Sametime
Version 9.0

# *Sametime 9.0*
# *Software Development Kit*
## *Integration Guide*

# Edition Notice

**Note:** Before using this information and the product it supports, read the information in "Notices."

This edition applies to version 9.0 of IBM Sametime (program number 5725-M36) and to all subsequent releases and modifications until otherwise indicated in new editions.

**© Copyright IBM Corporation 2006, 2013.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

# Chapter 1. Introduction

The IBM® Sametime® Connect instant messaging client is built on the Eclipse-based IBM® Lotus® Expeditor® platform. It leverages the Eclipse plug-in framework to provide developers with extensibility features that go far beyond those available in previous releases.

This document, along with the samples and Javadoc™ in the IBM Sametime Software Development Kit (SDK), provides you with the information you need to build Eclipse plug-ins that extend the capabilities of IBM Sametime Connect and integrate your own applications with IBM Sametime.

This document presents information in the following order:

- An overview of IBM Sametime Connect
- An overview of IBM Sametime Connect architecture
- A high-level tutorial on extending IBM Sametime Connect
- Descriptions of the sample extension plug-ins.

For more information about the contents of the IBM Sametime SDK, see the readme.txt file in the st9sdk subdirectory of the directory in which you installed the SDK.

## 1.1 Terms to know

The following terms are used throughout this integration guide.

Table listing terms and definitions used throughout this guide.

| TERM | DEFINITION |
|---|---|
| API | Application Programming Interface |
| Eclipse | An open platform for rich client development. Although Eclipse is a Java™-based platform, it can be used to build tools for other programming languages |
| Extension | A mechanism that expands the functionality of a plug-in by connecting to an extension point. An extension is also referred to as a "contribution" to another plug-in. |
| Extension point | The specification that declares how extensions can add to the functionality of a plug-in. Several plug-ins can contribute to an extension point by defining extensions in the plug-in's extension manifest file, plugin.xml. |
| Lotus Expeditor | The platform used by IBM Sametime Connect and IBM managed client products.<br><br>Lotus Expeditor provides a runtime environment and integrated middleware components for extending many enterprise applications to server-managed laptop and desktop systems, and mobile devices running supported operating systems.<br><br>Expeditor includes the Eclipse Rich Client Platform (RCP) and Java Runtime Environment (JRE), as well as additional services used by managed client products. The Expeditor platform is available as a separate product, so that third parties can build their own Rich Client applications. |
| IDE | Integrated Development Environment, The IBM® Rational® Application Developer IDE and Eclipse IDEs are examples of IDEs. |
| ISV | Independent Software Vendor |

| | |
|---|---|
| J2SE | Java™ 2 Platform, Standard Edition. This is the standard JRE for desktop applications. |
| JRE | Java Runtime Environment. This is the technology that allows Java applications to run. |
| OSGi ™ | The OSGi Service Platform is a standard that defines, among other things, how Eclipse plug-ins are packaged. |
| Plug-in | An Eclipse platform feature component. A plug-in is the basic building block of an Eclipse application. |
| Plug-in registry | Registry of declared plug-ins, extension points, and extensions managed by the Eclipse Runtime Platform. |
| RTC | Real-time Collaboration, which describes synchronous technologies such as instant messaging, presence awareness, Web conferences, telephony, and so on. |
| SIP | Session Initiation Protocol, a standard protocol for managing interactive sessions between users. SIP is used for instant messaging, presence, telephony, and a number of other applications. |

## 1.2 Suggested reading

IBM Sametime Connect is built using a number of different technologies, including Eclipse, Java™ and Lotus Expeditor.

In order to build plug-ins for the client for IBM Sametime, it is necessary to have some understanding of these technologies. Although you do not need to be an expert, the more familiar you are with these technologies, the more success you will have building plug-ins for IBM Sametime Connect and other Expeditor based products.

The following list suggests readings including tutorials that you can use to better acquaint yourself with different technologies IBM Sametime Connect uses.

Table listing suggested readings.

| WHAT | WHERE |
|---|---|
| Eclipse Organization Community home page | http://www.eclipse.org |
| Eclipse Workbench User's Guide basic tutorial | http://help.eclipse.org/help32/index.jsp |
| Eclipse project resources | http://www.ibm.com/developerworks/opensource/top-projects/eclipse.html |
| The Java Tutorial | http://java.sun.com/docs/books/tutorial/ |
| Java technology | http://www.ibm.com/developerworks/java |
| IBM® Lotus® Expeditor | http://www.ibm.com/software/sw-lotus/products/product1.nsf/wdocs/expeditor |

## 1.3 Development requirements

In order to work with the sample plug-ins described in this guide, or to create your own plug-ins for IBM Sametime Connect, you will need the following:

- Version 3.4 of the Eclipse IDE.
- The Lotus Expeditor 9.0 Toolkit.
- A standard JRE, 1.5.0 or higher, to run the Eclipse IDE. Eclipse does not include its own JRE. Note that the JRE used to run the Eclipse IDE is not the same one you'll use to compile and run your plug-ins – Eclipse allows you to specify a different runtime environment to use for compiling and running your code. It's recommended that you use a J2SE 5.0 JRE to run the Eclipse IDE. For information about downloading the IBM J2SE 5.0 JRE, see the following site:
  http://www.ibm.com/developerworks/java/jdk
- A Microsoft® Windows®, Linux, or Mac OS X platform supported by the IBM Sametime Connect release. See the release notes on your IBM Sametime server for a list of supported client platforms. Note that the samples documented in this guide were only tested with Microsoft Windows XP.

**Note**: The Development checklist in Chapter 4 provides step-by-step instructions for setting up your development environment.

# Chapter 2. What is IBM Sametime?

IBM Sametime is a market-leading product and platform for real-time collaboration. IBM Sametime Connect is a desktop application that provides instant messaging, presence and Web conferencing features that enable people to collaborate in real time, regardless of their physical location.

## 2.1 Why extend IBM Sametime Connect?

IBM Sametime Connect offers more than simple instant messaging and presence features. Because it is built on the Eclipse-based Lotus Expeditor platform, a variety of plug-ins that expand the functionality of IBM Sametime Connect are shipped with the product, and third parties can build additional plug-ins. Some examples of plug-in capabilities that could be added by third parties are enhanced audiovisual conferencing; instant polls of a user community; or the ability to capture, store, and search instant message conversations or meetings.

The ability to create plug-ins to meet the growing needs of the instant messaging community, combined with its already proven security model and numerous user interface enhancements, makes IBM Sametime Connect a powerful tool to help companies harness the potential of their employees.

In addition, and perhaps most important, the instant messaging and presence features in IBM Sametime Connect are used by other Lotus Expeditor-based products, including IBM Notes 8.0 and later and the Lotus Expeditor Desktop Client. This means that plug-ins that you build for IBM Sametime Connect will work with the other Lotus Expeditor-based products, as long as you adhere to the public interfaces documented in this guide. Your investment in developing plug-ins for IBM Sametime Connect gives you access to a market that goes beyond IBM Sametime Connect users.

## 2.2 The relationship between IBM Sametime and Eclipse

IBM continues its commitment to the future of open source code with the release of IBM Sametime Connect. Because IBM Sametime Connect is developed on and for the Eclipse platform, you should have a basic understanding of Eclipse. If you do, skip this section and go directly to section 2.4, *Using plug-ins to extend IBM Sametime Connect*. Otherwise, read the next section for a high-level introduction to the main concepts of Eclipse.

## 2.3 A brief overview of Eclipse

Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software. Eclipse provides extensible tools and frameworks that span the software life cycle, including support for modeling; language development environments for Java, C/C++, and others; testing and performance; business intelligence; and rich client applications and embedded development. A large, vibrant ecosystem of major technology vendors, innovative start-ups, universities and research institutions, and individuals extend, complement, and support the Eclipse Platform.

http://www.eclipse.org

At its simplest level, Eclipse is an open-source foundation on which you can develop applications. The building blocks of Eclipse are called plug-ins, which are the smallest unit of Eclipse functionality.

Eclipse plug-ins are like a power strip: there is a part that plugs into the electricity source and parts that other plugs can plug-into. An Eclipse plug-in may define

- Extensions, which are analogous to the plug part of the power strip, and
- Extension points, which are analogous to the sockets of the power strip.

