JBuddy COM/.NET
Getting Started Guide
Version 6.3

1. Setup

Below are the basic instructions for developing instant messaging and presence (IM) applications using COM and .NET development technologies and the JBuddy SDK as quickly as possible.

1.1 System Requirements

• J2SE JRE v1.3.1+ (see http://www.java.com/)
• .NET Runtime if developing using .NET
• JBuddy*Installer.jar (Installer for JBuddy Developer Tools)

1.2 Installation Steps

I.   Make sure a modern Java runtime (JRE 1.6+ recommended) is installed on your machine. The Microsoft version of Java is not supported. If you lack a modern JRE, go to http://www.java.com/ and follow the Download instructions.
II.   Download the JBuddy*Installer.jar package and double click it to install. (If you're reading this then chances are you've already completed both installation steps). If Java has been successfully installed on your machine, the JBuddy*Installer.jar file will be double clickable and a small language selection dialog box will appear. If you do not see the dialog box, please minimize other windows on your screen (sometimes the language prompt dialog does not come to the front above other windows).
III.   Be sure to select the **JBuddy COM/.NET Tools** installation option which is enabled on Windows platforms only.

1.3 Licensing

A free, limited use license is included with the JBuddy SDK. The license supports up to three (3) concurrent IM chat sessions and up to three simultaneously connected IM clients. A chat session is defined as a conversation between a buddy and an IM client connected using the JBuddy SDK. The chat session ends automatically when fifteen (15) minutes of time pass and no conversation takes place between the IM client connected by JBuddy SDK and the other buddy. If you wish to raise the limits of the free, included license, please contact Zion Software, LLC at http://www.zionsoftware.com/company/contact.html

1.4 Further Help

Further help may be obtained online at:
http://www.zionsoftware.com/support/ or within the forums at
http://www.zionsoftware.com/forums//

2. Visual Studio integration for JBuddy COM/.NET

Assuming you have successfully installed the JBuddy SDK with the JBuddy COM/.NET Tools components, this 'developer tutorial' will describe how to get started using JBuddy COM/.NET within Microsoft's **Visual Studio** IDE. General concepts from this tutorial are also applicable to other languages and IDEs for the Microsoft platform.

2.1 Add Reference

The JBuddy*Installer.jar attempts to register the JBuddy.dll with Windows. It is possible that security measures enabled within Windows could prevent this important step from occurring. If this is the case, you will need to manually register the JBuddy.dll. This can be done by right clicking on the JBuddy.dll and selecting the option to register the JBuddy.dll. Once registered, the JBuddy component is now accessible from within the IDE. In Visual Studio, click on the "Project" menu and then the "Add Reference…" sub menu. Once the "Add Reference" dialog is opened, click on the "COM" tab and scroll down and select "JBuddy SDK for COM & .NET", then click OK. See Figure 1 below. This allows the IDE to reference the JBuddy component which is natively a COM component. Visual Studio builds an 'interop' wrapper so that JBuddy can be used with .NET languages including C#, VB.NET and C++.
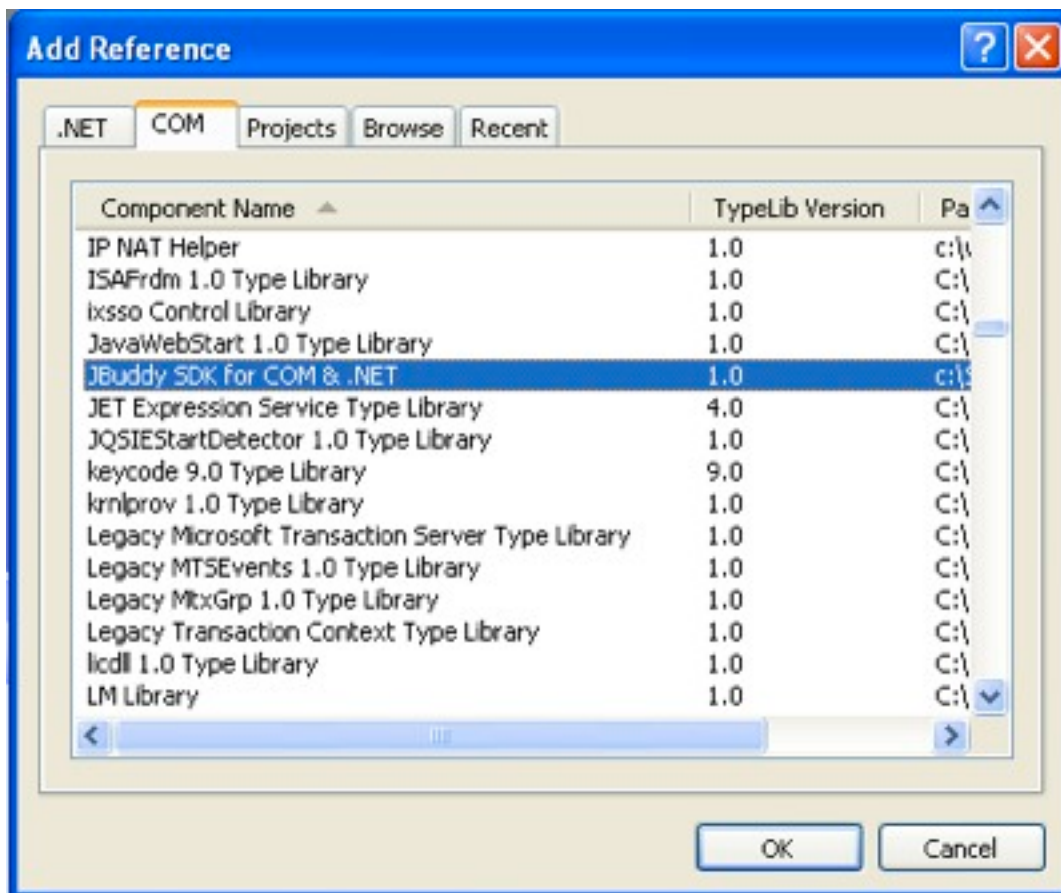

