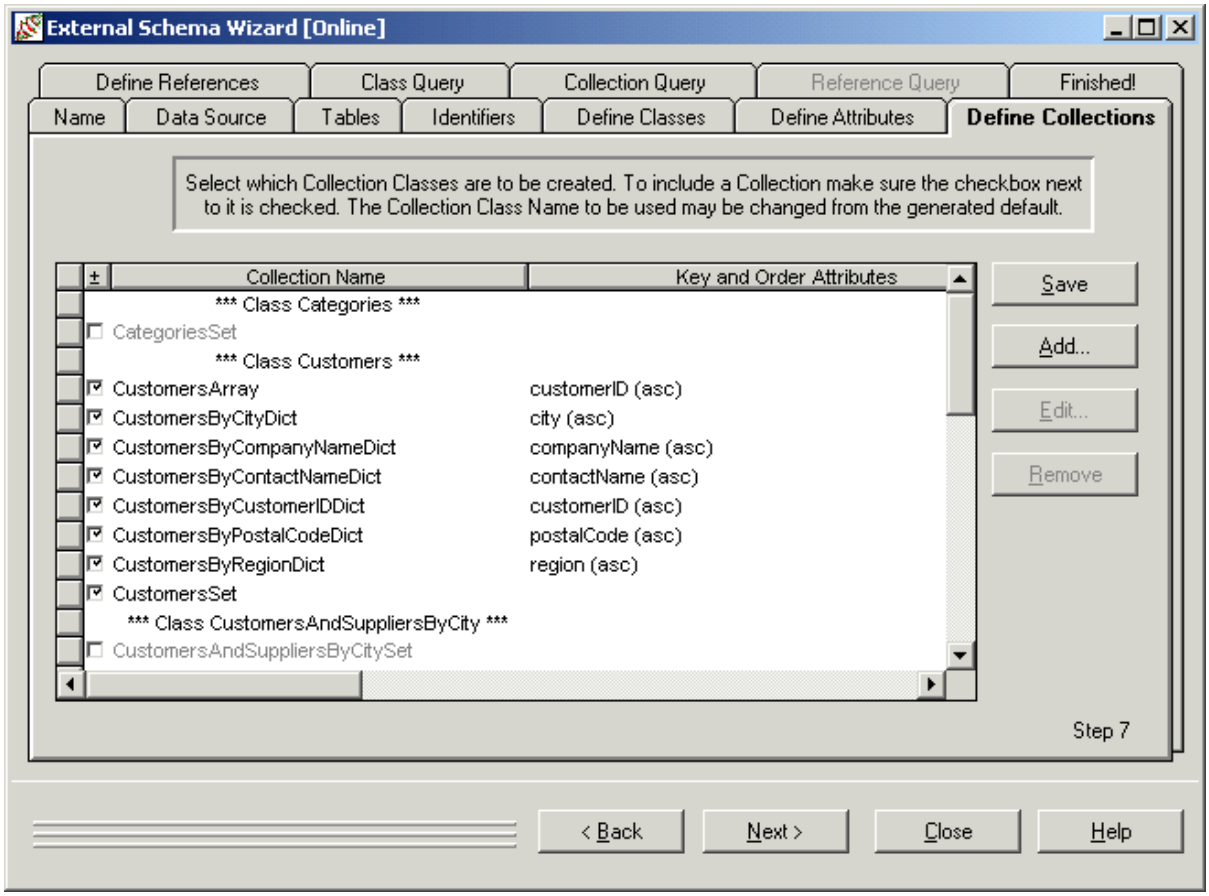
When you click the **Next >** button, the External Schema Wizard then enables you to define collections.

Note If you have made any changes to your attribute definitions that you have not yet saved, a warning is issued before the next sheet is displayed.

Defining Collection Classes

When you have defined attributes for your classes, the **Define Collections** sheet of the External Schema Wizard then enables you to define your collection classes, which you can then use to define references in the following step.

An example of the **Define Collections** sheet of the External Schema Wizard is shown in the following image.



The **Collections** table displays the potential and defined collections for the current external database schema.

The rows of the **Collections** table are grouped into sections, with a separate section for each member class of the collection; that is, the subclasses of the **ExternalObject** class. Each section contains a row for an array-based collection, a set-based collection, and a dictionary-based collection.

Dictionaries are defined for each primary key and for each index with keys of the attributes that correspond to the table columns.

Each row of the table displays a check box at the left of the generated collection name. The row for a dictionary collection also contains its keys, and for an array collection, its order attributes. Rows that correspond to an existing collection are displayed in black.

Rows that do not correspond to an existing collection are displayed in gray and enable you to change the collection name by specifying the required name in an edit text box in the table. A changed collection name is then displayed in black.

To define collections

1. To include all collection subclasses in your external database schema with the displayed generated names, click the **Save** button.
2. To exclude a collection subclass from your external database schema, uncheck the check box at the left of that column in the **Collections** table.

If you have not yet defined any collection subclasses, all generated collection subclasses are checked except for array collections. If you have added one or more collection subclasses, only those collection classes are checked; that is, your previous selections are retained and displayed when you next access the **Define Collections** sheet of the Wizard.

3. Click the **Save** button, to retain your selection or de-selection of the collection class.
4. Repeat steps 2 and 3 for each collection subclass that you want to create or an existing external database schema collection class that you now want to exclude.
5. To change a generated collection class name or its options (that is, access type, keys, and the sort order):
   a. Select the required class in the **Collections** table.
   b. Make the required changes in the Define Class dialog that is then displayed. (Options that do not apply to the external database schema transformation process are disabled.) For details, see "Defining Your Own Classes", in Chapter 3 of the *JADE Development Environment User's Guide*.
   c. Click the **OK** button in the Define Class dialog to save your changes and close the dialog.
6. To add a collection subclass that is not automatically generated:
   a. Click the **Add** button to specify your new **ExternalCollection** subclass in the Define Class dialog that is then displayed. (Options that do not apply to the external database schema transformation process are disabled.) For details, see "Defining Your Own Classes", in Chapter 3 of the *JADE Development Environment User's Guide*.

      Select the **ExternalDictionary**, **ExternalArray**, or **ExternalSet** class as the superclass.
   b. Click the **OK** button in the Define Class dialog to save your new collection class (which is then displayed in the **Collections** table).
7. To delete a collection subclass, uncheck its check box and then click the **Save** button. Alternatively, select the class that you want to delete in the **Collections** table, and then click the **Remove** button. If the collection is used anywhere in the external database, you are prompted to confirm that you want to delete it.
8. Click the **Next >** button when you have defined all of the required collection classes.

   Alternatively, click the **Close** button to close the wizard.

When you click the **Next >** button, the External Schema Wizard then enables you to define references.

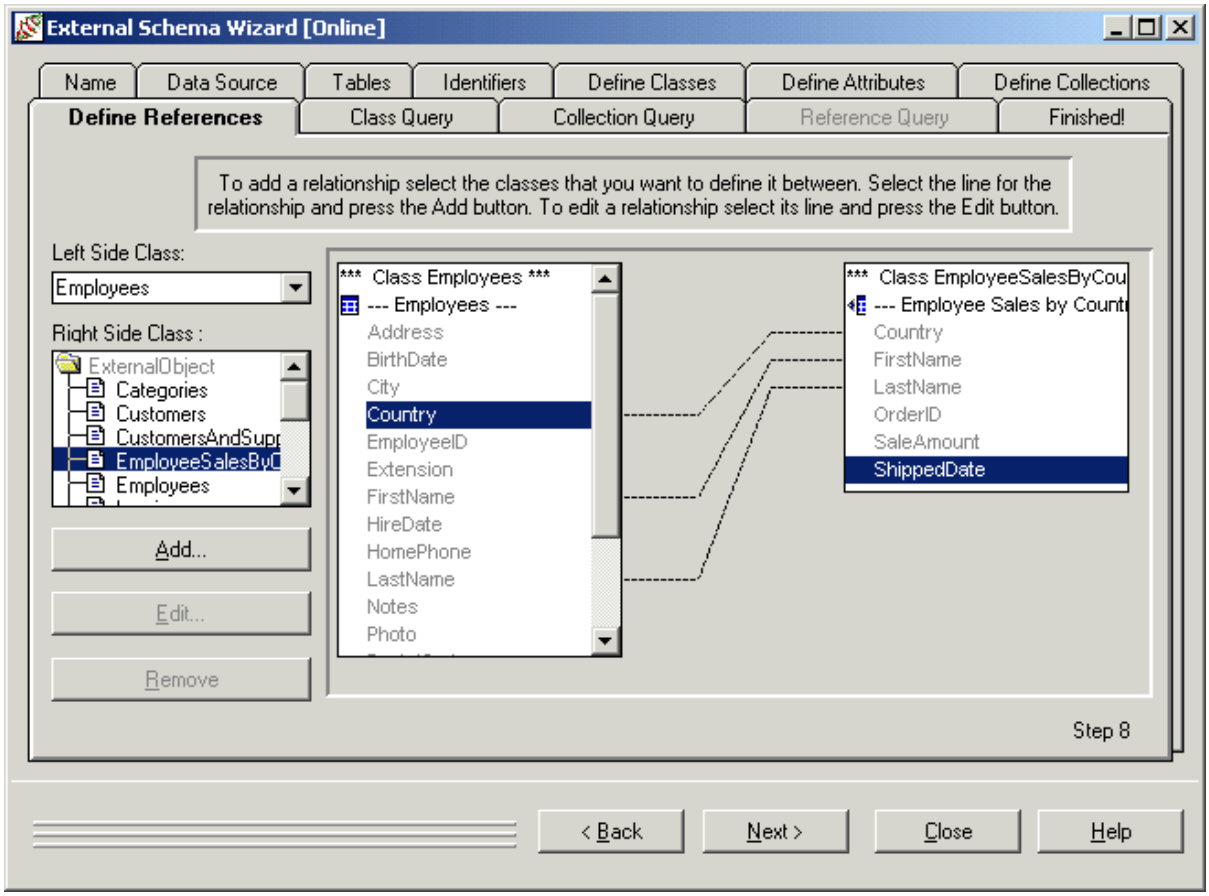
Note If you have made any changes to your collection definitions that you have not yet saved, a warning is issued before the next sheet is displayed.

Adding References to a Class

When you have defined your collection subclasses and dictionary class keys, the **Define References** sheet of the External Schema Wizard then enables you to add references to your external database classes. References added to a class are based on any foreign keys defined for or by the class member tables, or by an association of arbitrary columns. *Foreign* keys represent one-to-one and one-to-many relationships. *Primary* keys determine the object identity for a class formed by the union of tables.

Note A reference property is a *one* relationship and a collection property is a *many* relationship. If a property defining the relationship is not a collection, only one reference is allowed. The collection can be a set, an array, or a dictionary.

An example of the **Define References** sheet of the External Schema Wizard is shown in the following image.



The **Left Side Class** combo box and the **Right Side Class** list box list all of the **ExternalObject** subclasses defined for your external database schema, to enable you to select the classes between which to define a relationship. The **Left Side Class** combo box also includes the external relational database class that corresponds to this external database schema, to enable you to define a root object reference.

The tables at the right of the sheet represent the two classes that you select. The tables display a section for each relational table defined for the selected classes (by using the **Define Classes** sheet, in an earlier step). These sections list the columns of the tables.

Columns that participate in the primary key of the table are indicated with a bitmap, as are columns that participate in a foreign key and those that participate in both types of key. A dotted (broken) line drawn between these two tables indicates a candidate relationship. A solid line indicates a relationship that you have already defined.

The types of relationship that can be shown are listed in the following table.

Relationship	Description
Foreign key-based	Uses primary and foreign key information from the relational database to determine the columns that are involved
Name-based	Matches columns with the same name and similar types, to identify potential relationships
User-defined	Relies on you to select the columns that are involved in the relationship

If a relationship uses several columns on one side of the relationship, these columns are bracketed. A directional arrow on a line indicates the relationship from a foreign key to a primary key.

A relationship line that already has a defined reference is displayed with a **1** or **M** character at each end of the line, indicating the cardinality of the relationship; that is, one or many.

To specify a reference

1. In the **Left Side Class** combo box, specify or select the class for the left-hand side of the relationship.
2. In the **Right Side Class** list box, select the class for the right-hand side of the relationship.
3. Click on the line that represents a candidate relationship.