Just as any appliance with a plug can use a power strip, so can any application with a plug, or extension, that fits use the extension points of a plug-in. In this way extension points can extend the original application in ways that grow with the needs of the user community.

The following figure shows a high-level outline of the Eclipse architecture as IBM Sametime Connect uses it.

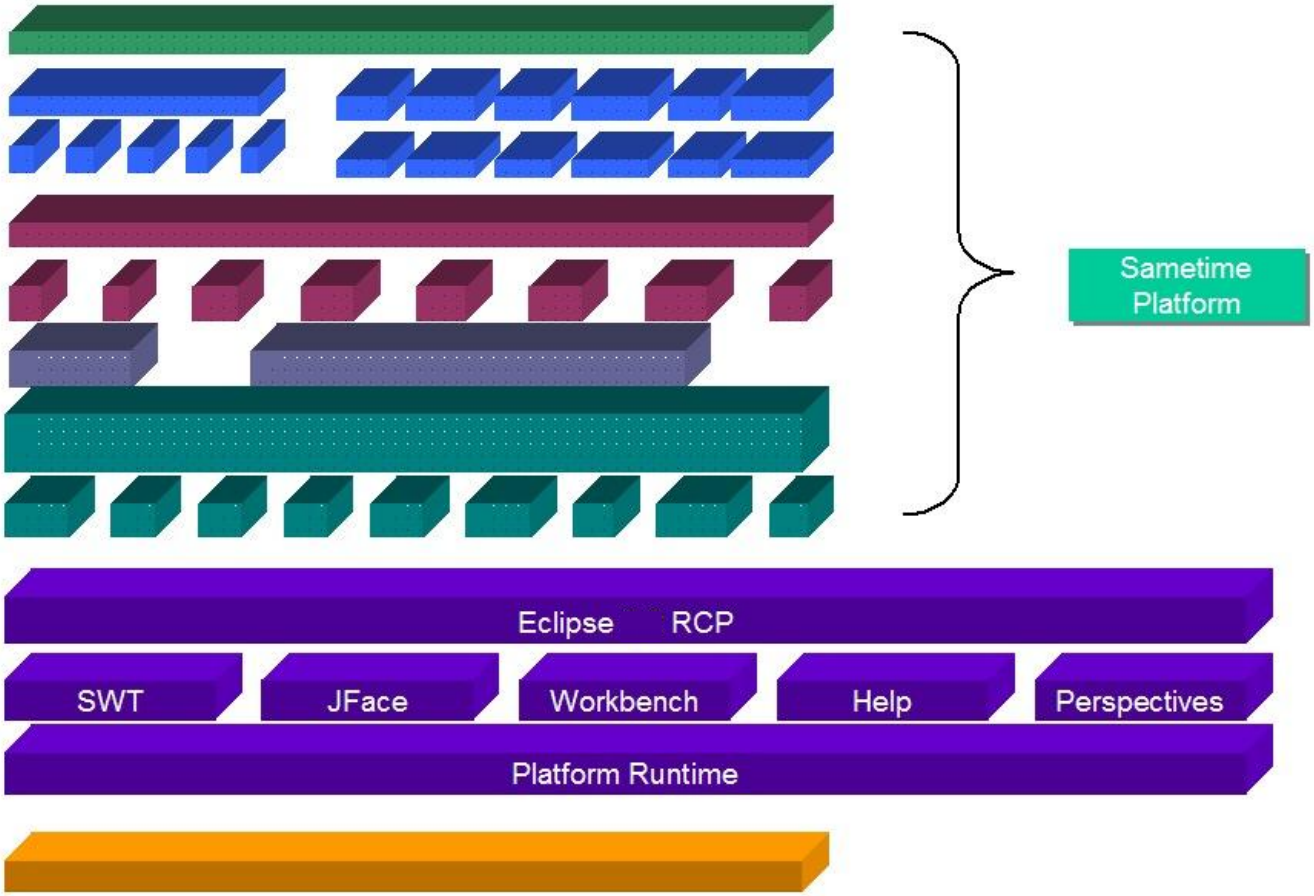



**Figure 1: Sametime architecture stack with an emphasis on Eclipse**

Chapter 3, "Overview of the IBM Sametime Connect architecture," discusses the components labeled Sametime Platform in the illustration above.

### 2.3.1 Eclipse platform components

The following table lists and describes some of the Eclipse platform components that IBM Sametime Connect uses.

Table listing some Eclipse platform components used by IBM Sametime Connect.

| COMPONENT | DESCRIPTION |
|---|---|
| Platform Runtime | Provides the foundational support for plug-ins and for the plug-in registry, a mechanism for declaring extension points and for extending objects dynamically. The Eclipse runtime uses the standard OSGi framework to define how plug-ins are packaged. |
| Help | Help provides a plug-in with HTML-based online help and search capabilities; help content is contributed through plug-ins that are recognized at runtime. |
| JFace | A UI framework, working in conjunction with the standard widget toolkit (SWT), for handling many common UI programming tasks. |
| Preferences | An Eclipse-managed collection of indexed dialog boxes. Plug-ins can contribute new Preference pages using an extension. |
| SWT | The Standard Widget Toolkit. SWT provides access to the user-interface facilities of the operating systems on which it is implemented. SWT-built applications leverage the UI of the host system more than do other Java toolkits, such as Swing, for example. |
| Workbench | The Workbench provides a highly scalable, open-ended, multi-window environment for managing views, editors, perspectives (task-oriented layouts), actions, wizards, preference pages, and more. |

For more information on Eclipse, refer to the following sources:

- http://www.eclipse.org
- http://www.ibm.com/developerworks/opensource/top-projects/eclipse-starthere.html

### 2.3.2 How are plug-ins put together?

One way to understand how plug-ins are constructed is to start with a high-level overview of the connections between a plug-in, its extensions and extension points, and the class or classes implemented by the extensions and extension points.

Typically, an extension's class attribute denotes the class that implements the behavior that the extension represents, for example, an extension that defines a contribution of a new action specifies

- The action's label (statically)
- A class that will be instantiated to perform the request function

To understand how extensions and extension points perform the tasks that they were designed to do, you must understand their methods (APIs) and the sequence of events to key interactions.

## 2.4 Using plug-ins to extend IBM Sametime Connect

That third-party developers can use plug-ins to extend the functionality of IBM Sametime Connect is one of the product's key values. The IBM Sametime SDK includes sample plug-ins, each of which can be used as guides to creating a more complex plug-in. These plug-ins are integrated through user interface buttons, menus, toolbars and custom actions.

The following images show how the user interface elements of the sample plug-ins appear in the IBM Sametime Connect main window.

