EAServer 6.1 New Features Guide

Document ID: DC38032-01-0610-01

Last revised: December 2007

.NET client support

EAServer 6.1 includes .NET client support, and enables IIOP/IIOPS communication between:

- .NET C# clients and EAServer components
- PowerBuilder® WinForm applications and EAServer
- .NET and J2EE distributed objects

.NET client support includes data marshalling: data from multiple sources is collected, converted to a common format if necessary, and prepared to send over a network.

These components provide .NET client support:

- **NetCompiler** Generates C# stubs and helper classes for EAServer components. To invoke NetCompiler, run:

```
%DJC_HOME%\bin\netcc.bat [class name] | [ejbjar-name] | [name.jar]
```

where:

Option	Description
`class name`	A class that is either an interface or serializable: • If the class is an interface, NetCompiler generates stubs for it. • If the class is serializable, NetCompiler generates helper classes for data marshalling. If the class name begins with "java" or if the class is a primitive, it is ignored.
`ejbjar-name`	An EJB that is deployed in EAServer. NetCompiler generates C# stubs for both the home and remote interfaces.
`name.jar`	An EJB JAR that is deployed in EAServer. NetCompiler generates C# stubs for both the home and remote interfaces.

- **.NET runtime assemblies** Assembly files that provide runtime support to enable C# stubs to communicate with EAServer:
  - com.sybase.iiop.net.dll – for data marshalling, and for managing connections, SSL, and compression. The next two runtime assemblies depend on this one.
  - com.sybase.ejb.net.dll – enables invoking EJBs.
  - com.sybase.jms.net.dll – enables calling JMS.

  The .NET runtime assemblies are located in *%DJC_HOME%\lib*.

Developing and running a sample client

This section describes how to develop and run a .NET C# client application that communicates with an EAServer EJB component.

❖ **Creating and deploying an EJB**

1 Create an EJB called EchoEJB and add it to *EchoEJB.jar*.

2 To deploy the EJB, run:

```
deploy.bat EchoEJB.jar
```

❖ **Generating the C# stubs**

1 Verify that the C# compiler (*csc.exe*) is in your path.

2 Run NetCompiler:

```
netcc.bat ejbjar-echoejb
```

NetCompiler creates these C# source files in *%DJC_HOME%\genfiles\cs\src*:

- *echoejb.cs* – interface file, which must be included to call the EJB.
- *echoejb_Stub.cs* – stub file.
- *echoejbHelper.cs* – helper classes for data marshalling and narrowing.

The source files are packaged in the output assembly *%DJC_HOME%\deploy\assemblies\echoejb.client.dll*.

Note You can use the C# stubs that are compiled into assembly files from Visual Basic .NET clients.

❖ **Developing the C# sample client**

- Create a C# file called *EchoClient.cs*, and add this code:

```
using System;
using System.Data;
using System.Collections;
using System.Net;
using System.Net.Sockets;

using com.sybase.ejb.net;
using com.sybase.net;

public class EchoClient
{
   public static void Main()
   {
      EjbProvider ejbProvider = EjbProvider.GetInstance();
      ejbProvider.SetProviderURL("iiop://host:port");
      ejbProvider.SetUsername("admin@system");
      ejbProvider.SetPassword("sybase123");

      EjbConnection ejb = EjbConnection.GetInstance();
      EchoRemoteHome home = (EchoRemoteHome)
          ejb.LookupHome(typeof(EchoRemoteHome), "EchoEJB/EchoBean");
      EchoRemote echo = home.create();
      Console.WriteLine( echo.hello("World") );
   }
}
```

where:

Code	Description
`using com.sybase.net`	Imports the stub classes that were generated by running *netcc.bat*.
`EjbProvider`	Acts as a name service. The code defines the URL (machine name and IIOP port number), user name, and password.
`ejb.LookupHome`	Gets the home interface com.sybase.net.EchoRemoteHome, looks up the server's naming context, and gets a stub from the server using the connection information from EjbProvider. **Note** The code for EjbProvider and EjbConnection is in com.sybase.ejb.net.dll.
`home.create`	Gets the remote interface com.sybase.net.EchoRemote.
`echo.hello`	Calls the remote interface business method hello.

❖ **Compiling and running the C# sample client**

1 To compile *EchoClient.cs*, run:

```
csc EchoClient.cs /r:%DJC_HOME%\lib\com.sybase.iiop.net.dll
/r:%DJC_HOME%\lib\com.sybase.ejb.net.dll
/r:%DJC_HOME%\deploy\assemblies\echoejb.client.dll
```

Compiling the client creates *EchoClient.exe*, which you can run from the command line.

Note com.sybase.iiop.net.dll and com.sybase.ejb.net.dll are shipped with EAServer 6.1 in the *lib* subdirectory, and are the basic framework of the .NET client runtime support. echoejb.client.dll is created by *netcc.bat* when you generate the C# stubs for the EJB. The classes in echoejb.client.dll call methods in com.sybase.iiop.net.dll and com.sybase.ejb.net.dll.