   Alternatively, if you want to define a relationship that is not automatically generated as a candidate relationship, perform the following actions.

   a. In the first table, select the column or columns that are to form one end of the reference.

   b. In the second table, select the column or columns that you want to form the other end of the reference.
4. Click the **Add** button.

Tip Alternatively, to specify an arbitrary relationship, you can drag-and-drop between a single column in each table, or you can select multiple compatible columns with the same type in the same order and then click the **Add** button.

The External Schema Wizard Relationship dialog is then displayed, to enable you to define your reference.

To edit an existing reference

1. Click the relationship line between the two references. The selected line is then displayed in white. You can select only one relationship line at any time.
2. Click the **Edit** button.

The External Schema Wizard Reference dialog is then displayed, to enable you to change the existing reference. For details, see "Using the External Schema Wizard Relationship Dialog", in the following subsection.

To delete an existing reference

1. Click the relationship line between the two references.
2. Click the **Remove** button.

The references between the two classes are then removed.

To proceed to the next step of the schema transformation process

- Click the **Next >** button when you have defined all required class references for your external database schema.

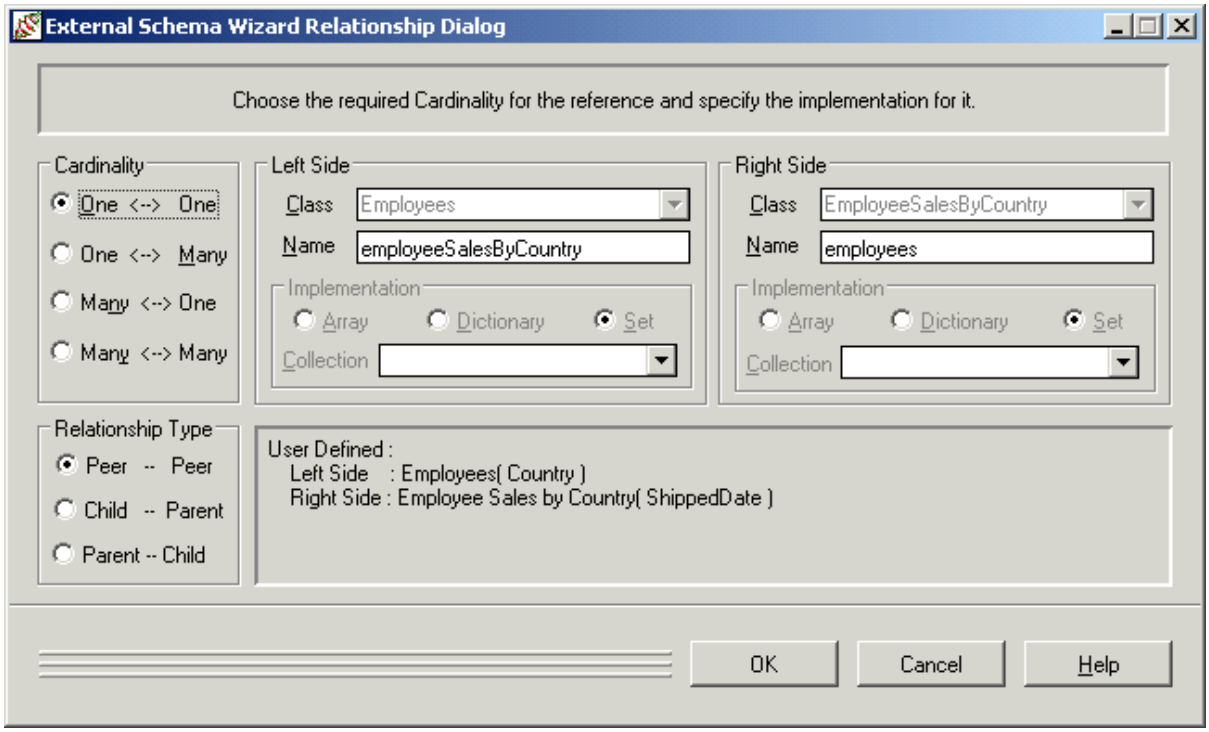
  Alternatively, click the **Close** button to close the wizard.

When you click the **Next >** button, the External Schema Wizard then enables you to refine your external database schema class SQL queries.

Using the External Schema Wizard Relationship Dialog

When you have selected a relationship line between two columns on the **Define References** sheet and clicked the **Add** button or the **Edit** button, the External Schema Wizard Relationship dialog is then displayed.

An example of the External Schema Wizard Relationship dialog is shown in the following image.



A reference property is a *one* relationship and a collection property is a *many* relationship. If a property defining the relationship is not a collection, only one reference is allowed. The collection can be an array, a set, or a dictionary with member keys.

Note All references are read-only, and all relationships between two **ExternalObject** subclasses are explicit inverse relationships; that is, you must define one reference for each side of the relationship.

The Left Side group box and Right Side group box represent the two sides of the relationship and correspond to the two classes that you selected in the **Define References** sheet. Each group box displays the selected class and a generated reference name.

The Implementation group boxes enable you to select the type of collection that is to be used to implement a many-sided relationship. The display area at the lower right of the dialog describes the selected relationship in terms of relational tables and columns.

The classes selected in the **Define References** dialog are displayed in the **Class** list boxes. Values that cannot be altered are disabled. For details about JADE references, see "Adding a Reference Property", in Chapter 4 of the *JADE Development Environment User's Guide*.

» To add or change a reference

1. If you do not want the generated default cardinality for your reference, select the appropriate option button in the Cardinality group box.

Change the cardinality to indicate a many-sided relationship, if required. The Implementation group box on the corresponding side of the relationship is then enabled and the name of a reference on that side of the relationship is replaced with the plural form of the name if you have not yet edited the reference on that side of the relationship.

2. If the generated default names are not suitable, change the names displayed in the **Name** text box of the Left Side group box and Right Side group box.

3. If an Implementation group box is enabled for either side, change the collection type by selecting the appropriate option button, if required.

   Any collections that match the selected implementation option are displayed in the **Collection** combo box, with the list box portion displaying the first match. Select another collection that matches your selected implementation, if required.

4. In the Relationship Type group box, select the appropriate option button to specify the composition semantics that you require. The relationship is between instances of two classes.

   A peer-to-peer relationship is an equivalent relationship between two objects. A parent-child relationship is a relationship in which the child object belongs to, or is subsidiary, to the parent object. This enables you to capture additional semantic information only; for example, for modeling purposes.

   There is no enforcement of deletion semantics, as the instances do not reside in the JADE database. (For details, see "Object Lifetimes", in Chapter 2.)

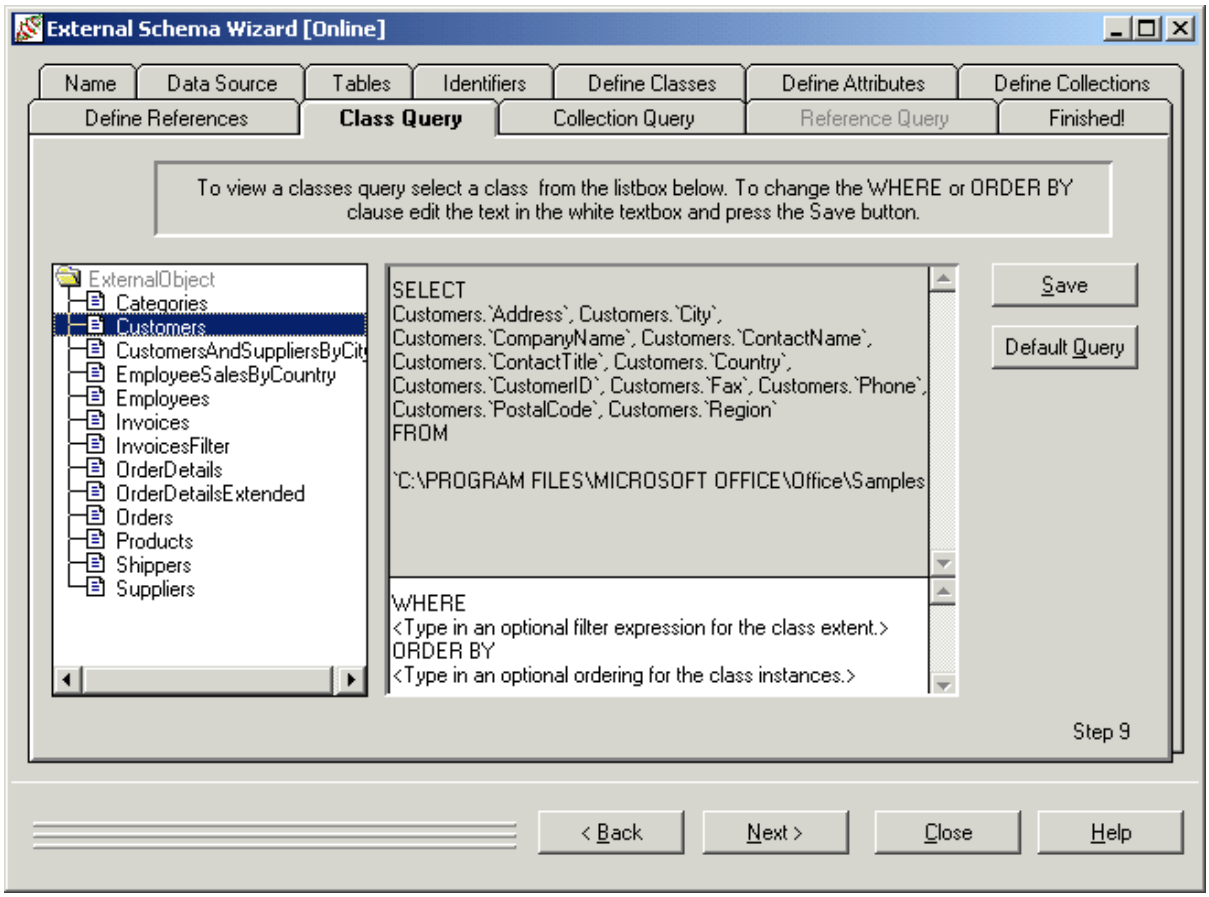
   Note In an external database schema, child objects are *not* automatically deleted when a parent object is deleted. (This differs from the JADE implementation.)

5. Click the **OK** button when you have defined your references. (Alternatively, click the **Cancel** button to abandon your selections.)

When you click the **OK** button (and you confirm that you want to use an existing reference, if required), the **Define References** sheet is then displayed, to enable you to define more references or to proceed to the next step of the External Schema Wizard.

Refining the Class Membership Query

When you have added references to your classes, the **Class Query** sheet of the External Schema Wizard, shown in the following image, then enables you to inspect and tailor the class membership query, based on the member tables and any columns used as attributes or references.



To inspect or tailor a generated class membership query

1. In the **Classes** list box, which displays all classes defined for the current external database schema, select the class whose membership query you want to view. The **SELECT** and **FROM** clauses of the generated SQL query for your selected class are then displayed in the display-only (gray) area in the upper center of the sheet.

2. In the edit area (the white area) at the lower center of the sheet, specify an optional **WHERE** clause, if required. If your class uses two or more tables, you must specify the **JOIN** condition for the tables in this edit area. You can also use the **WHERE** clause to exclude rows of the selection (equivalent to object instances) from the class membership.

3. In the edit area (the white area) at the lower center of the sheet, specify an optional **ORDER BY** clause, if required. You can use the **ORDER BY** clause to determine the order in which rows of the selection (equivalent to object instances) are iterated through when accessing the instances collection of the selected class.

4. To save your specified SQL query clause for the class, click the **Save** button. The query is then validated by the current driver (if you are currently connected to the external relational database) before your query is

saved.

5. To return to your original query for the selected class (that is, to refresh the screen), click the **Default Query** button.

6. Repeat steps 1 through 4 for each class whose query you want to inspect or to tailor.

   Note If you have made any changes to your class query that you have not yet saved, a warning is issued before the next class (or sheet) is displayed.

7. Click the **Next >** button when you have inspected or tailored all required class membership queries. Alternatively, click the **Close** button to close the wizard.

When you click the **Next >** button, the External Schema Wizard then enables you to inspect **Collection** class queries.