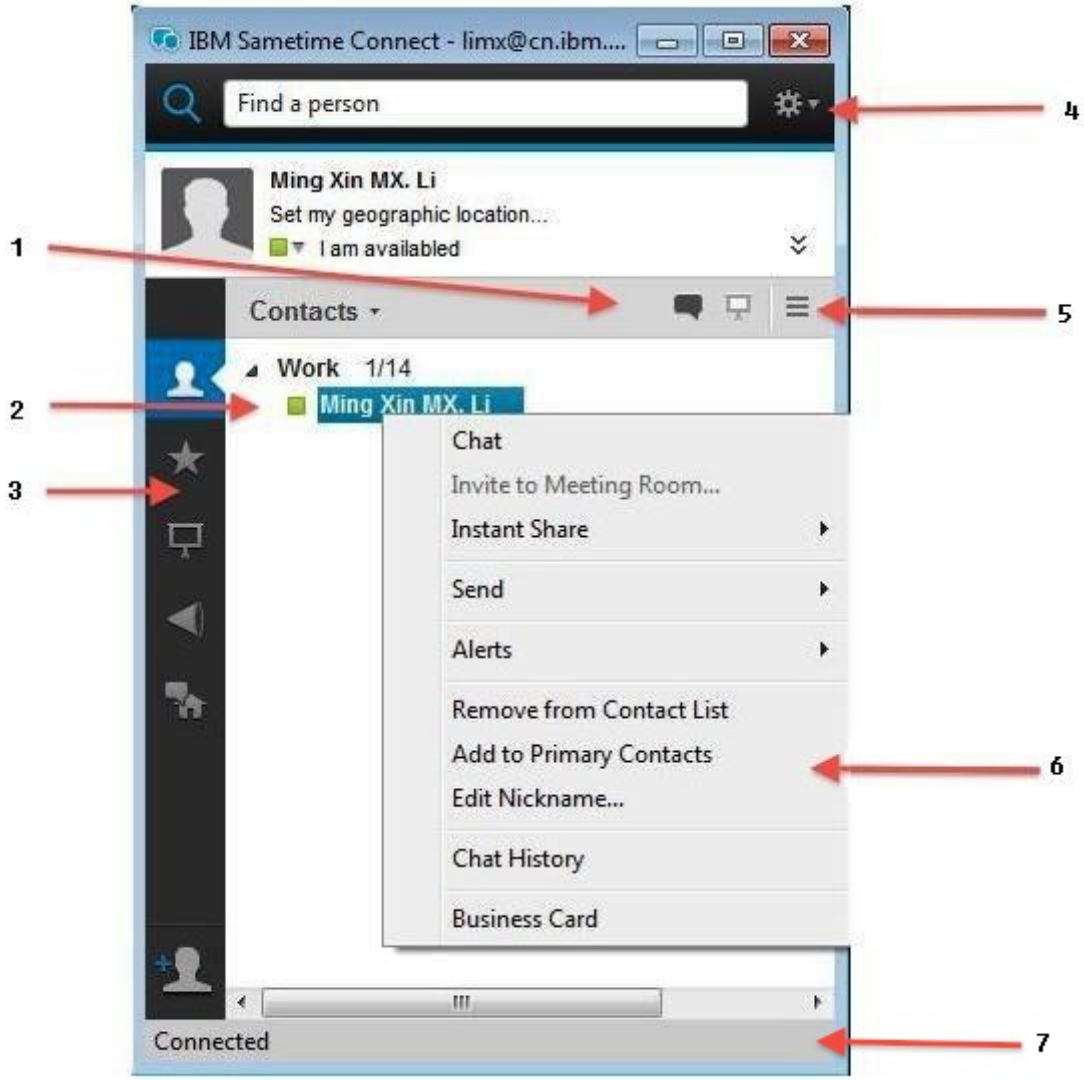



**Figure 2: Main UI**

Key:

1. Ribbon Title Bar
2. Contact names and status icons
3. Ribbon Bar
4. Gear Menu
5. Menu option for Ribbon title bar
6. Context Menu

7. Status bar

The following image shows a chat UI that has several plug-in enhancements, such as an icon to start conversations with voice chat (using Voice over Internet Protocol, or VoIP):

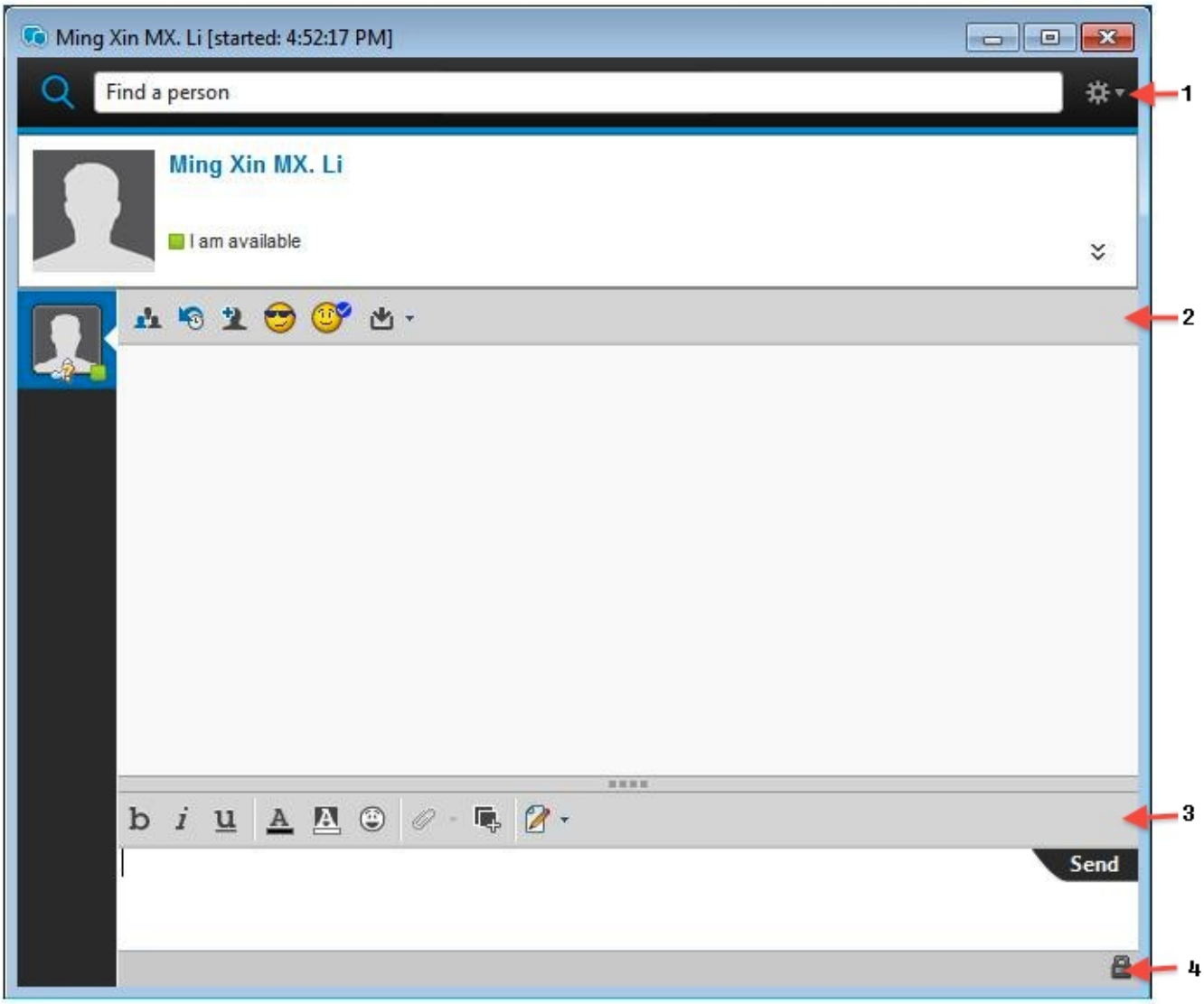



**Figure 3: Chat UI**

Key:

1. Gear Menu(Extensible)
2. Action bar
3. Message tool bar (an action bar)
4. Status bar

## 2.5 Table of sample plug-ins

The following table describes the sample plug-ins that ship with the IBM Sametime SDK.

Table listing sample plug-ins shipped with the Sametime SDK.

| PLUG-IN | PURPOSE | EXTENSIONS |
|---|---|---|
| Recent Buddies | Stores a list of most recent chat partners into a type of preferred users list. | *core.serviceListeners<br>*messages.MessageHandlerListener<br>com.ibm.rcp.ui.shelfViews<br>org.eclipse.ui.views<br>org.eclipse.ui.popupMenus (object contributions) |
| Buddy Note | Adds an application window to the shelf view area for typing text notes about selected buddies. | *messages.MessageHandlerListener<br>com.ibm.rcp.ui.shelfViews<br>org.eclipse.ui.views |
| Quick Response | Creates an action button that opens a set of chat responses that can be added to. | *chatwindow.popupAddOn<br>*chatwindow.chatAction<br>org.eclipse.ui.preferencePages |
| Acronym Expander | Automatically converts common acronyms, like LOL and BRB to “laughing out loud” and “be right back”. | *messages.MessageHandlerListener<br>org.eclipse.ui.preferencePages |
| Snippets | Provides a grab bag of small code samples. | *chatwindow.chatAction<br>*chatwindow.chatArea<br>*chatwindow.nwayListExtension<br>*chatwindow.popupAddOn<br>*messages.MessageHandlerListener<br>org.eclipse.ui.viewActions<br>org.eclipse.ui.actionSets<br>org.eclipse.ui.actionSetPartAssociations<br>org.eclipse.ui.popupMenus (object contributions)<br>org.eclipse.ui.view.Actions |

To save space, the prefix com.ibm.collaboration.realtime is shown simply as "*" in the table above.