Figure 1 – Visual Studio 2008 Add Reference

3. Developing with JBuddy COM/.NET

After performing 2.1 **Add Reference** to make the JBuddy component available within the IDE, we are ready to look at the major components and how to build a simple IM application. The basic objects in JBuddy COM/.NET include JBuddyClass, Client/IClient, IBuddy, IBuddyList and IMessage.

3.1 General Concepts

First we need to create an instance of the factory class, JBuddy. This will allow us to create instances of IClient which are objects that represent users on the IM service. Once we have an IClient instance, we can tell it to connect to the IM service. Once connected, we need to wait for the handshake to complete so we wait for IClient's IsOnline() to return true. At this point, we are connected and online. Anyone with this client's screenname in their buddy list should see you 'online'. The IClient is now ready to send and receive instant messages (IM) and presence with 'buddies' on the IM service network.

Below is a snippet of C# code to illustrate how to obtain an instance of an IClient from the JBuddy factory, connect to the IM service and then wait for the client to go 'online'.

```
// get a handle on the Factory class
JBuddyClass jbuddy = new JBuddyClass();

// ask the Factory for a new iclient instance (a client for AIM with screenname and password).
Client client = (Client)jbuddy.ClientFactory(ProtocolType.PROTOCOL_AIM, screenname, password);

// register event handlers before going online so we don't miss our events of interest
client.AdminMessage += new _IClientEvents_AdminMessageEventHandler ( OnAdminMessage );
client.IncomingMessage += new _IClientEvents_IncomingMessageEventHandler ( OnIncomingMessage );
client.IncomingBujddy += new _IClientEvents_IncomingBuddyEventHandler ( OnIncomingBuddy );
client.ConnectionLost += new _IClientEvents_ConnectionLostEventHandler ( OnConnectionLost );

// wait until client isOnline and ready for commands
while(client.IsOnline() == false)
        System.Threading.Thread.Sleep(1000);

//Once online, we can send and receive instant messages and presence.

// send an instant message
client.SendIM("buddyname","Hello World ");

// Snippet of code below is the actual event handlers where we process the events
// we are interested in. Fill in the details…

private void OnAdminMessage(IMessage msg) {
    string sMsg = msg.Message; // admin message
}

private void OnIncomingMessage(IMessage msg) {
    string sMsg = msg.Message; // incoming message, now check the msg type
    MessageType type = msg.Type;
    if(type == MessageType.MESSAGE_TYPE_IM)
        Console.WriteLine("OnIncomingMessage: " + sMsg); // display Ims to the console
}

private void OnIncomingBuddy(IBuddy buddy) {
    // handle incoming buddy messages with presence
}
```

3.2 Samples & Demos

Inside the JBuddySDK\demos directory are small sample applications written in Java, VC++6, VB6, Excel (VBA), C#, and more. Notes on running the COM/.NET demos are given below:

- All C# demos require the .NET Framework. See Microsoft's knowledge base article http://support.microsoft.com/kb/318785 to determine if you have .NET framework installed.