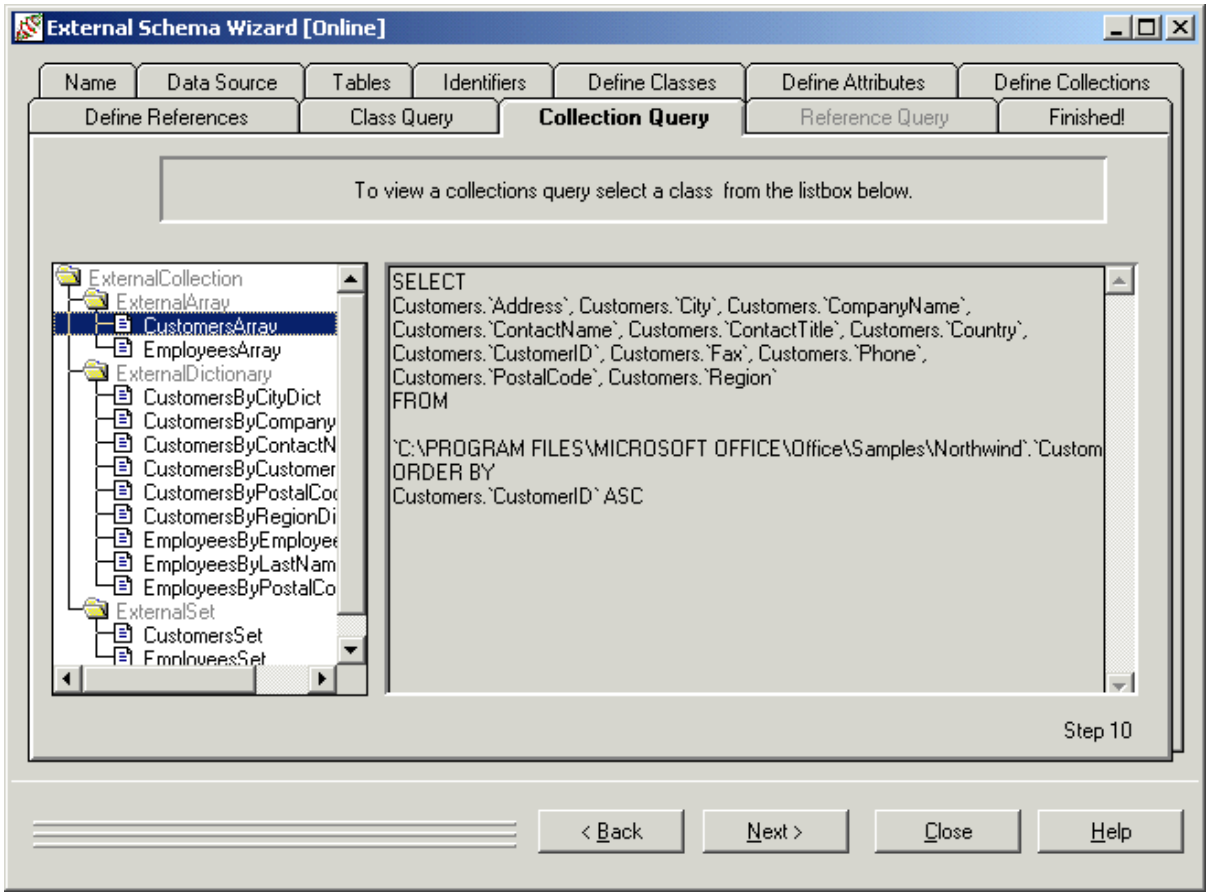
Inspecting Collection Class Queries

When you have refined the class membership queries for your external database schema, the **Collection Query** sheet of the External Schema Wizard is then displayed, to enable you to inspect your collection queries.

An SQL query based on the member tables and any columns that are used as attributes or references is generated for each collection.

Note An **ORDER BY** clause is generated only for dictionary and array collection classes.

An example of the **Collection Query** sheet is shown in the following image.



The **Collections** list box at the left of the sheet lists all collection classes defined for this external database schema.

To inspect a generated collection class query

1. In the **Collections** list box, select the collection class whose query you want to view.

   The SQL Query frame at the right of the sheet then displays the entire SQL query that was generated for the selected collection class. You can view the SQL query only; you cannot tailor a collection query.

   Tip To change your collection criteria, return to the **Define Collections** sheet.

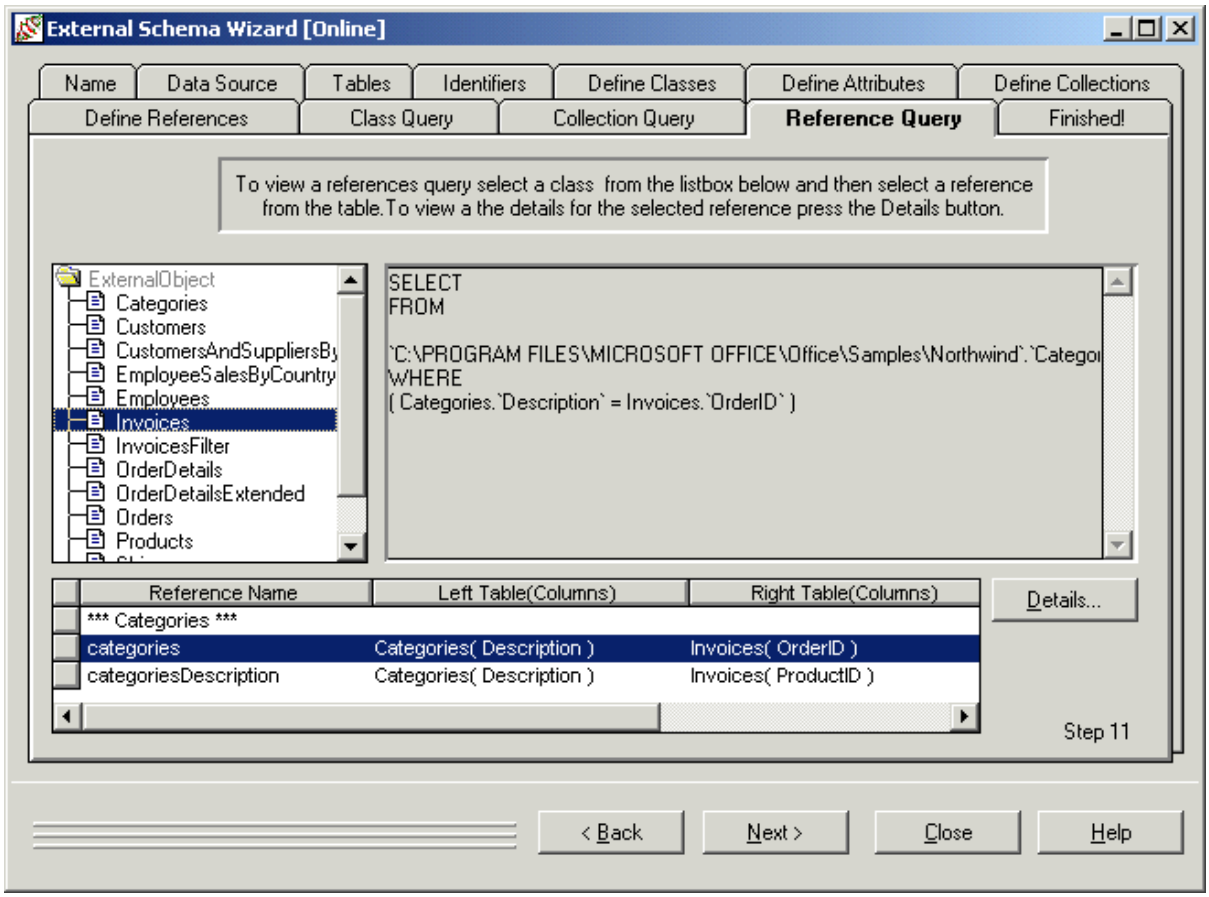
2. Repeat step 1 for each collection class whose query you want to inspect.
3. Click the **Next >** button when you have inspected all required collection queries.

   Alternatively, click the **Close** button to close the wizard.

When you click the **Next >** button, the External Schema Wizard then enables you to inspect your reference queries.

Inspecting Reference Queries

When you have inspected your collection class queries, the **Reference Query** sheet of the External Schema Wizard, shown in the following image, then enables you to inspect your reference queries.



An SQL query is generated for each reference in your external database schema. An **ORDER BY** clause is generated for dictionary and array collections only.

The **Classes** list box at the left of the sheet lists all classes (that is, *From* classes) defined for this external database schema.

To inspect a generated reference query

1. In the **Classes** list box, select the class whose reference query you want to view.

   The rows of the References Table are grouped into sections, with a separate section for each *To* class. Each row of the table displays the name of the reference, the **From Table(Columns)**, and the **To Table(Columns)** of the external relational database. On the far right is a **Details** button.

2. In the **Reference Name** list in the References Table, select the reference to the class whose SQL query you want to view. You can view the SQL query only; you cannot tailor a reference query.

   The SQL Query frame at the upper right of the sheet then displays the entire SQL query that was generated for the selected class reference. You can view the SQL query only; you cannot tailor a reference query.

3. To view the definition of the selected reference, click the **Details** button at the right of the sheet.

The External Schema Wizard Relationship dialog is then displayed in read-only mode, to enable you to view the definition of the selected reference. To edit the relationship, return to the **Define References** sheet.

For details, see "Using the External Schema Wizard Relationship Dialog" and "Adding References to a Class", respectively, earlier in this chapter.

4. Repeat steps 1 through 3 for each reference whose query you want to inspect.
5. Click the **Next >** button when you have inspected all required reference queries.

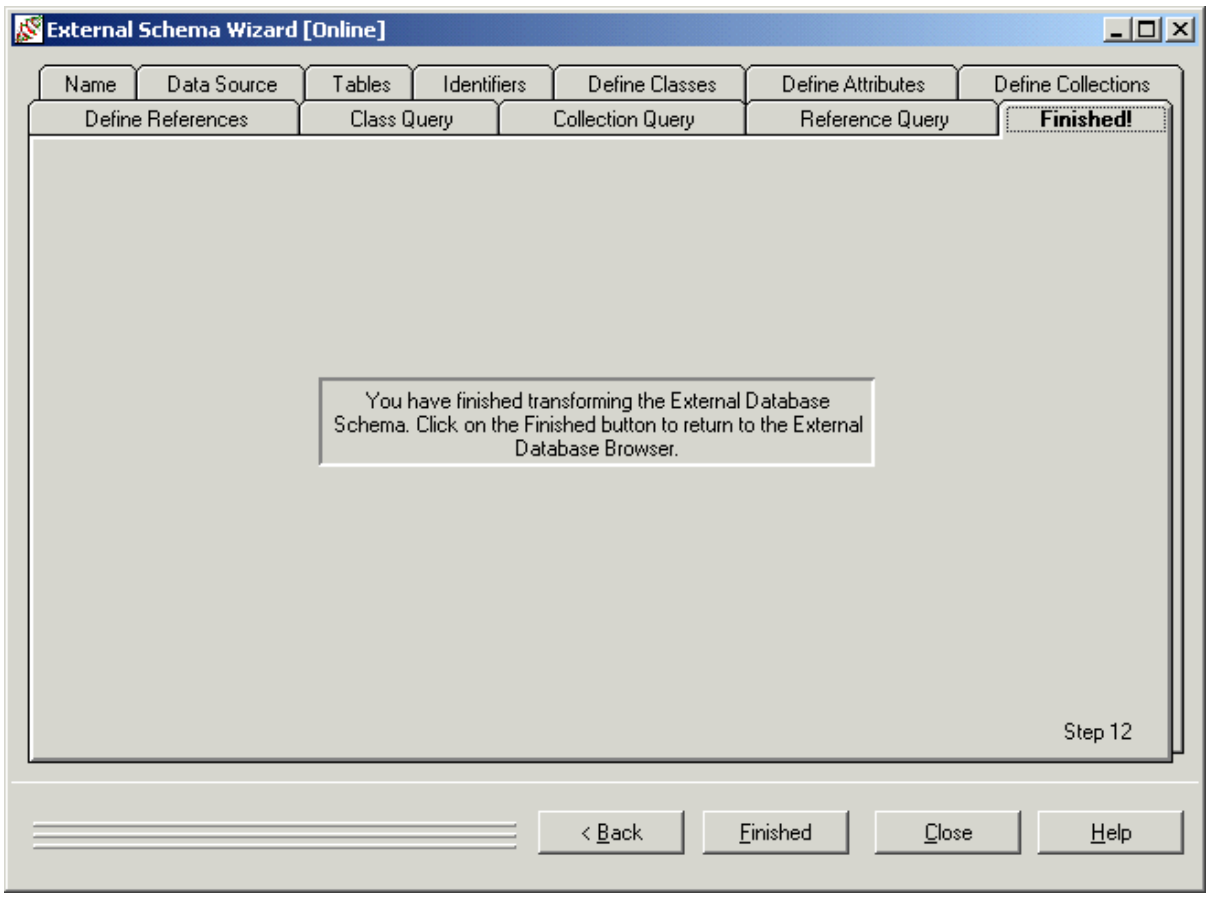
   Alternatively, click the **Close** button to close the wizard.