## 2.6 Deploying custom plug-ins to IBM Sametime Connect users

When you’ve finished developing and testing your custom plug-in for IBM Sametime Connect, you’ll need to make your plug-in available to your target end users. To do so, you have a couple of options:

Work with the Sametime administrator to automatically provision your plug-in to all IBM Sametime Connect users in a particular community. When using this option, your plug-in is automatically downloaded to the client when the user launches IBM Sametime Connect. The user is notified of the update and can have the client restart automatically to load the new plug-in. The user doesn't need to take any other action.

Post your plug-in on an Eclipse update site, and instruct end users to install the plug-in using the IBM Sametime Connect Install Plug-ins UI. Note that Sametime administrators can disable the ability for end users to install plug-ins in this way, in which case you'll need to use the first option.

For more information about using an Eclipse update site to distribute plug-ins, see the following article:

http://www.eclipse.org/articles/Article-Update/keeping-up-to-date.html

**Note**: Starting with release 8.0, you can no longer just drop new plug-ins into the plug-ins directory of IBM Sametime Connect. The IBM Sametime Connect client is now a fully managed client on the Lotus Expeditor platform, so all new plug-ins must be provisioned using an update site (see Chapter 6).

# Chapter 3. Overview of the IBM Sametime Connect architecture

The previous chapter introduced the features of IBM Sametime Connect and gave you a flavor for what is necessary to extend the client with added functionality. This chapter presents an overview of the IBM Sametime Connect architecture so that you will understand how the sample plug-ins described in subsequent chapters, as well as the plug-ins that you build, integrate with IBM Sametime Connect. For reasons of brevity, our focus will be on the major components you will use frequently in your own code that are demonstrated in the guide's source code samples. Despite the occasional cursory treatment of some subjects, it is our intent that the chapter's main subjects will develop your "programmer's intuition" of how the pieces fit together. This understanding will serve you well as a foundation for your continued exploration of the possibilities the client for IBM Sametime offers you and your customers.

**Note:** At this point you should have a firm understanding of the Eclipse platform, especially with regard to plug-ins and how their extension points and extensions bring functional units of code together. In addition, you should be familiar with the Eclipse user interface programming model (JFace, Workbench views, and SWT). See *Suggested reading* in section 1.2 as required.

## 3.1 Architectural goals

Every programmer begins learning how a system works by understanding the layers of components that ultimately deliver its functionality; whether he or she is examining a new operating system or an application framework. Beginning with the study of component layers makes it easier to grasp the division of responsibilities of layered designs. In this respect, learning about IBM Sametime Connect is no different than learning about any other product. Its base layers provide common capabilities and the upper layers add more specialized behaviors. This chapter will introduce these layers.

The Eclipse rich client platform serves as the foundation of IBM Sametime Connect. A key benefit of this choice of platform is that it provides for the inheritance of its ability to seamlessly integrate with other "application components," namely, other Eclipse plug-ins. As you read in the introduction to the samples, the contributions of these plug-ins can be as simple as a new menu choice that displays information in a dialog or as complex as a wholly integrated mini application. The overarching goal of the IBM Sametime Connect architecture is to provide three benefits: extensibility, integration, and reuse.

## 3.2 Programmer's point of view

The benefits of using Eclipse as an integration platform to the end-user is its ability to add new functionality into the environment. A flexible yet consistent user experience is a strong selling point, but the advantage for Eclipse programmers is the leveraging of their existing skills beyond developing standard Eclipse Rich Client Platform (RCP) line of business application development. Those who are new to Eclipse will find the Eclipse Web site and DeveloperWorks bursting with helpful articles, and books on Eclipse application development are readily available. Most of the principles of Eclipse application development are directly applicable to creating extensions to IBM Sametime® Connect. Even those without previous Eclipse application development experience will recognize in these principles the common object-oriented programming patterns. For example, Eclipse follows the classical separation of model (Workbench) and view (ViewParts and their lower-level graphical components in the JFace framework). The clear separation of component responsibility reduces the need to learn each individual piece to be productive; as you will learn, the IBM Sametime architecture embraces this same principle.

## 3.3 Key concepts

Earlier sections described the layers of the Eclipse components and defined plug-ins as its smallest unit of functionality. Plug-ins define extensions points to declare where other plug-ins can "hook into" their functionality with extensions. The relationship between plug-ins is defined not only by the component layer that the plug-ins belong to within the architecture, but also by the integration points among plug-ins.

The diagram below shows major IBM Sametime Connect components. The Java run-time, Eclipse RCP, and Lotus® Expeditor layers form the foundation common to IBM's managed client products, which include IBM® Notes® 8 and Lotus® Expeditor Client™.

The components enclosed by the box are specific to the client. These components include public and implementation-specific plug-ins. At this stage of development, only a subset of these components contribute to the public API and are discussed in this guide.

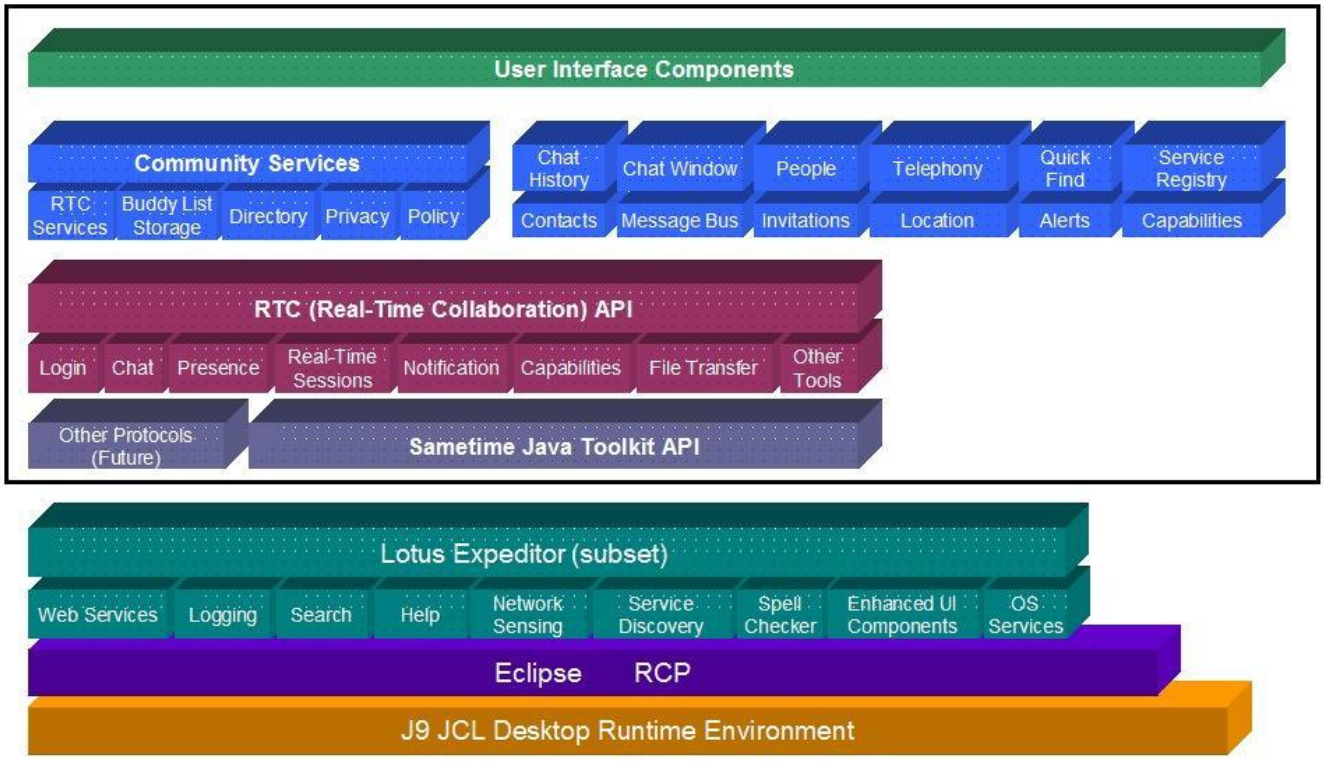



**Figure 4: Sametime architecture**

Most of the rectangles shown above the Eclipse 3.4 RCP layer correspond to one or more plug-ins. Some, like the Sametime Java Toolkit, represent libraries that may be referred to by plug-ins above them, but aren't plug-ins themselves. That is, not all code is "plug-in aware", either because it pre-dated the Eclipse plug-in model, or because it does not require the extensibility that Eclipse provides. The rest of this chapter will introduce the key component layers, plug-ins, extension points, and classes and interfaces that will be part of your extensions to IBM Sametime Connect.