2 Run:

```
EchoClient.exe
```

The program output is:

```
Hello World!
```

❖ **Troubleshooting**

- If you see the following error message when you run the client application, the necessary assembly files cannot be found:

```
Unhandled Exception: System.IO.FileNotFoundException: Could not load
file or assembly 'com.sybase.ejb.net, Version=0.0.0.0, Culture=neutral,
PublicKeyToken=null' or one of its dependencies. The system cannot find
the file specified. File name: 'com.sybase.ejb.net, Version=0.0.0.0,
Culture=neutral, PublicKeyToken=null' at EchoClient.Main()
```

To solve this problem, you can either:

- Copy the assembly files to the current directory, or
- Create an XML configuration file to identify the location of the assembly files:

  a In the directory that contains *EchoClient.exe*, create an XML file called *EchoClient.exe.config*, and add this code:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <probing privatePath="lib"/>
   </assemblyBinding>
 </runtime>
</configuration>
```

  b Copy these assembly files to the location identified by privatePath, which must be a subdirectory of the application's base directory:

  - com.sybase.iiop.net.dll
  - com.sybase.ejb.net.dll
  - com.sybase.jms.net.dll

Creating a .NET JMS sample client

.NET clients can use the Java Messaging System (JMS) to create, send, and receive messages. You can also use JMS from Visual Basic .NET clients. The clients can use either a point-to-point or publish/subscribe messaging model, as illustrated in the following sample client. The point-to-point messaging model delivers messages to specific queues. The publish/subscribe messaging model defines a topic and publishes messages that can be read by all subscribers to the topic.

This sample client acts as both the producer and consumer of JMS messages. In real applications, .NET clients can also send messages to, and receive messages from, Java applications. You can compile this client application by running *csc.exe*:

```
using System;
using System.Threading;
using System.Net;
using com.sybase.jms.net;

public class JmsTest
{
  // To get this test to pass, first run:
  //
  // set-password testuser testpass1

  public static void Main()
  {
    int retry = 3;
```

Create a JmsProvider instance, and set its URL, user name, and password properties:

```
JmsProvider jmsProvider = JmsProvider.GetInstance();
jmsProvider.SetProviderURL("iiop://" +
   Dns.GetHostName() + ":2000");
jmsProvider.SetUsername("testuser");
jmsProvider.SetPassword("testpass1");
```

Create a JmsConnection object to act as the gateway between EAServer and the .NET client:

```
JmsConnection jmsConn = jmsProvider.GetConnection();
```

Define the local-storage location for messages, which allows .NET clients to temporarily store messages locally, then send the messages when the clients connect to EAServer. This example uses the Windows registry for local storage. WorkOffline directs the client to work locally.

```
jmsConn.SetLocalStore(LocalRegistry.GetStore());
jmsConn.WorkOffline();
Console.WriteLine("*** working offline ***");
```

Create a text message:

```
Message msg1 =
   MessageUtil.GetTextMessage("QueueQueue");
```

Send delivers the message to MyQueue1. Since the client is working in offline mode, the message is stored locally:

```
string q1 = "MyQueue1";
jmsConn.Send(q1, msg1);
Console.WriteLine("<== sent message: " +
   msg1.body.text());

Thread.Sleep(3000);
```

Tell the client to work in online mode. Messages that are stored locally are sent to the remote server. The client can also receive messages from the remote server.

```
jmsConn.WorkOnline();
Console.WriteLine("*** working online ***");
Thread.Sleep(3000);
```

Calling Receive(q1, 1000) receives messages from MyQueue1 with a timeout value of one second. After the client receives a message, it calls Acknowledge to tell EAServer that the message was received and it is safe to delete it.

```
for (int i = 0; i < retry; i++)
{
   Message msg = jmsConn.Receive(q1, 1000);

   if (msg != null)
   {
      Console.WriteLine("==> received message :" +
        msg.body.text());
      jmsConn.Acknowledge(msg);
```

```
    }
}
```

To publish a text message, define the topic, and create the message:

```
String topic = "MyTopic";
Message msg2 =
MessageUtil.GetTextMessage("TopicTopic");
```

Set the client ID, which identifies the client application:

```
jmsConn.SetClientID("ccID");
string queue = jmsConn.GetClientID();
```

Call Subscribe to register a subscription to the topic. This example uses the client ID as the subscribing queue ID:

```
jmsConn.Subscribe(queue, topic);
```

Working in offline mode, publish the message (msg2), which is stored locally. When the client changes to online mode, any queue that is subscribed to the topic receives the message:

```
jmsConn.WorkOffline();
Console.WriteLine("*** working offline");

jmsConn.Publish(topic, msg2);
Console.WriteLine("<== published message: " +
   msg2.body.text());

Thread.Sleep(3000);
```

Change to work online. Receive and acknowledge messages with the topic MyTopic:

```
jmsConn.WorkOnline();
Console.WriteLine("*** working online");
Thread.Sleep(3000);

for (int i = 0; i < retry; i++)
{
   Message msg = jmsConn.Receive(queue, 1000);

   if (msg != null)
   {
      Console.WriteLine("==> received message from
```

```
            topic: " + msg.body.text());
        jmsConn.Acknowledge(msg);
    }
}
```

Call Unsubscribe to tell EAServer that this queue is no longer interested in receiving messages with the specified topic.