When you click the **Next >** button, the External Schema Wizard then enables you to confirm that you have finished defining your external database schema.

Finishing Your External Database Schema Definition

When you have inspected your SQL queries for class references, you have finished specifying your external database schema definition.

The **Finished!** sheet of the External Schema Wizard, shown in the following image, indicates that the transformation process is complete and enables you to change the status of the external database schema so that it can be used at run time.



» To confirm that you have finished specifying your external database schema

- Click the **Finished** button.

  Alternatively, click the **< Back** button to redisplay the previous sheet or the **Close** button to close the wizard without marking the external database for runtime use.

When you click the **Finished** button, JADE performs the following actions.

1. Completes the transformation process.
2. Marks your external database schema so that it can be used by the Runtime Query Engine.
3. Closes the External Schema Wizard and then displays the External Databases Browser that contains your new external database schema.

   When you select the external database schema in the External Databases Browser, the status line displays the creation timestamp and your user identifier.

You can now use your external database schema as you can any other part of JADE, by using the ODBC driver and your SQL queries.

Deleting an External Database Schema

From the External Databases Browser, the **Remove** command in the Databases menu enables you to remove (delete) the external database schema that is currently selected.

Notes You cannot remove an external database schema that is currently being accessed by an ODBC driver.

When a subclass of **ExternalObject** is deleted, all collections and references that use the class are deleted. A message box displays a warning, prompting you to confirm that you want to delete the external object.

» To remove an external database schema

1. In the External Databases Browser, select the external database schema that you want to remove.
2. Select the **Remove** command from the Databases menu.
3. A message box is then displayed, to enable you to confirm that you want to remove the selected external database schema.
4. Click the **OK** button to confirm that the selected external database schema is to be removed.

   Alternatively, click the **Cancel** button to abandon the deletion.

The External Databases Browser is then updated to reflect the removal of the selected external database schema. There may be a momentary delay while this updating occurs.

When you delete an external database schema, JADE performs the following actions.

1. Deletes all ODBC catalog information that was held internally.
2. Removes all **ExternalObject** and **ExternalCollection** subclasses defined for this external database schema.
3. Deletes the singleton instance of the external database schema.
4. Removes its own **ExternalDatabase** subclass.

Changing an External Database Schema

Use the External Schema Wizard to maintain an existing external database schema selected in the External Databases Browser. If the definition of the external relational database changes, the transformed (external database schema) definition that is stored in the JADE database may be incomplete or wrong.

To ensure that the external relational database and your external database schema retain concurrency, JADE performs the following actions.

1. Stores a complete description of the parts of the original external schema that are transformed, to allow comparison against any later version of the same schema.
2. Detects differences when establishing a connection to an external database and prevents access at a table level when such differences are detected.
3. Enables you to update the transformed (external database schema) definition, including the removal of obsolete elements. This process indicates new, out-of-date, or deleted elements.

» To change an external database definition

1. Select the external database schema in the External Databases Browser.
2. Select the **Change** command from the Databases menu.

When you change an existing external database schema, the **Name** text box in the first sheet of the External Schema Wizard displays the name of your external database schema. As you cannot change the name of an existing external database schema, this text box is disabled.

The External Schema Wizard is then displayed. For details about using the External Schema Wizard, see "Adding an External Database Schema", earlier in this chapter.

Viewing an External Database Definition in Read-Only Mode

You can browse your external database definitions by using the External Schema Wizard in read-only mode. When you browse an external database in read-only mode, the definition cannot be inadvertently modified.

» To view a selected external database definition in read-only mode

1. Select the external database schema in the External Databases Browser.
2. Select the **View** command from the Databases menu.

The External Schema Wizard is then displayed in read-only mode to enable you to browse through your definition, but not to change it.

For details about the External Schema Wizard, see "Adding an External Database Schema", earlier in this chapter.

Printing an External Database Schema

From the External Databases Browser, the **Print** command in the Databases menu enables you to output information about the currently selected external database schema to your printer.

To print your external database schema

1. In the External Databases Browser, select the external database schema that you want to print.
2. Select the **Print** command from the Databases menu.
3. The Printing progress dialog is then displayed. (You can click the **Cancel** button from the Printing progress dialog to cancel your print request.)

Your print output contains a full description of the external database schema, including connection information, class and table mappings, and property and column mappings.

Extracting an External Database Schema

You can extract an external database schema as part of the schema in which it is defined or you can extract only the external database schema itself.

To extract an external database schema only

1. In the External Databases Browser, select the external database schema that you want to extract.
2. Select the **Extract** command from the Databases menu. The common File Save dialog is then displayed.
3. Change the default file path and names, if required, and then click the **OK** button.

   By default, your external database schema is extracted as .**scm** and .**ddb** or .**ddx** files that are prefixed with the name of your external database schema; for example, **ExtDb.scm** and **ExtDb.ddb**.

   These files are extracted to your JADE working directory by default; for example:

   ```
   s:\jade\test\bin
   ```

For details about extracting the external database schema as part of the schema in which it is defined, see "Extracting Your Schema", in Chapter 10 of the *JADE Development Environment User's Guide*.

Loading an External Database Schema

You can load an external database schema as part of the schema in which it is defined or you can load only an extracted database schema itself.

To load an external database schema only

1. In the External Databases Browser, select the **Load** command from the Databases menu.

   The Load Options dialog is then displayed.
2. In the **Schema File Name** text box, specify the name and location of the external database schema file (.**scm** file) that you want to load. You must specify a value in this text box.

   If you are unsure of your file name or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file.
3. In the **Forms File Name** text box, specify the name and location of the extracted form and data definition (.**ddb** or .**ddx**) file that you want to load.

   If you are unsure of your file name or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file.

4. Click the **OK** button to confirm your selections.

   Alternatively, click the **Cancel** button to abandon your selections.

The loading of your external database schema is then initiated.

For details about loading the external database schema as part of the schema in which it is defined, see "Loading Your Schema", in Chapter 10 of the *JADE Development Environment User's Guide*.

Chapter 4 Using External Component Libraries

This chapter covers the following topics.

Overview

JADE enables you to interface to two types of external components in a similar way.

- External ActiveX control and automation servers
- NET components and controls

ActiveX automation enables you to control a Component Object Model (COM) application from within JADE. In JADE, the **ActiveXControl** subclass of the **Control** class supports the use of all interfaces on the control whereas the **Ocx** control that was available in JADE release 5.0 and earlier (which is retained for backward compatibility) provides a limited interface to ActiveX.

A set of interface classes (which are subclasses of the **IDispatch** class) is generated, and these classes map to each of the interfaces defined for an ActiveX object. You then access the functionality of the ActiveX objects through these interface classes, using the methods and properties that have been generated during the ActiveX import process.

In JADE thin client mode, ActiveX control and automation objects and .NET controls run only on the presentation client.

A selective extract of an ActiveX object extracts only class, method, and property descriptions of the selected ActiveX classes, and *not* the invisible user data that goes with these classes. To extract all ActiveX data so it is usable in another schema that has not had the ActiveX objects loaded or imported, perform a full schema extract or extract the specific ActiveX object (from the Components menu).

The Component Object Model (COM) Standard

COM is a standard that defines how objects and users of those objects interact. As this is a binary standard, many different ActiveX objects can use objects defined within COM, regardless of the language used. An ActiveX control, for example, can work with a Visual Basic or a C++ application.

COM objects are manipulated by an *interface*, which is simply a set of methods that can be called and properties that you can set or get. Most objects have many different interfaces. The most important of these interfaces is the *IUnknown* interface, which all COM objects implement and all other interfaces inherit. Although **IUnknown** is the only interface that all objects must support, most objects also support the *IDispatch* interface. Objects that support the **IDispatch** interface are called *automation servers*.

A *type library* is a special file (which usually has a .**tlb** extension but can also be imbedded in an executable or dynamic link library file) that contains a description of an object and its interfaces. Some of the information that this file contains includes method names, parameters, and parameter types.

ActiveX Automation

Automation is the ability of a client to drive or direct a COM object by calling methods or setting properties using one or more interfaces of that object. For example, Microsoft Excel and Word are automation controllers; that is, they are objects that can be controlled by automation.

Automation is simply the execution of a set of commands that set and get properties and call methods using the properties and methods of the generated ActiveX interface classes. Each ActiveX automation library has a corresponding class of the ActiveX automation type library name created as a subclass of the **ActiveXAutomation** class. This ActiveX library class is the superclass that supports the automation classes within that library.

Each ActiveX automation class has a set of properties and methods created that correspond to the default interfaces for that class.

Creating an instance of the ActiveX automation object in JADE creates the **ActiveXAutomation** object. An example of an ActiveX automation type library is **Microsoft Word 9.0 Object Library**, which creates the automation object classes for Word 97.

When JADE initializes a COM object, the first thread in the application must be the last thread that finalizes (un-initializes) the application. Although **jade.exe** and **jadapp.exe** do this, JADE applications such as **jadrap.exe** do not do so. When you run automation objects on the database server and automation is to be performed on the database server, you must ensure that the first thread to initialize COM is the last thread to finalize (un-initialize) it. For example, if two server applications that use an **ActiveXAutomation** object are started, the first application must run longer than the second application.

Note Using an automation object in server methods can cause problems, as there is no way to ensure that such server methods are run on a specific thread and to control when those threads are closed down.

In such situations, you could start a server application that does nothing more than start the automation object and that will not close until the database server is closed down. We therefore do not recommend that you run automation objects on the database server, particularly in-process objects (that is, DLLs), as these can interfere with the operation of JADE.

ActiveX Controls

An ActiveX control is a special sort of COM object, which is a user interface object that implements a number of interfaces that support its use on forms. (This control was formerly known as the OCX control.)

An ActiveX control can optionally have a graphical user interface, and can fire events. This enables existing third-party functions, such as highly specialized controls, to be used within your JADE applications.

Each ActiveX control library has a corresponding class of the ActiveX control type library name created as a subclass of the **ActiveXControl** class. This ActiveX library class is the superclass that supports the control classes within that library. Each control class can have a set of properties, methods, and events created that correspond to the default interfaces.

Creating an instance of the ActiveX control in JADE does not create an instance of the **ActiveXControl** object. This occurs only when the control is added to a form.

ActiveX controls have a default interface and can have a default event interface.

An imported ActiveX control is added to the Control palette of the JADE Painter, and then cannot be distinguished from standard JADE-supplied controls.

Note If you have added a property to an imported **ActiveXControl** object and flagged that property as a design-time property (for details, see "Selecting Your Design Time Properties", in Chapter 5 of the *JADE Developer's Reference*), any value assigned to that property by using JADE Painter Properties dialog will not be propagated through to the runtime instance of that control when the form is created.

If the property value is required in the runtime instance, you can copy it from the persistent instance to the runtime instance in the **windowCreated** method on the **ActiveXControl** subclass. (As the **windowCreated** method will be a reimplementation of the **Control**::**windowCreated** method, you must include an **inheritMethod** instruction in your **windowCreated** method.)

An example of an ActiveX control type library is **Microsoft Windows Common Controls 6.0**, which creates classes for the common Microsoft Windows controls; for example, the **ImageList**, **Slider**, and **TreeView** controls.

ActiveX Interfaces

ActiveX objects are manipulated by their interfaces. An *interface* consists of a set of properties that you can set or get, and a set of methods that can be called. The caller of an ActiveX object needs to know the property types and method names and parameters to make use of these interfaces.