## 3.4 Summary of component layers

This section briefly describes some of the key component layers depicted by the diagram in the previous section. The Eclipse rich client platform was already described in the section, "*A* brief overview of Eclipse."

### 3.4.1 IBM Lotus Expeditor

The Lotus Expeditor platform provides services used by IBM Sametime Connect. Lotus Expeditor is also used in IBM Notes®. Lotus Expeditor includes the Eclipse RCP and Java run-time environment (JRE), as well as additional services used by managed client products.

Lotus Expeditor provides a common services platform on a variety of devices, ranging from handheld devices to desktop computers. The full Lotus Expeditor platform provides support for J2EE (Java 2 Enterprise Edition) and Web applications. IBM Sametime Connect includes a subset of the full Expeditor platform that includes only those services required by Sametime.

Lotus Expeditor includes the Lotus Expeditor Toolkit, which facilitates the development of applications for the Lotus Expeditor platform. The latest current release of Lotus Expeditor is 9.0. IBM Sametime Connect 9.0 is based on Lotus Expeditor 9.0.

With the release of Lotus Expeditor, you have access to the complete set of Lotus Expeditor Deployment APIs and services, and you'll be able to use the Lotus Expeditor Client Toolkit to build plug-ins for IBM Sametime Connect.

### 3.4.2 RTC (Real-Time Collaboration) API

The Real-Time Collaboration (RTC) platform provides synchronous collaboration services used by IBM Sametime Connect and other IBM offerings. Some of the services provided by the RTC platform include:

- Login services
- Real-time session management
- Presence features
- Event notification
- Instant messaging and file transfer meeting tools

The RTC API is a protocol-agnostic API, so that applications do not need information about the transport protocol being used to deliver messages. The RTC API will support both the Sametime Virtual Places (VP) protocol and SIP.

### 3.4.3 Sametime Java Toolkit

The Sametime Java Toolkit provides Java APIs to access many of the services provided by the IBM Sametime client and server products. The Sametime Java Toolkit is a widely-used API that has been available for many years. The Sametime Java Toolkit is used by the RTC platform to access Sametime-specific services. When you build plug-ins for IBM Sametime Connect, you can use the Sametime Java Toolkit to access Sametime services that are not provided by other IBM Sametime Connect APIs.

The Sametime Java Toolkit is included in the Sametime SDK. See the readme.txt file in the st9sdk directory of the IBM Sametime 9.0 SDK for more information.

### 3.4.4 Key classes and interfaces

The samples included with this guide demonstrate the use of the client's public interfaces and are designed to be modestly useful as well as a good source for learning to program to IBM Sametime

Connect. Many of your own implementations may well start from code copied from these samples. However, sometimes it is difficult to distinguish the IBM Sametime Connect-specific code from the surrounding code related to programming to Eclipse's Rich Client environment. This section will bring to your attention the IBM Sametime Connect classes and interfaces that will be part of most, if not all, extensions to the client. Where appropriate, the samples demonstrating the use of a particular interface will be noted too.

**Reminder:** This chapter assumes you are familiar with the Eclipse plug-in development model. The architecture of IBM Sametime Connect closely follows the same pattern for extending the system's functionality.

#### 3.4.4.1 Platform

As discussed in the introductory chapter, the IBM Sametime Connect components can be divided into two major categories: the model and the view (user interface). To further reduce interdependency among non-UI components, the platform defines pluggable "services" managed by the singleton ServiceHub. The following sections introduce the model classes and interfaces you will most commonly use in your extensions of the client.

##### 3.4.4.1.1 Model

The model classes represent the server data that is reflected in the client. For example, the chat partners (Person instances) shown in the IBM Sametime Connect contact list are grouped into Group instances. Person.addPropertyChangeListener enables your code to detect changes to specific individual instances of Person. To receive more general notifications like status changes (online, offline, do not disturb), chat message received, etc., extend the com.ibm.realtime.collaboration.messages.MessageHandlerListener extension point and override methods like the handleMessage(PartnerStatusUpdateMessage) and handleMessage(ImTextMessageReceived) method in your handler class. See the RbMessageHandler in the Recent Buddies sample for details.

Table describing the data model classes.

| CLASS (REALTIME.CORE) | DESCRIPTION |
|---|---|
| Person | Interface representing a person. The Person instances are maintained in an in-memory cache. |
| Group | Interface representing groups of Person instances and shown in the contact list (superinterface of PrivateGroup and PublicGroup). |

##### 3.4.4.1.2 Service and Plug-in Registry

The services available to users are dependent on the IBM Sametime Connect features they have installed, the communities they belong to, and any third-party products that extend the base product. The Services framework builds upon the ability of Eclipse to dynamically bind plug-ins by managing a registry of services. Examples of services defined by the base include directory services and services that manage IM communities.

The ServiceHub class manages access to the services layer. The static getService method allows plug-ins to request the use of various services without having to know the provider’s identity in advance.

Table listing interfaces and classes related to the service hub.

| CLASS (REALTIME.CORE) | DESCRIPTION |
|---|---|
| ServiceHub | Registry of services defined with the com.ibm.collaboration.realtime.core.services point. |

| | |
|---|---|
| ServiceException | Exception that is thrown when a ServiceHub method fails (e.g., ServiceHub.getService(svcType). |

#### 3.4.4.1.3 Community Services

Instant Messaging users are grouped into communities, and the capabilities of a community determine what services are available to users, for example directory services used to search for community members. The community manager maintains the list of all available communities registered with the client.

Table listing community services classes.

| CLASS (REALTIME.COMMUNITY) | DESCRIPTION |
|---|---|
| Community | The community is the entry point into the registration information, such as the user's ID, and services offered by a particular IM server. Community-specific services, such as directory services, are available from the community instance. |
| **CLASS (REALTIME.DIRECTORY)** | **DESCRIPTION** |
| DirectoryService | Service to retrieve directory information for people or groups<br><br>Note: Because each community may have a different DirectoryService implementation, do not use ServiceHub.getService to get a DirectoryService instance. Instead, use Community.getService to get a DirectoryServiceFactory instance, and then use the DirectoryServiceFactory.getDirectoryService method to get a DirectoryService instance for the community. |
| ContactInfo | Contact's community ID, name, and directory attributes. |
| GroupInfo | Group's description, members, etc. |
| DirectoryInfo | Directory attributes for a user (address, e-mail, etc.). |

#### 3.4.4.1.4 Message Bus – General Event Notification

IBM Sametime Connect components communicate key events through a common messaging bus. Components that use the messaging framework are called *participants*. The sender does not need to know anything about the component responsible for processing the message. The sender only interacts with the message bus.

Message classes are defined to make the creation and processing of the messages simple, fast, and type safe. For each message, there is a subclass of the base Message class that is specific to that type of message. For example, the class ImTextReceivedMessage represents the message that is broadcast when the user receives an incoming text message.