```
    msConn.Unsubscribe(queue);
  }
}
```

Datatype support

EAServer 6.1 supports marshalling for the following datatypes in .NET clients. Marshalling is supported for data passed either by reference or by value.

Datatype
boolean
byte
int
string
Array (one, two, or three dimensional)
Exception
Object Reference
Structure
Union

Table 1 describes the datatype mappings between IDL, Java, and C#.

Table 1: Datatype mappings

IDL datatype	Java datatype	C# datatype
boolean	boolean	System.Boolean
char	char	System.Char
octet	byte	System.Byte
short	short	System.Int16
long	int	System.Int32
long long	long	System.Int64
float	float	System.Single
double	double	System.Double
string	String	System.String

IDL datatype	Java datatype	C# datatype
BCD::Binary	byte	System.Byte[]
BCD::Decimal	java.math.BigDecimal	System.Decimal
BCD::Money	java.math.BigDecimal	System.Decimal
MJD::Date	java.util.Calendar	System.DateTime
MJD::Time	java.util.Calendar	System.DateTime
MJD::Timestamp	java.util.Calendar	System.DateTime
TabularResults::ResultSet	java.sql.ResultSet	System.Data.DataTable
TabularResults::ResultSets	java.sql.ResultSet[]	System.Data.DataTable[]
MyModule::MyException	MyModule.ejb.MyException	MyModule.ejb.MyException
MyModule::MyComp	MyModule.ejb.MyComp	MyModule.ejb.MyComp
MyModule::MyStruct	MyModule.ejb.MyStruct	MyModule.ejb.MyStruct
MyModule::MyUnion	MyModule.ejb.UnionName	MyModule.ejb.UnionName
MyModule::MySequence	MyModule.ejb.MyElement[]	MyModule.ejb.MyElement[]

Management Console Eclipse plug-in

In EAServer 6.1, the Management Console is available as an Eclipse 3.2 plug-in. The plug-in is installed automatically when you install EAServer and upgrade Eclipse. You can run the Management Console either in Eclipse or in a Web browser.

Single sign-on and SiteMinder support

Single sign-on (SSO) support allows clients to access multiple applications that require credentials by logging in only once.

EAServer 6.1 works with Computer Associates (formerly Netegrity) SiteMinder, which provides security features, such as X509 single sign-on.

❖ **Enabling SSO**

1 In the Management Console, expand the Servers node, and select the server.

2 On the server’s HTTP tab, select Single Sign On, then click Apply.

Customizing the location of system log files

To customize the location of the EAServer system log files, you can either:

- Specify the location for all log files by setting the djc.logFileLocation property in *local-setenv.bat* (Windows) or *local-setenv.sh* (UNIX), or
- Specify the location for the server log only by setting the djc.logFile property in the server properties file.

Note If you set both property values, djc.logFileLocation takes precedence.

The log files and their archives are moved from the old location to the new location.

Tracking HTTP sessions

You can specify whether to track HTTP sessions using a cookie by setting the web.useCookie property in a Web-application configuration script, either *webapp-${webapp}.xml* or *webapp-${webapp}-user.xml*. For example:

```
<property name="web.useCookie" value="true"/>
```

To disable cookies for the Web application, set the value of web.useCookie to false; the default value is true.

Java SE 6.0 support

EAServer 6.1 includes the Java Standard Edition (SE) 6.0 JDK. To start the server using JDK 1.6, run:

- Windows:

  ```
  start-server -jdk16
  ```

- UNIX:

  ```
  start-server.sh -jdk16
  ```

For a complete list of the server start-up options, see “Starting the server” in Chapter 3, “Creating and Configuring Servers,” in the *System Administration Guide*.

Deploying J2EE archives

To use the deploy command line tool to deploy a J2EE archive, use these options to specify the JDK version, the runtime library, and the compiler version for compiling the generated Java source files:

Compiler version	Options
JDK 1.4	-jdk14 -rt14 -target14
JDK 1.5	-jdk15 -rt15 -target15
JDK 1.6	-jdk16 -rt16 -target16

For example, to deploy *test.war*, load *eas-server-15.jar*, and compile the generated Java source files with the JDK 1.5 compiler, run:

```
deploy.bat -jdk15 -rt15 -target15 test.war
```

The full description of the deploy command is in Chapter 12, “Command Line Tools,” in the *System Administration Guide*.

Internationalization support

EAServer 6.1 supports internationalization-compliant entity names for:

- CORBA/Java packages and components
- EJB packages and components
- Web application packages and components
- Application clients
- Application packages and components
- Connectors
- Data sources
- Web services exposed from CORBA and EJB components