A subclass of the **IDispatch** class is generated for each of the interfaces defined for an ActiveX automation or control object (which have had subclasses of **ActiveXAutomation** and **ActiveXControl** generated). Each of these interface classes has properties and methods generated that correspond to the properties and methods of the interface.

Note You can create neither transient nor persistent instances of ActiveX interface classes. Instances of these classes are created by JADE at run time when an ActiveX object supplies an interface pointer, and they are deleted when the ActiveX object that is using them is deleted. However, you can delete an instance of an interface if you want to release resources.

Use the **getInterface** method provided by instances of the **ActiveXAutomation** and **ActiveXControl** classes to access any interface for an imported ActiveX object.

How JADE Imports ActiveX Object Definitions

When you import ActiveX object definitions into JADE, the actions that are taken are determined by the type of object; that is, an ActiveX interface, control, or automation class. For details, see the following subsections.

ActiveX Interface Classes

A JADE class is created as a subclass of the **IDispatch** class for each **dispInterface** defined in the type library. A class constant called **GUID**, which is a binary value corresponding to the Globally Unique Identifier (GUID) of the interface, is added to each class. JADE properties and methods are added to the class that corresponds to the properties and methods of the interface.

Some ActiveX type libraries do not expose any properties but define a **get** and a **put** method for each property. During the import process, **get** and **put** pairs of methods on an interface are imported as a JADE property with a mapping method.

The two cases in which **get** and **put** pairs of methods cannot be mapped to a JADE property are:

- When the **get** method has a parameter or the **put** method has more than one parameter, as JADE does not support mapping methods with more than one *value* parameter.
- When the **get** and **put** methods operate on a VARIANT type property, as Variant properties are imported as a JADE **Any** primitive type and mapping methods cannot be defined for a property of type **Any**.

If either of these is the case, JADE creates **get** and **set** method pairs.

Control Classes

A JADE class is created for each registered component object class (coclass) defined in the type library that is flagged as a control in the operating system registry and where the library is imported as a control library. These control object classes are created as a subclass of the class created to represent the library, which is in turn a subclass of the **ActiveXControl** class.

A *coclass* in ActiveX corresponds to a class in JADE. An *interface* is a definition of properties and methods that can be used to access an instance of a coclass.

The **GUID** class constant, which is a binary value containing the object class identifier (CLSID), is created for the generated control type library class.

Methods and properties are created in the same way as interfaces. The methods and properties are based on the default interface and the default event interface (if any) for the object. Calling a method on an object is the same as calling the method of the default interface.

Automation Classes

A JADE class is created for each registered coclass defined in the type library that is not flagged as a control in the operating system registry. These automation object classes are created as a subclass of the class created to represent the library, which is in turn a subclass of the **ActiveXAutomation** class.

The **GUID** class constant, which represents the object **CLSID**, is created for the generated automation type library class. The generated properties and methods are based on the default interface of the object.

Using Generated ActiveX Classes

The following subsections describe the use of **ActiveXControl** classes and **ActiveXAutomation** classes generated from the imported ActiveX type libraries.

Automation events are handled differently from controls. In automation, you must register your interest in a specific set of events. For details, see "Using Automation Events", later in this chapter.

Using the Generated ActiveX Control Classes

Using ActiveX controls from within JADE is identical to the use of the **Ocx** control class.

The creation of an ActiveX control creates only the JADE object. The ActiveX object itself is not created until the control is attached to the form. JADE creates a maximum of two transient instances of the interface classes (subclasses of the **IDispatch** class): one that corresponds to the default interface and the other to the default event interface. Reference properties on the JADE ActiveX control object hold reference to these interface instances.

If the ActiveX object returns a reference to another interface in response to a method call or a **get** action on a property, JADE creates an instance of the corresponding JADE interface class and returns a reference to that instance. (A mapping is maintained between JADE interface instances and the interface pointers of ActiveX controls.)

Note As **ActiveXControl** class methods run only on the client node, all methods generated for imported ActiveX controls include the **clientExecution** option in the method signature.

Using the Generated ActiveX Automation Classes

To use an automation object from within JADE, you simply create an instance of the JADE class (that is, a subclass of the **ActiveXAutomation** class) that corresponds to the automation object. The **ActiveXAutomation** class **createAutomationObject** method is then called to create the automation object and a transient instance of the default interface (which is a subclass of the **IDispatch** class).

A reference is established between the automation class and its default interface. You can then call JADE automation class methods as you can for any other JADE class. These method calls are passed by the default interface to the actual automation object.

If the automation object returns a reference to another interface in response to a method call or the getting of a property, JADE creates an instance of the corresponding JADE interface class and returns a reference to that instance instead. (A mapping is maintained between JADE interface instances and automation server interface instances.)

Note You can create only transient instances of **ActiveXAutomation** subclasses.

The following example is a Workspace or **JadeScript** method that loads three Excel cells with data, draws a chart, and prints the result.

```
chartExample();
vars
    xl     : ExcelApp;      // ActiveX automation subclass
    wrkSht : Worksheet;     // interface subclass of the IDispatch class
    sht    : I_Worksheet;   // interface subclass of the IDispatch class
    rng    : Range;         // interface subclass of the IDispatch class
    chrts  : Sheets;        // interface subclass of the IDispatch class
    chrt   : I_Chart;       // interface subclass of the IDispatch class
begin
    // Start Excel
    create xl;
    xl.createAutomationObject;
    xl.visible := true;                     // See what's going on
    xl.workbooks.add(xl.XlWorksheet);       // Add Workbook (with one sheet)
    sht := xl.activeSheet.I_Worksheet;      // Get top sheet and fill cells
    sht.range("A1", null).putValue("One");
    sht.range("B1", null).putValue("Two");
    sht.range("C1", null).putValue("Three");
    sht.range("A2", null).putValue(10);
    sht.range("B2", null).putValue(5);
    sht.range("C2", null).putValue(3);
    rng := sht.range("A1", "C2");           // Select cells
    chrts := xl.charts;                     // Add a chart
    chrts.add(null, null, null, null);
    chrt := xl.activeChart;
    // Start chart wizard
    chrt.chartWizard(rng,                   // source
                     xl.Xl3DPie,            // gallery
                     7,                     // format
                     xl.XlRows,             // plotBy
                     1,                     // categoryLabels
                     0,                     // seriesLabels
                     2,                     // hasLegend
                     "Jade Example",        // title
                     null,                  // categoryTitle
                     null,                  // valueTitle
                     null);                 // xtraTitle
    // Output chart
    chrt.printOut(null,                     // first page
                  null,                     // last page
                  null,                     // copies
                  null,                     // preview
                  null,                     // printer
                  null,                     // print to file
                  null);                    // collate
epilog
    if xl <> null then
        xl.activeWorkbook.saved := true;    // Don't ask to save!
        xl.quit;                            // Close down Excel
        delete xl;
    endif;
end;
```

Using Automation Events

To register your interest in a specific set of automation events, you must first define a reference to the set of automation events in which you are interested. When you have registered your interest, these event methods are then triggered in the same way as event methods for JADE controls.

To set up an automation event handler

1. Select the **Add Reference** command from the Properties menu in the Class Browser to create a property that is a reference to an event in the **IDispatch** class.

2. Use the Define Reference dialog that is then displayed to specify your event reference property, as you would for any other reference property. (For details, see "Adding a Reference Property", in Chapter 4 of the *JADE Development Environment User's Guide*.)

   To summarize:

   a. In the **Name** text box, specify the name of the event reference that you want to define.

      The maximum length of the property name varies and it cannot be greater than the length of the longest method name of the event interface. The name cannot be the same as the name of any existing property or non-mapping method in the class, its superclasses, or subclasses.

   b. In the **Type** combo box, perform one of the following actions.

      - Select the required type in the Type list.

        The types that are listed are the subclasses of the **IDispatch** class that correspond to each ActiveX event **dispInterface** defined in the type library.

      - Specify the first character or the first few characters of the type in the text box and then select the required type from the Type list.

        The selected type is then displayed in the **Type** combo box.

   c. Check the **Subschema Hidden** check box if you want to specify that the event reference is available only in the local schema. This check box is unchecked by default; that is, the event reference is available in subschemas.

      Subschema-hidden event references that are not available for use in any subschemas cannot be referenced by subschema code and they are not displayed in subschema Class Browsers. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1 of the *JADE Development Environment User's Guide*.

   When you have defined an event reference and you then select it in the Properties List of the Class Browser, the Methods List is populated with disabled methods that correspond to the methods of the event interface.

3. To add an automation event method, from the Methods List select the disabled method name that corresponds to the event in which you are interested.

   The editor pane then displays the method template with a signature that matches that of the automation event method. As it also marks your new method as having the same signature as the selected event and checks it at compile time, you cannot change the signature of your event method.

4. Unlike control and form events, you must register interest in an event set. The events are not triggered until the event set has been registered by calling the **ActiveXAutomation** class **beginNotifyAutomationEvent** method, which has the following signature.

```
beginNotifyAutomationEvent(receiver: Object; eventClassRefName: String);
```

The **receiver** parameter specifies the object to which the event is sent (that is, that receives the message) and the **eventClassRefName** parameter specifies the name of the event reference property that you defined in the Define Event Reference dialog.

The methods defined by the event reference are then executed each time its corresponding automation event occurs. This event notification continues until the JADE automation object is deleted or until the **endNotifyAutomationEvent** method is called, which has the following signature.

```
endNotifyAutomationEvent(receiver: Object; eventClassRefName: String);
```

Caution There may be an impact on performance, particularly in JADE thin client mode or on a slow communications link, if you register for large numbers of automation events or events that are triggered often; for example, a cell **change** event in the Excel automation type library. (For details about achieving maximum performance in the JADE thin client mode of operation, see "JADE Thin Client Performance Considerations", in Appendix A of the *JADE Thin Client Guide*.)

Using Controls as Automation Objects

An ActiveX control is a standard automation object with interfaces to handle the Graphical User Interface (GUI) requirements. As a control can be defined so that it does not require a form in which to live, some ActiveX controls can be treated as standard automation objects. To use such a control outside a form, the **makeAutomationObject** method is called to create the ActiveX object after an instance of the JADE control has been created but before that control is added to a form. When this has occurred, the control can then be treated as any other automation object.

The following example assumes that you have imported the Microsoft **SysInfo** control (that is, **sysInfo.ocx**) as an ActiveX control. This example creates a JADE control instance and calls the **makeAutomationObject** method instead of adding the control to a form.

```
createSysInfoAsAutoObject();
vars
    x : SysInfo;
begin
    create x transient;
    x.makeAutomationObject;
    write x.oSVersion;
    write x.oSBuild;
    delete x;
end;
```

ActiveX Class Interfaces

Instances of ActiveX interface classes are not created or deleted by user logic. Instances of the classes are created by JADE at run time when an ActiveX object supplies an interface pointer and they are deleted when the ActiveX object that uses them is deleted.

Using Standard Classes

As most ActiveX type libraries include the OLE Automation library, which contains standard interfaces, the **STDOLE2**.**TLB** OLE Automation library is preloaded into the JADE RootSchema. You can create instances of the OLE Automation object and then manipulate them in the same way you can any other imported ActiveX object.

The preloaded OLE Automation type library created the standard font and picture **JadeAutoFont** and **JadeAutoPicture** objects as subclasses of **OLE_Automation** in the **ActiveXAutomation** superclass and their corresponding **IJadeAutoFont**, **IJadeAutoFontEvents**, and **IJadeAutoPicture** interface subclasses of the **IDispatch** class.

The following example shows the creation of a font object and the setting of font properties.

```
createAFont();
vars
    autoFont : JadeAutoFont;
begin
    create autoFont;
    write autoFont.bold;
    autoFont.bold := true;
    write autoFont.bold;
epilog
    delete autoFont;
end;
```

The JADE methods that are provided for the **JadeAutoPicture** and **IJadeAutoPicture** classes preloaded into the **RootSchema** with the OLE Automation object are summarized in the following table.

Method	Description
loadPicture(fileName: String): IJadeAutoPicture;	Creates a picture object from an external file
makePicture(binary: Binary): IJadeAutoPicture;	Creates a picture object from a JADE binary
loadPicture(fileName: String);	Saves the image of a picture to the specified external file

Editing ActiveX Methods That Return a StringArray

JADE ActiveX maps a COM string array into the JADE **StringArray** type. The maximum length of a string that can be inserted into a **StringArray** is 62 characters. An exception (1035 - String too long) is raised if you attempt to add a string with greater than 62 characters to the ActiveX array.