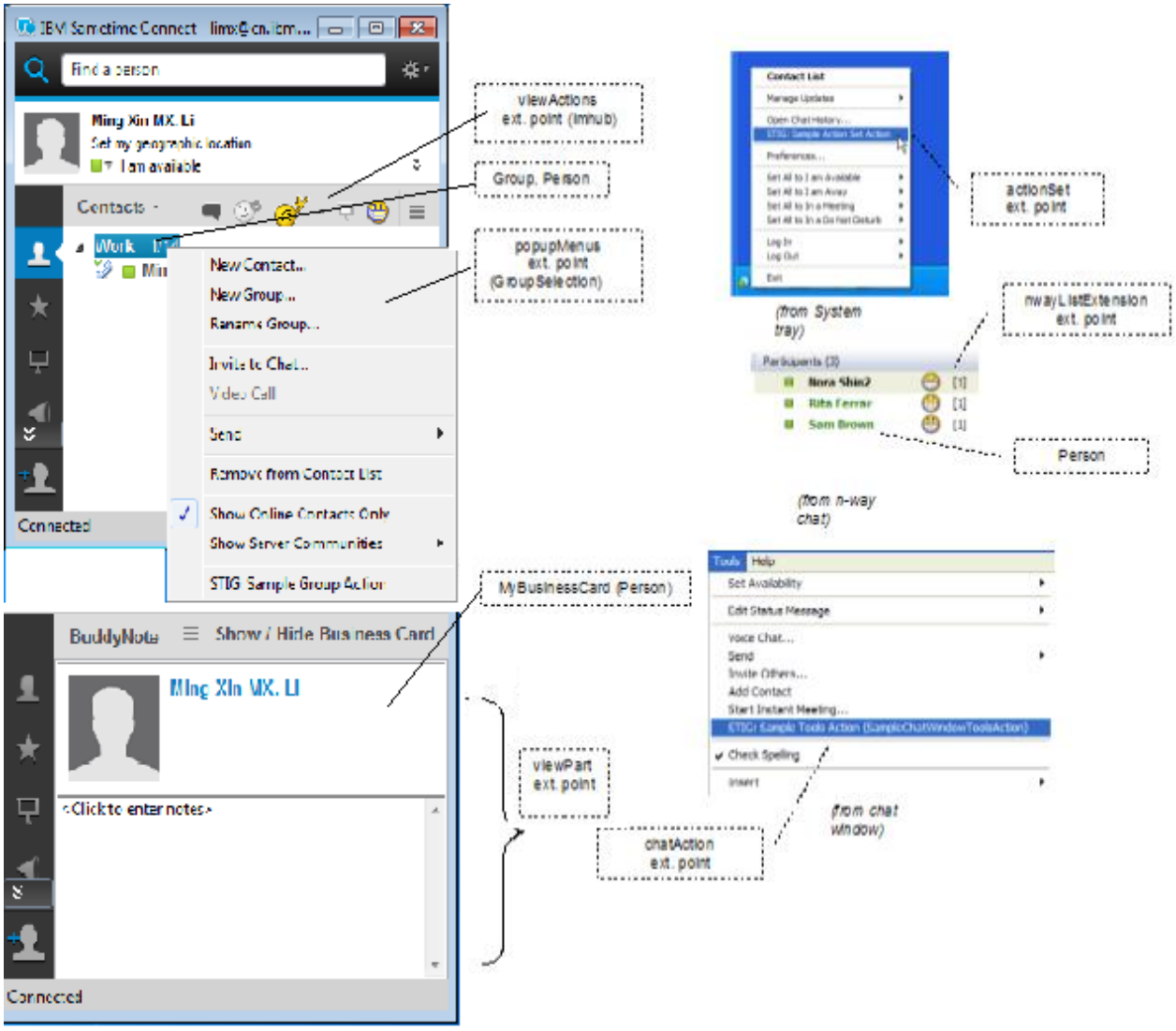
The MessageHandler interface and DefaultMessageHandler class provide separate methods for each type of Message subclass. A user would typically subclass DefaultMessageHandler and override the handleMessage (*a Message subclass*) methods only for the messages that you want to process. Methods that are not explicitly overridden get passed to the subclass' handleDefaultMessage method. This DefaultMessageHandler subclass would typically be used in conjunction with a MessageHandlerAdapter.

To receive messages from the messaging bus, the programmer does the following:

- Creates a DefaultMessageHandler subclass that overrides the handleMessage(a Message subclass) methods corresponding to the messages of interest.
- Creates a MessageHandlerAdapter subclass that in its constructor simply passes a new instance of its DefaultMessageHandler subclass into the MessageHandlerAdapter constructor.
- Declares an extension of the com.ibm.collaboration.realtime.messages.MessageHandlerListener extension point, specifying the MessageHandlerAdapter created in the prior step.

### 3.4.4.2 User Interface Components

A portion of the code you will write for IBM Sametime Connect relates to integrating your application's functionality seamlessly into the user interface. The diagram below shows examples of where you can add your own contributions, such as actions to person and group objects, actions to the contact list action bar, and mini applications in the contact list window. As is often the case with well-designed object-oriented architectures, the business objects depicted in the user interface are directly reflected in the underlying programming model itself. Thus the screen images are annotated with an abbreviated name of the related programmer interface, whether it be a model object (Person, Group) or extension point (org.eclipseu.ui.popupMenus, org.eclipse.ui.actionSet, etc). Subsequent sections will reintroduce these interfaces and explain more about their use and relationships.




The MyBusinessCard view provides active status feedback, extended user information such as photos (if supported by the directory service), and the standard person menu choices.

Table listing MyBusinessCard classes.

| CLASS (REALTIME.PEOPLE) | DESCRIPTION |
|---|---|
| MyBusinessCard | Standard view of Person. |
| ChatActionDelegate, ChatActionWindowDelegate | Interfaces for a contributed action to the chat window. Used with the chatAction extension point. |
| ChatWindowAction | Abstract superclass for contributed chat window actions. Used with the chatAction extension point. |

To contribute actions to person or group objects, use the org.eclipse.ui.popupMenus extension point to create object contributions for the interfaces com.ibm.collaboration.realtime.livenames.PersonSelection or com.ibm.collaboration.realtime.livenames.GroupSelection. See the plugin.xml extension manifest in the com.ibm.collaboration.realtime.sample.snippets project for examples.

#### 3.4.4.2.1 UI-Specific Event Notification

IBM Sametime Connect uses the Eclipse-based JFace and SWT components to create its user interface. The client defines an action-related extension point that is similar to the org.eclipse.ui.popupMenus extension point in Eclipse, with the addition of some parameters specific to the chat window. For example, the chatAction extension point requires an implementor of ChatActionWindowDelegate or extension of ChatWindowAction be specified in the extension's class attribute, similar to Eclipse's IActionDelegate. The run method of this class includes a "handler", ChatWindowHandler, which defines methods you can use to manipulate the chat window. The Quick Response sample uses the handler's sendText method to insert the selected response into the transcript and forward it to the user's chat partner.

Most of the other UI-related event notifications follow the Eclipse and Java graphical programming model (for example, action delegate notifications, bean property change events, listener events). Refer to the *Suggested Reading* section (1.2) for more details.

## 3.5 Extension points summary

Table describing model-related extension points.

**MODEL-RELATED EXTENSION POINTS**

| PLUG-IN | EXTENSION POINT | DESCRIPTION | INTERFACES | NOTES |
|---|---|---|---|---|
| messages | MessageHandlerListener | Subscribe / receive events. Callback handlers can process message events before or after they are delivered to message handlers; optionally callback handlers can stop continued processing of an event. | MessageHandlerAdapter, MessageHandlerCallback<br><br>Also see:<br><br>MessageHandlerPreCallback, MessageHandlerPostCallback | Recent Buddies and Buddy Note demonstrate message handlers. Acronym Expander defines a callback hander. |

Table describing user interface-related extension points.

**USER INTERFACE RELATED EXTENSION POINTS**

| PLUG-IN | EXTENSION POINT | DESCRIPTION | INTERFACES | NOTES |
|---|---|---|---|---|
| chatwindow | chatAction | Contribute action to | ChatActionDelegate, ChatActionEventDelegate, | The Snippets sample defines |

| | | | | |
|---|---|---|---|---|
| | | main menu area, format toolbar, or buddy list. | ChatActionWindowDelegate<br><br>Also see:<br><br>ChatWindowHandler | several chat actions. |
| | nwayListExtension | Add column to table of n-way chat participants for displaying added information or responding to selection. | NwayTableCellSelectionListener, NwayTableLabelProvider | The Snippets sample defines added columns and cell selection listeners for the n-way chat participant list. |
| | popupAddOn | Contribute "temporary" area to chat window | PopupAddOn | Quick Response defines a pop-up add-on to show its list of responses. |
| org.eclipse.ui | org.eclipse.ui.popupMenus | Contribute action items to context menus of Person-like instances. | PersonSelection, GroupSelection<br><br>Also see:<br><br>GroupActionFilter, LiveNameActionFilter, LiveNameService, Group, Person | The Recent Buddies sample defines a Person-like class (RecentBuddy) that displays a standard Person context menu. |
| | org.eclipse.ui.views | Contributes to the application window area | ViewPart<br><br>Also see:<br><br>com.ibm.rcp.ui.shelfViews extension point | Recent buddies and Buddy Note samples define application window shelves. |

**Reminder:** See the extension points' schema definition for the complete list of valid elements and attributes. You can access them by selecting the Schema hyperlink for the selected extension point, or by opening the .exsd file found in the plug-in's schema subdirectory.