- CmdLineClientDemo (java, C#, and VC6) demos and TrivialSendApp (VC6) run from a command window (run cmd) and require command-line parameters. In the case of C# and VC6 demos, enter the *.exe executable without parameters to see the required parameters. In the case of the CmdLineClientDemo for VC6, please see comments in GetLine.h regarding an operating system bug that may make entering commands difficult.

- GUIClient (VB6, C#) presents a very simple graphical IM client using the JBuddy COM/.NET API. The VB6 version requires three ocx system modules to fully work and they must be installed before it will display a post-login window: comdlg32.ocx, mscomctl.ocx, richtx32.ocx. These components should be readily available and installed many graphical applications. The C# GUIClient demo requires a valid username and password for an AIM account by editing the GUIClient.exe.config xml file. Replace username and password with account authorization credentials for an AIM account. If you wish to try a different IM network, you will need to edit the GUIClientForm.cs and replace the reference to PROTOCOL_AIM with a different reference and then recompile.

- TicTacToeBot (C#) demo - please see the README.txt for the java demo of the same name. The msgrp2p.xml file must be copied to the Live Messenger directory in order for this demo and the java demo to work. Also, this demo must be built (compiled) before it can be used. Finally, it requires a valid username and password for an MSN account by editing the TicTacToeBot.exe.config xml file. Replace username and password with account authorization credentials for an MSN/Windows Live Messenger account.

- JBuddyLoan (Microsoft Excel) demo runs as a VBScript Macro within Microsoft Excel and therefore requires Excel to be installed. The spread sheet demo requires a username, password and IM network to be specified and will signon to the IM network as a loan amortization IM bot. This demo has been tested with Excel '97 and may need adjustment to work with newer versions of Microsoft Excel. Additinally, Macros must be enabled for this demo to work.

- MFCSample (VC6) demo presents a very simple graphical IM window using MFC technologies. The MFCSampleDlg.cpp file requires modification and a valid username and password for an AIM account. Replace username and password with account authorization credentials for an AIM account. If you wish to try a different IM network, you will need to change PROTOCOL_AIM to a different network. This demo needs to be compiled (built) before it can be used.

4. JBuddy API for COM - API Documentation

The JBuddy API for Java is documented as Javadoc html pages inside the JBuddySDK \javadocs directory of your installation or online at http://www.zionsoftware.com/support/jbuddy/sdk/javadocs/

There are however several notable differences between the JBuddy API for Java and for COM. We shall refer to each simple as Java and COM for brevity. The differences will be covered below.

4.1 Java and COM/.NET Methods

All of the methods defined in Java are defined with standard Java casing style (lowercase letter first, then uppercase at each new word). COM methods on the other hand use headline cased (uppercase letter begins each new word). For example, IClient's isOnline() in Java is IsOnline() in COM. The description of the method in javadoc is the same for both Java and COM. In addition, standard getter and setter methods in Java are available in COM as properties. Read-only [get] properties in COM represent a method in Java where only a getter method exists. For example, IBuddy's getName() method in Java is a read-only property Name in COM and would be accessed as: string sName = buddy.Name; in C#. A property that is read/write [get, set] in COM represents two methods in Java – a getter and a setter. For example, IClient's getNickName() and setNickName(String nickName) in Java are represented in COM as [get, set] NickName. Setting the NickName of a client in C# would look like a typical assignment: client.NickName = "SuperMan"; If a setMethod in Java takes more than one argument, then it will usually look the same in COM with headline case since a [get, set] method is for getting or assigning one value only. For example, IClient's getStatus() and setStatus(int status, String customerAwayMessage) in Java is represented as a read-only [get] Status property AND a SetStatus(StatusType, string) method in COM.