If you edit the methods generated by the ActiveX import, you can use the **HugeStringArray** type in place of the **StringArray** type. This allows strings with a length up to 2047 characters before an exception is raised.

To make use of a **HugeStringArray**, you need to tell the ActiveX interface to use them. In the following example, an ActiveX object implements a member method called **ListOfThings** that returns an array of strings. The JADE method generated by the import for such a member is as follows.

```
listOfThings():StringArray updating, clientExecution;
begin
    return _jadeActiveXInvoke("ListOfThings" 1,'0.8200').StringArray;
end;
```

To enable the method to handle a **HugeStringArray**:

- Change method return type
- Pass the HugeStringArray class number to the ActiveX interface in the **parameter description** argument to the **_ jadeActiveXInvoke** method

After making the changes, the **listOfThings** method is as follows.

```
listOfThings():HugeStringArray updating, clientExecution;
begin
```

```
    return _jadeActiveXInvoke("ListOfThings" 1,
                    '0.8200.'&HugeStringArray.number.String).HugeStringArray;
end;
```

As the class number of the **HugeStringArray** class is 429, the method can be simplified as follows.

```
listOfThings():HugeStringArray updating, clientExecution;
begin
   return _jadeActiveXInvoke("ListOfThings" 1,'0.8200.429').HugeStringArray;
end;
```

Note You may need to make similar changes in two places; in the **ActiveXControl** or **ActiveXAutomation** subclass, and also in the corresponding **IDispatch** interface subclass.

ActiveX Default Values and Considerations

When using ActiveX type libraries, you should be aware of the functionality and performance considerations described in the following subsections.

Default Names

The ActiveX Import Wizard generates default names for all components that need to be named. You can override these default names for classes (that is, for controls, automation objects, and interfaces), methods, and properties.

These default names are based on the name of the corresponding component in the imported type library. However, as the COM naming and the JADE naming rules are not the same, name transformation takes place. The rules for this transformation are as follows.

- If the ActiveX name starts with an invalid JADE naming character (for example, the underscore character), the name is prefixed by **I** for an interface class, **O** for an object class, **m** for a method, and **p** for a property and method parameter. For example, the **_WorkbookEvents** interface name becomes **I_WorkbookEvents**.
- The first letter of a class name is an uppercase character and the first letter of methods and properties is a lowercase character.
- If the ActiveX name contains any invalid JADE naming characters, they are converted to an underscore character.
- If the name is longer than 100 characters, it is truncated to 100 characters.
- If the generated name is a duplicate name, a counter is appended to that name to make it unique. The counter is an underscore character followed by an integer value, which is incremented until the name becomes unique. For example, the method **IsKindOf** becomes **isKindOf_1**.

Data Types

The **invoke** method of the **IDispatch** class uses the **VARIANT** data type to pass properties and return values and method parameters. These variant types are mapped to JADE equivalents for the properties and methods of the generated ActiveX interface classes.

The following table lists the JADE primitive types that are used for each of the **VARIANT** types.

Variant Type	Description	C++ Type	JADE Primitive Type
VT_I2	2 byte integer	short	Integer

Variant Type	Description	C++ Type	JADE Primitive Type
VT_I4	4 byte integer	long	Integer
VT_UI1	Unsigned char	unsigned char	Character
VT_UI2	Unsigned 2 byte integer	unsigned short	Integer
VT_UI4	Unsigned 4 byte integer	unsigned long	Integer
VT_INT	Machine integer	int	Integer
VT_UINT	Machine unsigned integer	unsigned int	Integer
VT_R4	4 byte real	float	Real
VT_R8	8 byte real	double	Real
VT_CY	Currency	CY (__int64)	Decimal
VT_18	8 byte integer	_int64	Decimal
VT_U18	Unsigned 8 byte integer	unsigned _int64	Decimal
VT_BOOL	Boolean	bool	Boolean
VT_ERROR	Error code	long	Integer
VT_DATE	Date	DATE	TimeStamp
VT_BSTR	Binary string	BSTR	String
VT_ARRAY	Safearray pointer	SAFEARRAY	Array
VT_VARIANT	Variant pointer	VARIANT	Any
VT_UNKNOWN	IUnknown pointer	IUnknown	IUnknown
VT_DISPATCH	IDispatch pointer	IDispatch	IDispatch
VT_Decimal	Decimal	No C++ equivalent	Decimal

Caution Array types are not supported. An interface method defined with a parameter or return value that is an array type is generated but an exception is raised when that method is executed.

Component Categories

An ActiveX control can specify a list of component categories. These component categories are used to specify the type of environment in which this control is intended to work. Categories are a suggestion of the preferred environment only, and are not a requirement.

The ActiveX control type library import process imports an ActiveX control regardless of any component category specification.

Optional Parameters

ActiveX methods support optional parameters but the methods generated by the library import process do not support optional parameters.

If, however, the ActiveX documentation states that a particular value is optional and you do not want to specify a value for that parameter in the method call, you can set the parameter to the JADE **null** value. This is then passed to the ActiveX object as an empty (undefined) optional parameter.

.NET Assemblies

A .NET assembly is a library file (**.dll**) or executable file (**.exe**) with additional .NET header information (the manifest) that describes the class methods and properties in the assembly.

Note In this release, JADE supports only in-process assemblies; that is, assemblies defined in Dynamic Link Libraries (DLLs).

When a .NET assembly is imported into JADE, the classes along with their members (properties, methods, and constants) can be selected or excluded and given appropriate names for the corresponding JADE entities. For details, see "Maintaining .NET Objects", in Chapter 16 of the *JADE Development Environment User's Guide*.

Location of .NET Assemblies

The location of .NET assemblies is managed by .NET run time and not by JADE. Assemblies that are shared with many different applications are usually loaded into the Global Assembly Cache (GAC), which resides in the Windows directory.

Assemblies are normally loaded into the GAC by the components installer and must meet certain requirements (such as having a strong name). It is also possible to create a private cache for the assembly or to store it in a specified directory.

The common location for an assembly is the GAC or the JADE **bin** directory (or a subdirectory). If you have imported a strongly named assembly, the first location checked is the GAC. If the assembly is not found in the GAC or it is a weakly named assembly, the directory of the executable (**jade.exe**) is checked.

You can use an XML configuration file in the same directory as the executable to specify an alternative location for an assembly. The name of the configuration file must be the same as that of the executable file but with the **.exe** extension replaced with **.config** (for example, **jade.config**).

The following example specifies that the .NET run time should check the **controls** directory, which is a subdirectory of the directory in which **jade.exe** is found.

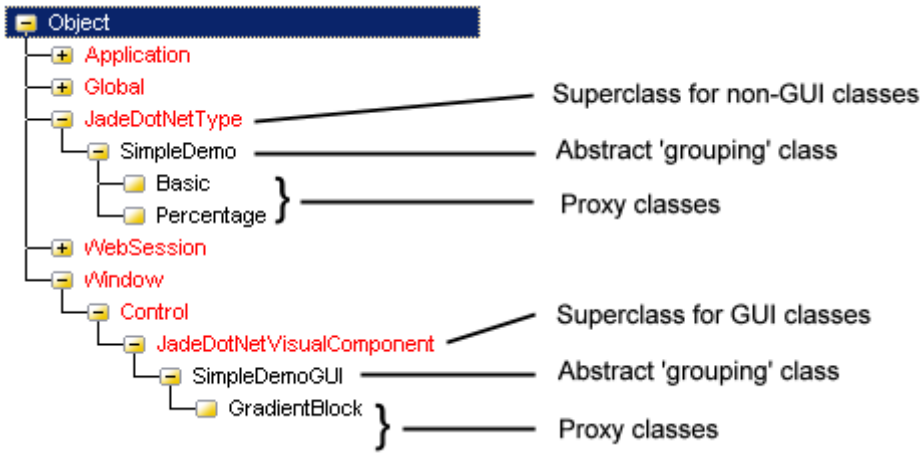
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <runtime>
        <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
            <probing privatePath="controls" />
        </assemblyBinding>
    </runtime>
</configuration>
```

How JADE Imports .NET Object Definitions

When you import .NET object definitions into JADE, the resulting classes depend on whether the object is a .NET control or a non-GUI component. For details, see the following subsection.

Abstract Grouping Classes

The following image shows the class structure that result from importing a .NET assembly.



In JADE, two abstract classes are created, which act as superclasses for the classes created to represent import types. GUI objects must be subclasses of the **Window** class, so one of these abstract classes is a subclass of the **JadeDotNetVisualComponent** class (a subclass of **Control**), the base class for all .NET GUI components.

All of the imported GUI components have a matching class defined for them under this abstract **Control** class. Likewise, an abstract class is created as a subclass of the **JadeDotNetType** class, which acts as the superclass of all imported non-GUI types.

For more details, see Chapter 16, "Importing ActiveX and .NET External Components", in the *JADE Development Environment User's Guide*.

.NET Default Values and Considerations

When using .NET assemblies, you should be aware of the functionality and performance considerations described in the following subsections.

Importing Into an ANSI JADE System

If the assembly being imported uses only ANSI characters for class and member names, the conversion from Unicode to ANSI is transparent. However, that is not the case if the names contain characters that cannot be represented in the current code page. For example, if a .NET class member with a name '员工工号' is imported into an ANSI JADE system with an English code page, the name displays as '**????**' and has a mapped JADE name of '____'.

Note JADE names consist of the characters **a** through **z**, **A** through **Z**, **0** through **9**, and the underscore character. During default name creation, characters not in this set are replaced by the underscore character.

There is also a problem with the generated property mapping methods and the generated methods, as the first parameter is the member name. For example, the generated mapping method for '员工工号' under Unicode or ANSI when running under a Chinese locale is as follows.

```
begin
    jadeDotNetProperty('员工工号', set_, value_);
end;
```

When running under ANSI with an English locale, the mapping method (which fails at run time) is as follows.

```
begin
    jadeDotNetProperty('????', set_, value_);
end;
```

If the .NET assembly has Unicode characters in class and member names, the Unicode version of JADE is recommended. The ANSI version of JADE can be used if the code page used for importing the assembly is the same as that used at run time and the code page can represent all of the characters used. If the code pages used for importing the assembly and at run time are different, exceptions regarding classes and or methods not found are likely to be raised.

Default Names

The .NET Import wizard generates default names for all components that need to be named. You can override these default names for classes (that is, for controls and non-GUI components), methods, and properties.

These default names are based on the name of the corresponding component in the imported assembly.

Data Types

Only .NET classes can be mapped to JADE objects. These classes can be a *reference* or *value* type (for example, a C# **'class'** or **'struct'**, or a C++ **'ref class'** or **'value class'**). Nullable .NET primitive types are mapped to the same JADE type as the non-nullable version of the type.

If a class is imported that does not have a definition in the assembly being imported, the class is mapped to the **JadeDotNetType** class.

The following table lists .NET types and the corresponding JADE types.

.NET Type	JADE Type	Conversion Notes
Boolean	Boolean	
Byte, SByte	Binary[1]	
Char	StringUtf8[1]	
DateTime	Timestamp	

DateTimeOffset	TimeStampOffset	The .NET **DateTimeOffset** type **DateTime** value represents a local time (UTC time + Offset) and the JADE **TimeStampOffset** type **TimeStamp** value represents a UTC time. Because of this, when converting from the JADE **TimeStampOffset** type to the .NET **DateTimeOffset** type, the **DateTimeOffset** is derived by applying the offset to the displayed **TimeStampOffset** type UTC time value; for example:
		`TimeStampOffset: 09 October 2018, 21:00:00 +1300`
		`DateTimeOffset: 10 October 2018, 10:00:00 +1300`

Decimal	Decimal	A .NET decimal variable can hold larger and smaller values than a JADE **Decimal** variable. An overflow exception is raised if a .NET decimal value is too large or too small for a JADE **Decimal** variable.

.NET Type	JADE Type	Conversion Notes
Double, Single	Real	A JADE **Real** variable can hold larger and smaller values than a .NET Single variable. An overflow exception is raised if a JADE **Real** value is too large or too small for a .NET Single variable.
Int16, Int32, UInt16, UInt32	Integer	A JADE **Integer** variable corresponds to a .NET Int32 variable. .NET Uint32 numbers are also mapped to JADE Integers, but if such a UInt32 contains a large number (top bit is set), JADE interprets this as a negative number. Similarly, a negative Integer becomes a large positive UInt32 .NET value.
Int64,UInt64	Integer64	See **Integer** to **UInt32** conversion notes, in the previous row of this table.
Object	Object	.NET object types are represented in JADE as an 'imported assembly' class. If JADE does not have a class to represent a particular .NET object, an instance of **JadeDotNetType** is used. Arrays of any type are also created as instances of **JadeDotNetType**.
String	StringUtf8	

Updating .NET Properties on Value Types

Unlike JADE, .NET has *reference* and *value* types (a reference type is a C# **'class'** or a C++/CLI **'ref class'** type, while a value type is a C# **'struct'** or a C++/CLI **'value class'** type). Both types, when imported from a .NET assembly, become JADE classes that are equivalent to reference types.

At run time, .NET value types are therefore converted to JADE reference types. This introduces a potential trap for the unwary. The problem is best explained by an example.

An imported .NET assembly generates the following JADE classes.