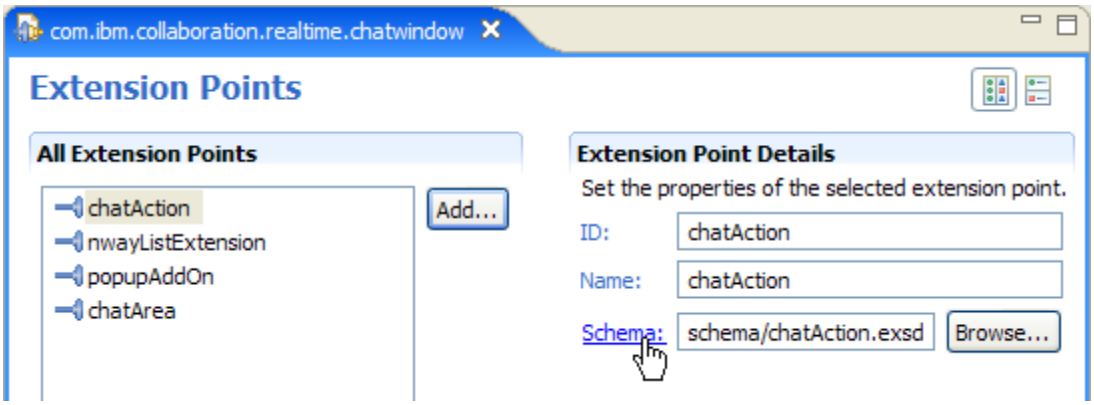



**Figure 6: Accessing the schema**

# Chapter 4. Extending IBM Sametime Connect By Example

The Recent Buddies plug-in captures and stores a list of most recent chat partners. This list of users is stored locally. Recent Buddies is shown through a pane in the shelf view area.

The following image shows an example of the Recent Buddies chat UI.

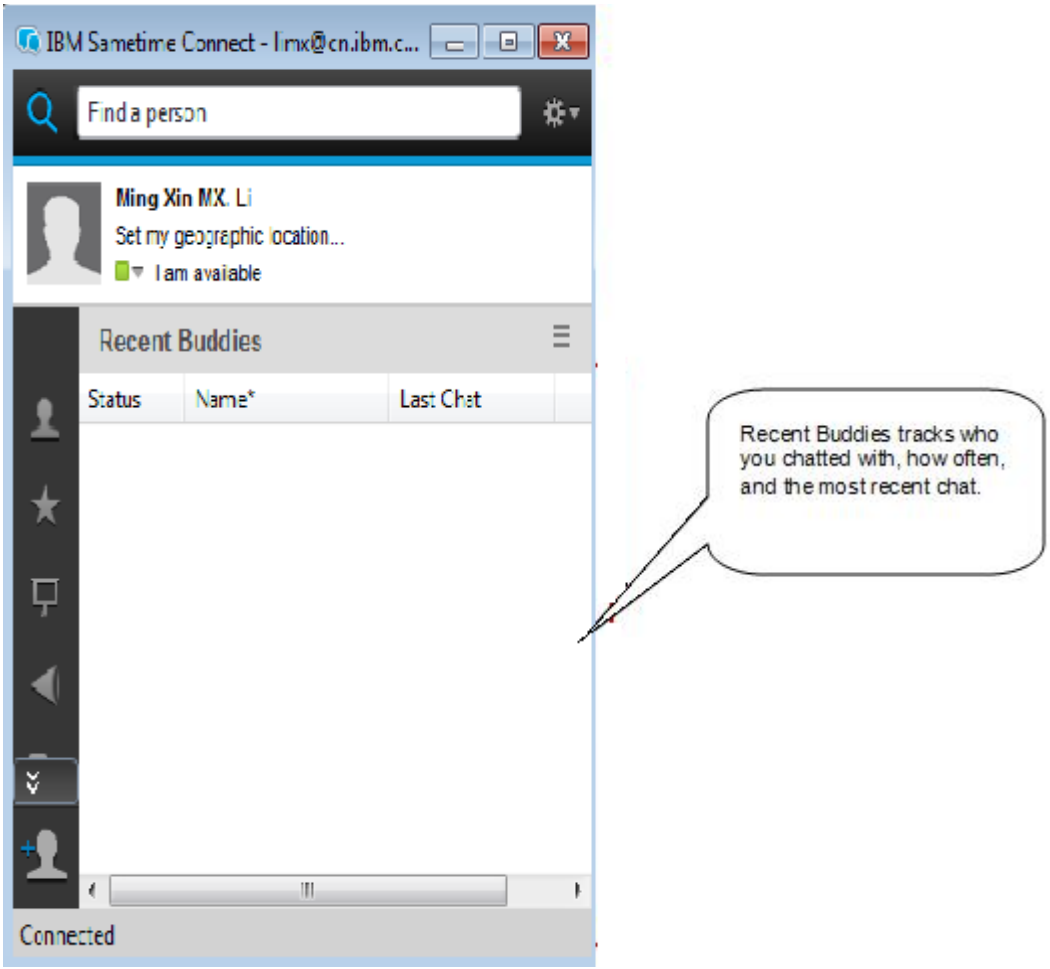



**Figure 7: Recent Buddies UI**

There are two recent buddy-specific options, **Primary** and **Remove**, that are accessible through a right-mouse click. There is also a third **Exclude** toggle option that is shown only if there are excluded buddies. Recent Buddies demonstrates

- Event notification
- Data storage
- Directory interface

The following section takes you through setting up your development environment so that you can work with Recent Buddies and other sample plug-ins, or create your own.

## 4.1 Development checklist

Before you can work with the Recent Buddies plug-in and the other sample plug-ins included in the IBM Sametime SDK, you'll need to make sure you have all the necessary components, and then set up your development environment.

Section 1.3 *Development requirements*, described the components required to work with the sample plug-ins or create your own. Before you can begin development, you'll need to complete each of the tasks listed in the table below and described in the following sections.

1. Download the IBM  Sametime 9.0 SDK.
2. Install Eclipse 3.4
3. Install the Lotus Expeditor (XPD) 6.2x Toolkit.
4. Install the Sametime Expeditor Toolkit Configuration
5. Configure the toolkit.
6. Import the sample plug-ins.
7. Create a launch configuration.

**Note:** You'll only have to perform the first 3 steps above once. However, the configuration you create in steps 4 - 6 is stored in your Eclipse workspace, and you will need to repeat those steps whenever you create a new workspace for IBM Sametime Connect plug-ins.

### 4.1.1 Download the IBM Sametime 9.0 SDK

If you haven't yet installed the SDK, you can download it from the IBM developerWorks Lotus toolkit download site at the following URL:

http://www.ibm.com/developerworks/lotus/downloads/toolkits.html

### 4.1.2 Install Eclipse 3.4

To work with the sample plug-ins or create your own, you'll need version 3.4 of the Eclipse IDE. You can download it from the following Eclipse Project Web site:

http://download.eclipse.org/eclipse/downloads/.

It's recommended that you use a J2SE 5.0 or higher JRE to run the Eclipse IDE. For information about downloading the IBM J2SE 5.0 JRE, see the following site:

http://www.ibm.com/developerworks/java/jdk

### 4.1.3 Install the Lotus Expeditor (XPD) Toolkit

In the Sametime 8.0 release, the IBM Sametime Connect client was re-based on the IBM Lotus Expeditor platform.  This ensured greater consistency between Lotus applications, such as IBM Notes® and Lotus Expeditor, in terms of runtime platform, serviceability, programming model, and enhanced administration, installation and policy management.  Part of this re-basing also included a shift in development tooling.  The IBM Lotus Expeditor Toolkit is now the recommended tool for Sametime

development as it provides a rich set of tools and greatly reduces the complexity of the development environment setup.

IBM Lotus Expeditor Toolkit provides a complete, integrated set of tools that allows you to develop, debug, test, package, and deploy client applications to IBM Lotus Expeditor V6.2.x, IBM Sametime 8.x, and IBM Notes 8.x.