4.2 Java Fields / COM enums

All of the constants defined in Java are defined as enums within COM and grouped according to their function and purpose. All the value names of the enums begin with the name of the enum (minus the "Type") in all CAPS_. These are described below:

4.2.1 StatusType

IBuddy fields in Java represent status or presence for both buddies and clients. They are all defined within the StatusType enum in COM. For example, IBuddy.OFFLINE in Java is StatusType.STATUS_OFFLINE in C# and just STATUS_OFFLINE in C++. See IBuddy javadoc for a description of the enum values.

4.2.2 PrivacyType

IBuddyList fields in Java related to permit modes (user privacy) are grouped in COM as PrivacyType enum. See IBuddyList javadoc for a description of the enum values.

4.2.3 BuddyListType

IBuddyList fields in Java related to types of BuddyList are grouped in COM as BuddyListType enum. See IBuddyList javadoc for a description of the enum values.

4.2.4 MessageFormatType

IClient fields PLAIN_TEXT and RICH_TEXT in Java are used with getPlainTextMode() and setPlainTextMode(int). In COM, IClient has [get, set] PlainTextMode property. The IClient fields are grouped as MessageFormatType enum and the RICH_TEXT field is called MESSAGE_FORMAT_RAW in COM to better describe that the incoming messages are in their native (raw) format.

4.2.5 ProtocolType

IClient fields in Java related to protocols are grouped in COM as ProtocolType enum. See IClient javadoc for a description of the enum values.

4.2.6 MessageType

IMessage fields in Java related to types of messages are grouped in COM as MessageType enum. See IMessage javadoc for a description of the enum values.

4.3 JBuddy, JBuddyClass

In COM, JBuddy represents the application object. It contains all the static methods located in Java's IClientFactory and IMessageFactory classes.

The IMessageFactory::factory has been renamed to MessageFactory.

There are two IClientFactory::factory replacement methods, ClientFactory and ClientFactoryEx.

ClientFactory implements all IGateway functionality as native COM events, therefore a reference to an IGateway implementation does not need to be supplied.

The ClientFactoryEx method was supplied to allow a developer to pass a COM object that implements an IGateway interface to the client factory. It will forward all callbacks through the methods defined in the IGateway interface (COM events will also still be fired). Using IClientFactoryEX and implementing a COM object is beyond the scope of this document.

For .NET, JBuddyClass is the interop wrapper class for JBuddy. It is used in place of JBuddy.

4.4 IClient, Client

All clients types created from the ClientFactory or ClientFactoryEx implement the IClient interface. The methods and properties are the same as the Java Docs.

The Client data type is only needed when COM events need to be processed. All client types can be cast to the Client Type.

If the ClientFactoryEx is used with a valid IGateway, only IClient will need to be used.

4.5 IGateway

This interface is only needed when you wish to handle all callbacks directly without using the COM's Event ConnectionPoint architecture. A valid COM object needs to be created that implements the IGateway interface. It should then be passed to ClientFactoryEx to create one or more clients. The IGateway methods will then be called whenever an IM event is received.

4.6 Callbacks and Event Handling

Each COM enabled language has a different way to sink COM events. The important thing to note is that only Client object raise events, not IClient.

A VB6 example of sinking events is:

```
Private WithEvents oClient as Client

Private Sub oClient_IncomingMessage( oMessage as IMessage )
End Sub
```

If COM events are not supported or an application needs to create multiple Client objects, yet only wants one method to handle all events. Then using the IGateway interface would be a better choice.

The developer will need to create a simple COM object that implements the IGateway interface. Pass that object's reference to each call to ClientFactoryEx.

4.7 Errors and Exceptions in COM

When an error occurs in the JBuddy dll, it will use the normal COM error handling procedure to return the error that occurred. However, there is more information available in the JBuddy.Exceptions collection. It will have an entry in the collection for each function that was called before the error occurred. There are methods exposed to all your program to all to the exception list so that an entire stack trace may be built. To process the exceptions, you may iterate through the collection, or you may call the method JBuddy.Exceptions.Display(). This method will display a dialog box with the entire stack trace displayed.

4.8 Troubleshooting

To enable lower level troubleshooting for JBuddy.dll, you can use regedit to add a DWORD to the registry:

```
HKEY_LOCAL_MACHINE\Software\Zion\JBuddy\EnableLogging=1
```

Once added, new launches of the JBuddy.dll enabled application will create (or append) to C:\JBuddy.log. Look here when problems occur and you suspect the JBuddy COM/.NET components may not be loading properly or misconfigured.