```
MyRectangle, which has properties x, y, width, height, and so on
(In .NET, MyRectangle is a value type)

MyClass, which has a property called 'rect' of type MyRectangle
(In .NET, MyClass is a reference type)
```

The following JADE logic updates the **rect** property of a **MyClass** instance in the expected way.

```
vars
    c: MyClass;
    r, r2: MyRectangle;
begin
    // create object
    create c;
    c.createDotNetObject();
    // set rectangle value
    r := c.rect;
    r.x := 10;
    c.rect := r;
    // refetch rect and output current value
    r2 := c.rect;
    write r2.x;                                    // output '10' as expected
end;
```

The following similar JADE logic fails to update the **rect** property of a **MyClass** instance.

```
vars
    c: MyClass;
    r, r2: MyRectangle;
begin
    // create object
    create c;
    c.createDotNetObject();
    // set rectangle value
    c.rect.x := 10;                                  // does not update c
    // refetch rect and output current value
    r2 := c.rect;
    write r2.x;                                      // outputs '0'
end;
```

The reason the **c.rect.x := 10;** instruction fails to update **c** is that a temporary reference is created to represent the **c.rect** value type. It is the temporary reference that is updated with the new value for the **x** property. Unlike the first coding example, nothing is ever done with the temporary reference, so the original instance **c** remains unchanged.

.NET-Related JADE Modules

The JADE modules that are loaded when .NET components are accessed are listed in the following table.

JADE Module	Description
jadedotnetthin	Communicates with the .NET run time. Located where the .NET object is run. For a presentation client running a .NET control, this is the presentation client. For a non-GUI .NET component, this is the presentation client unless the value of the **usePresentationClient** property is **false** when it must be on the application server. In thin client mode, you could have **jadedotnetthin.dll** on the presentation client supporting controls and **jadedotnetthin.dll** on the application server supporting components run from the application server if the value of the **usePresentationClient** property is **false**.
jadedotnet	Communicates with **jadedotnetthin.dll** and calls into JADE logic for events, exceptions, and so on. Located on application servers and standard clients.
jadedotnetdesignerloader	Used by the JADE Painter to support .NET controls that have a designer (usually a form that enables you to set the properties of the control). Located on standard clients and presentation clients.
jadewpf	Displays .NET controls in JADE Painter and on runtime forms using Windows Presentation Foundation (WPF). Located on standard clients and presentation clients.

Using .NET Components

Proxy classes are created in JADE to access the imported .NET components. The location of the proxy class depends on whether the component is GUI or non-GUI; that is, whether the imported component is a .NET control or not. See "Abstract Grouping Classes", earlier in this chapter.

Non-GUI .NET Components

The classes created under the **JadeDotNetType** class are proxy classes for their corresponding .NET classes. By using the proxy object, you gain access to the .NET object. Therefore, to access the actual .NET object you must first create an instance of the JADE class. This JADE instance can then be used to create the .NET object, by using the **createDotNetObject** method, which in turn causes the .NET run time to create the instance of the corresponding .NET object. Generated methods on the JADE proxy class can then be used to access corresponding members on the actual .NET object.

Before creating the .NET object, the **usePresentationClient** property can be set to specify where the component should be run. (By default, all components run on the presentation client.)

The following method shows how the generated classes and methods are used.

```
vars
    b, b2 : Basic;                          // Subclass of JadeDotNetType
begin
    // Create a JADE instance of the class representing the .NET object
    create b transient;
    // Optionally, specify where the actual .NET object is created
    b.usePresentationClient := false;
    // Request the creation of the corresponding .NET object
    b.createDotNetObject;
    // Use methods and properties on the JADE instance
    // to access members of the .NET object
    write b.integer;
    b2 := b.anotherOne;
epilog
    delete b;
    delete b2;
end;
```

Event handling on components is a little more complex. By default, component events are not passed onto JADE. To start receiving such events, a call is made to the **beginEventNotification** method if the **JadeDotNetType** class, which takes the following parameters.

- The first parameter indicates to which object the event should be delivered.
- The second parameter specifies a reference property on the receiver, of the type that contains the events.

Events are implemented in the Class Browser, by clicking on the reference property in the receiver class, which displays all of the events available in the method pane. These events are shown grayed out until you add code to that event. This is identical to how you code an event on a control in a form; that is, with the form selected, click the control reference, select the methods to implement (**paint** or **click**), and then add your event logic.

.NET Helper Methods

When you import a .NET assembly, JADE generates classes for each class that is selected. However, an assembly can also reference other assemblies (or use types defined in other assemblies). For example, a control assembly (say **myControls.dll**) could refer to .NET **Color**, **Font**, and **Image** objects that are defined in the **System.Drawing.dll** assembly. If these types have not been imported, JADE creates instances of the **JadeDotNetType** class to represent them.

To enable the use of **Color**, **Font**, and **Image** .NET objects without having to import the **System.Drawing.dll** assembly, the *helper* methods listed in the following table are provided to access properties of **Color**, **Font**, and **Image** objects from a **JadeDotNetType** instance.

JadeDotNetType Helper Method	Returns …
getColor	A 32-bit number, representing the ARGB color value
getFont	The font properties of the Font object
getPicture	A binary, representing the image

An exception is raised if these methods are called on objects that do not represent the appropriate type (for example, the **getColor** can only be used on a .NET **Color** object).

The *helper* methods listed in the following table are provided as an alternative to the **createDotNetObject** method to create **Color**, **Font**, and **Image** objects from a **JadeDotNetType** instance.

JadeDotNetType Helper Method	Creates a .NET …
createColor	Color object using the supplied ARGB color value
createFont	Font object from the supplied font properties
createPicture	Image object from the supplied binary value

An exception is raised if the .NET object cannot be created for any reason.

.NET Controls

An imported .NET control is a GUI .NET component and is added to the Control palette of the JADE Painter. It cannot be distinguished from the standard JADE-supplied controls.

As with non-GUI components, creating an instance of the JADE control object does not create the corresponding .NET object. The .NET object is created when the control is dropped onto a form in the JADE Painter, when the form (with a .NET control) is loaded at run time, or when the **addControl** method of the **Form** class is used to add a dynamically created control.

Some controls include *designers*, which are usually forms that enable properties of the control to be set. These are particularly useful when the control is made up of a number of subcontrols or when properties have dependencies on other properties. The first time a control with a designer is dropped onto a form in the JADE Painter, the designer form is displayed. When the designer form is closed the control, modified by the designer, is shown on the form being painted.

When a control with a designer has been displayed in the JADE Painter, the designer can usually be reactivated using the context menu. The context menu in the JADE Painter, when activated over a .NET control, can include up to 10 extra entries corresponding to the first 10 available *designer verbs*. These are options defined in the designer that perform various designer functions, including re-activating the designer, or altering the control in some way.

Chapter 5 XML Metadata Interchange (XMI) Support

This chapter contains the following topics.

Overview

JADE supports XML Metadata Interchange (XMI), specifically XMI 2.1 as generated by the Sparx Systems Enterprise Architect (EA) tool.

XMI is an Object Management Group (OMG) standard for exchanging metadata information using Extensible Markup Language (XML).

XMI support is provided by the **XMI Import** command in the JADE development environment File menu, which accesses the Extract Schema dialog that enables you to select the name and location of the XMI file to extract and the location of the output files. This generates schema files from an XMI file (XMI 2.1 from the EA tool).

Supported Version of XMI Files

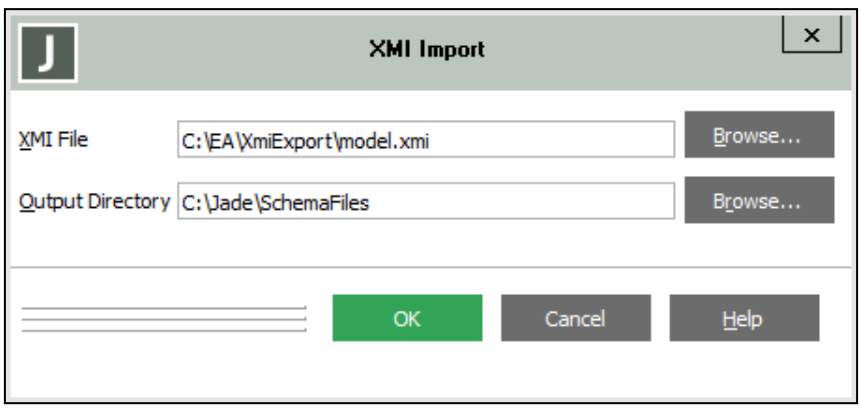
The only version that is supported is an XMI 2.1/UML 2.1 file, which is exported from Enterprise Architect.

Generating Schema Files from an XMI File

To generate schema files from an XMI file

1. Select the **XMI Import** command from the File menu in the JADE development environment.

   The XMI Import dialog, shown in the following image, is then displayed.



2. In the **XMI File** text box, specify the name of the XMI file extracted from Enterprise Architect.

If you are unsure of your file name or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file.

3. In the **Output Directory** text box, enter the name of the directory that will contain the **scm** and **ddb** or .**ddx** files generated by the application. If you are unsure of the location, click the **Browse** button.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

You can then load the generated schema files in the target system.

Note Minimal validation is done during the import process. Errors are detected when the schema is loaded and displayed to the user for correction.

Enterprise Architect 7 and JADE Mappings

The correspondence between JADE entities and their corresponding entities in Enterprise Architect 7 are described in the following sections.

In each section, the focus is on a specific entity in JADE; for example, a class or a method. The JADE data relating to that element is presented in the first column of a table with the Enterprise Architecture equivalent in the second column.

JADE Schema Maps to Enterprise Architecture Package

A JADE schema maps to an Enterprise Architecture package with the stereotype **JadeSchema**, as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Superschema	Dependency Connector Target = superschema	RootSchema
Default Map File	Tagged Value tag =JadeMapFile value = *<file-name>*	
Forms Management Style	Tagged Value tag = JadeFormsManagement	0
	■ value = 0 Multi-multi	
	■ value = 1 Single-single	
	■ value = 2 Single-multi	
Text	Notes	

JADE Class Maps to Enterprise Architecture Class

A JADE class maps to an Enterprise Architecture class, as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Superclass	Generalization Connector Target = superclass	Object
Map File	Tagged Value tag = JadeMapFile value = *<map-file-name>*	<none>
Access	Scope	public
Type	Abstract (if class is abstract)	real

JADE	Enterprise Architecture 7	Default Value
Persistence	Persistence	persistent
SubschemaHidden	Tagged Value tag = JadeSubschemaHidden value = true | false	false
SubschemaFinal	Tagged Value tag = JadeSubschemaFinal value = true | false	false
Final	Tagged Value tag = JadeFinal value = true | false	false
Instance Volatility	Tagged Value tag = JadeInstanceVolatility value = volatile | stable | frozen	volatile
Class Lifetimes	Constraint classLifetimes value = P | S | T | P/S|P/T|S/T|P/S/T	P/S/T
Subclass Lifetimes	Constraint subclassLifetimes value = P | S | T | P/S|P/T|S/T|P/S/T	P/S/T
Text	Notes	

Additional Data for JADE Collection Classes

The properties of JADE collection classes are mapped as follows.

JADE	Enterprise Architecture 7	Default Value
Membership	defaultCollectionClass	
Expected Population	Tagged Value tag = JadeExpectedPopulation, value is an integer	0
Entries Per Block	Tagged Value tag = JadeEntriesPerBlock, value is an integer	0
Exclusive Instances Map File	Tagged Value tag = JadeExclusiveInstanceMapFile, value is name of map file	Same file as collection owner
Membership Length	Required for Array subclasses of String, Binary, or Decimal membership	30 (String), 30 (Binary), 12 (Decimal)
	Constraint = membershipLength, value is an integer	
Membership Scale Factor	Required for Array subclasses of Decimal	0
	Constraint = membershipScaleFactor, value is an integer	

Additional Data for JADE Dictionary Classes

The properties of JADE dictionary classes are mapped as follows.

JADE	Enterprise Architecture 7	Default Value
Duplicates Allowed	Constraint = duplicatesAllowed, value is true or false	false
Load Pattern	Tagged Value tag = JadeLoadPattern, value = random | sequential	random

If the **JadeLoadPattern** tag value is **random**, the load factor is 66 percent; if the tag value is **sequential**, it is 98 percent.

Additional Data for JADE External Key Dictionary Classes

The properties of JADE external key dictionary classes are mapped as follows.

JADE	Enterprise Architecture 7	Default Value
Key Name	Attribute name = key name, stereotype = JadeExternalKey	
Key Length	Constraint = keyLength for String, Binary or Decimal key value is an integer	
Key Scale Factor	Constraint = keyScaleFactor, value is an integer	
Key Sequence	Constraint = keySequence, value = ascending | descending	ascending
Key Case Sensitive	Constraint = keyCaseSensitive, value = true | false	false
Key Sort Order	Constraint = keySortOrder, value = Binary or Locale id	Binary

Additional Data for JADE Member Key Dictionary Classes

The properties of JADE member key dictionary classes are mapped as follows.

JADE	Enterprise Architecture 7	Default Value
Key Name	Attribute name = attribute name, stereotype = JadeKey	
Key Sequence	Constraint = keySequence, value = ascending | descending	ascending
Key Case Sensitive	Constraint = keyCaseSensitive, value = true | false	false
Key Sort Order	Constraint = keySortOrder, value = Binary or locale id	Binary

JADE Property Maps to Enterprise Architecture Attribute

A JADE property maps to an Enterprise Architecture attribute, as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Type	Type (Must correspond to the JADE primitive types)	
Access	Scope Private = Read Only	Read Only
Length	Constraint = length, value is an integer	
Scale Factor	Constraint = scaleFactor, value is an integer	
Virtual	Derived	
Subschema Hidden	Tagged Value tag = JadeSubschemaHidden value = true | false	false
Allow Transient to Persistent	Tagged Value tag = JadeAllowTransToPersist value = true | false	false
Inverse Not Required	Tagged Value tag = JadeInverseNotRequired value = true | false	false
Constraint	Constraints	

JADE	Enterprise Architecture 7	Default Value
Text	Notes	
Relationship	Use Aggregation Connection Peer-to-Peer Aggregation = shared Parent-Child Aggregation = composite (on parent)	
Update Mode	Tagged Value tag = JadeUpdateMode, value = manual | auto	

JADE Method Maps to Enterprise Architecture Operation

A JADE method maps to an Enterprise Architecture operation, as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
JADE Method	No stereotype	
Condition	Stereotype = JadeCondition	
Mapping Method	Stereotype = JadeMappingMethod	
External Method	Stereotype = JadeExternalMethod	
	Alias specifies entry point and library: is *entry-point* in *library-name*	
Updating	Const checked = not updating	false
Abstract	Abstract	false
Access	Scope (only Public and Protected)	public
Subschema Hidden	Static checked = hidden	false
Final	Tagged Value tag = JadeFinal, value = final | ssfinal | sscopyfinal	
Execution Location	Tagged Value tag = JadeExecutionLocation, value = default | server | client	
Text	Notes	

JADE Constant Maps to Enterprise Architecture Constant Attribute

A JADE constant maps to an Enterprise Architecture constant attribute, as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Type	Type	
Definition	Initial Value	
Text	Notes	

The value for a constant of type **String** must be enclosed in double quotation marks (**""**).

If a constant is defined in terms of another constant, the definition of the other constant must occur earlier in the XMI file.

JADE Exported Package Maps to Enterprise Architecture Package

A JADE exported package maps to an Enterprise Architecture package with the stereotype **JadePackage** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Application	Tagged Value tag = JadePackageApplication, value = *application-name*	
Description	Notes	

JADE Exported Class Maps to Enterprise Architecture Class

A JADE exported class maps to an Enterprise Architecture class with the stereotype **JadeExportedClass** as follows.

JADE	Enterprise Architecture 7	Default Value
Reference	Connector = Dependency to the related class with a stereotype of import	
Persistence	Persistence	Same as related class
Class Lifetimes	Constraint classLifetimes value = P | S | T | P/S|P/T|S/T|P/S/T	Same as related class

JADE Exported Property Maps to Enterprise Architecture Attribute

A JADE exported property maps to an Enterprise Architecture attribute with the stereotype **JadeExportedProperty** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Access	Scope private = read only, public = public	Same as related property

JADE Exported Method Maps to Enterprise Architecture Operation

A JADE exported method maps to an Enterprise Architecture operation with the stereotype **JadeExportedMethod** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	

JADE Exported Constant Maps to Enterprise Architecture Attribute

A JADE exported constant maps to an Enterprise Architecture attribute with the stereotype **JadeExportedConstant, const** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	

JADE Imported Package Maps to Enterprise Architecture Package

A JADE imported package maps to an Enterprise Architecture package with the stereotype **JadeImportedPackage** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Connector	PackageImport, stereotype = import	
Description	Notes	

JADE Interface Maps to Enterprise Architecture Interface

A JADE interface maps to an Enterprise Architecture interface with the stereotype **Interface** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Extends	Connector = Generalization	
Text	Notes	
Implementors	Connector = Realization	

JADE Interface Method Maps to Enterprise Architecture Operation of Interface

A method of a JADE interface maps to an Enterprise Architecture operation of an interface as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Text	Notes	

JADE Interface Constant Maps to Enterprise Architecture Operation of Interface

A constant of a JADE interface maps to an Enterprise Architecture constant attribute of an interface as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Type	Type	
Definition	Initial Value	
Text	Notes	

JADE Constant Category Maps to Enterprise Architecture Class

A JADE global constants category maps to an Enterprise Architecture class with the stereotype **JadeGlobalConstantCategory** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Text	Notes	

JADE Global Constant Maps to Enterprise Architecture Attribute of the Constant Category Class

A JADE global constant maps to an Enterprise Architecture constant attribute of the **ConstantCategory** class as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Type	Type	
Definition	Initial Value	
Text	Notes	

JADE Library Maps to Enterprise Architecture Class

A JADE library maps to an Enterprise Architecture class with the stereotype **JadeLibrary** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	

JADE External Function Maps to Enterprise Architecture Operation of the Library Class

A JADE external function maps to an Enterprise Architecture operation of the **Library** class as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Entry point	Alias	
Text	Notes	

JADE Locales Map to Enterprise Architecture Class

JADE locales map to an Enterprise Architecture class called **Locales** with the stereotype **JadeLocale**.

Add attributes to this class for each locale defined for the schema as follows.

JADE	Enterprise Architecture 7	Default Value
Locale Name	Name (for example, 5129)	
Locale Description	Initial (for example, "English (United Kingdom)")	
Default	Const is checked	
Clone Of	Constraint name=cloneOf Constraint value=<locale name>	

JADE Translatable String Maps to Enterprise Architecture Attribute of the Locale Class

A JADE translatable string maps to an attribute in an Enterprise Architecture class with a name of the locale id and a stereotype of **JadeTranslatableString**. Add attributes to this class for each translatable string defined for the locale as follows.

JADE	Enterprise Architecture 7	Default Value
TS Name	Name	
Value	Initial Value	

JADE Locale Format Maps to Enterprise Architecture Attribute of the Locale Class

A JADE locale format maps to a constant attribute of an Enterprise Architecture class with a name of the locale format and a stereotype of **JadeLocaleFormat**. Valid locale format names are **DateFormat**, **NumberFormat**, **CurrencyFormat**, and **TimeFormat**.

Add attributes to the class for each locale format defined for this locale.

JADE	Enterprise Architecture 7	Default Value
Format Name	Name	
Value	Initial Value	

JADE Application Maps to Enterprise Architecture Class

A JADE application maps to an Enterprise Architecture class with the stereotype **JadeApplication** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Connector	Dependency to JadePackage	

JADE Subschema Copy Class Maps to Enterprise Architecture Class

A JADE application maps to an Enterprise Architecture class with the stereotype **JadeSubschemaCopy** as follows.

JADE	Enterprise Architecture 7	Default Value
Name	Name	
Connector	Dependency	

Appendix A ODBC Reserved Words

This appendix covers the ODBC reserved words, listed in the following table.

ALL	AND	AS	ASC
BY	CREATE	DELETE	DESC
DISTINCT	DROP	FROM	INNER
INSERT	JOIN	LEFT	LIKE
NOT	ON	OR	ORDER
OUTER	RIGHT	SELECT	UNION
UPDATE	WHERE		

When you create a relational view of your JADE database by using the Relational Views Wizard, some words should not be used as Table or Column names if the configuration setting for ODBC reserved words that the JADE ODBC driver will use is **Upper and Lower**. (For details about minimizing ODBC reserved word conflicts by converting the ODBC reserved words to an initial capital letter, see "Configuring a JADE ODBC Driver", in Chapter 2.)

Appendix B External Functions

This appendix covers the JADE external functions declared in the RootSchema **jomos** external function library that call Windows library functions as defined in the Microsoft Developer Network (MSDN). These external functions are declared to avoid having to declare ANSI- and Unicode-specific definitions for some commonly called Windows library functions; for example, **josShellExecute** calls **ShellExecuteA** in an ANSI environment and **ShellExecuteW** in a Unicode environment.

The following table maps functions defined in the RootSchema **jomos** external function library to the corresponding MSDN function.

JADE External Function	Windows Function	Description
josCreateDirectory	CreateDirectory	Creates a directory that inherits information from other directories (security attributes default to null)
josDeleteDirectory	RemoveDirectory	Deletes an existing empty directory
josFileAccess	GetFileAttributes	Retrieves attributes for a specified file or directory (for details, see the paragraph that follows this table)
josFileCopy	CopyFile	Copies an existing file to a new file
josFileDelete	DeleteFile	Deletes an existing file
josGetKeyState	GetKeyState	Retrieves the status of the specified key; that is, whether the key is up, down, or toggled
josGetLastError	GetLastError	Retrieves the last error code value of the calling thread
josShellExecute	ShellExecute	Opens or prints the specified file; for example, to start another program under Microsoft Windows

In addition to these functions, the **josValidateDirectory** external function is declared, which validates that the specified name is a directory.

The **josFileAccess** function checks the specified file to determine if it exists and whether it can be read, written, or executed. This pointer is a call for files only; if you call it with a directory name, **F_FILE_NOT_FOUND** is returned. (Call the **josValidateDirectory** function to validate a directory.) The **josFileAccess** parameters are:

- **fileName**, which is a string that specifies the name of the file.
- **aMode**, which specifies a bit pattern constructed as follows.
  - 06, which checks for read and write permission
  - 04, which checks for read permission
  - 02, which checks for write permission
  - 01, which checks for execute permission
  - 00, which checks for the existence of the specified file
- **log**, which specifies if invalid path exceptions are logged to **jommsg.log**.

The **josFileAccess** and **josValidateDirectory** calls return zero (**0**) if successful or they return a JADE file-handling error in the range 5000 through 5099.

For details about:

- Adding an external function to a class, see "Defining External Functions", in Chapter 8 of the *JADE Development Environment User's Guide*
- Browsing the RootSchema **jomos** external function library to obtain the JADE external function method signature, see "Adding an External Function, in Chapter 8 of the *JADE Development Environment User's Guide*
- External function calls, see "Using External Functions", in Chapter 1