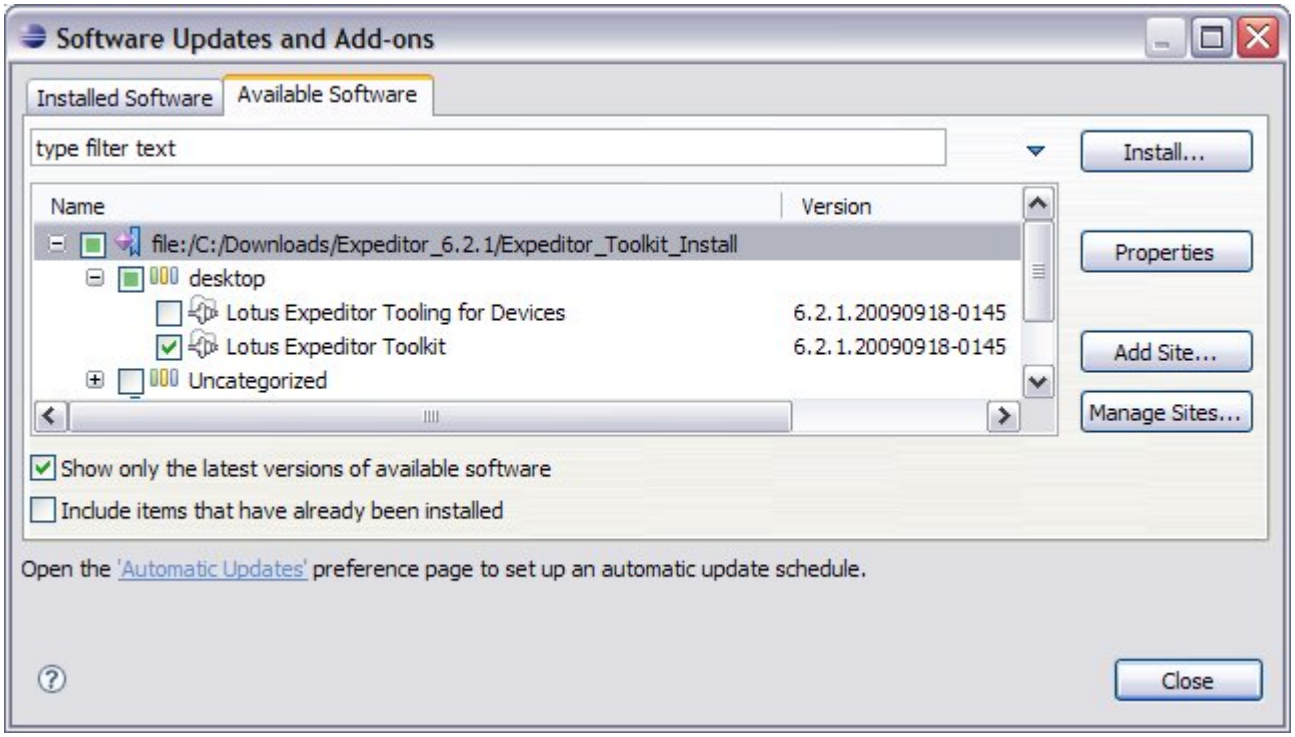
If you haven't yet installed the Lotus Expeditor Toolkit, you can download it from the IBM developerWorks Lotus toolkit download site at the following URL:

http://www.ibm.com/developerworks/lotus/downloads/toolkits.html

**Note:** Development for Sametime 9 requires the XPD 6.2.x Toolkit. At the time of the Sametime 9 release, the XPD 6.2, 6.2.1, 6.2.2 and 6.2.3 toolkits have all been tested and verified to work with Sametime 9 Connect development.

After you download the toolkit ZIP file, unzip the file in a temporary location. To access the toolkit installation instructions, release notes, and other documentation, open the autorun.html file in your browser of choice.

1. Start Eclipse if it isn't running.
2. Select Software Updates from the Eclipse Help menu.
3. Choose the "Available Software" tab.
4. Click "Add Site" option and navigate (using the "Local..." button) to the Expeditor_Toolkit_install directory where you unzipped the Expeditor download and click OK .
5. Select the Lotus Expeditor Toolkit feature under the "desktop" category and click "Install...".



6. Verify the install, Accept the license and then click Finish.

7. Accept the IBM / Lotus security certificate if prompted.

When the installation process is complete, you are prompted to restart the platform. Click OK.

### 4.1.4 Install the Sametime Expeditor Toolkit Configuration

After you restart the IDE platform, you are presented with the toolkit configuration dialog box again. This time click the cancel option since the Sametime Configuration must be installed.

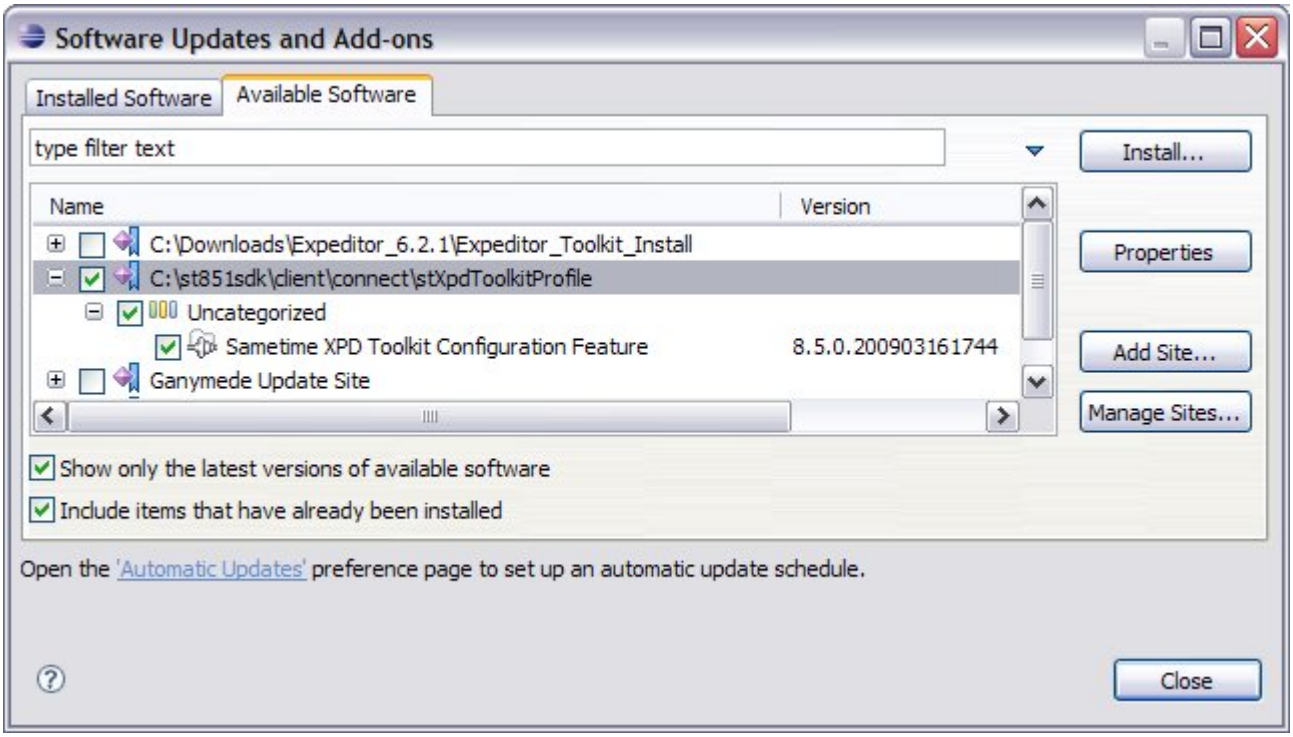
Follow these steps to install the Sametime Configuration

1. Select Software Updates from the Eclipse Help menu.

2. Choose the “Available Software” tab.

3. Click "Add Site" option and navigate (using the “Local...” button) to the client\connect\stXpdToolkitProfile subdirectory of the st9sdk directory in which you installed the IBM Sametime 9.0, for example, C:\st9sdk\client\connect\stXpdToolkitProfile and click OK

4. Ensure that the Sametime XPD Toolkit Configuration Feature is the only item checked and click “Install...”



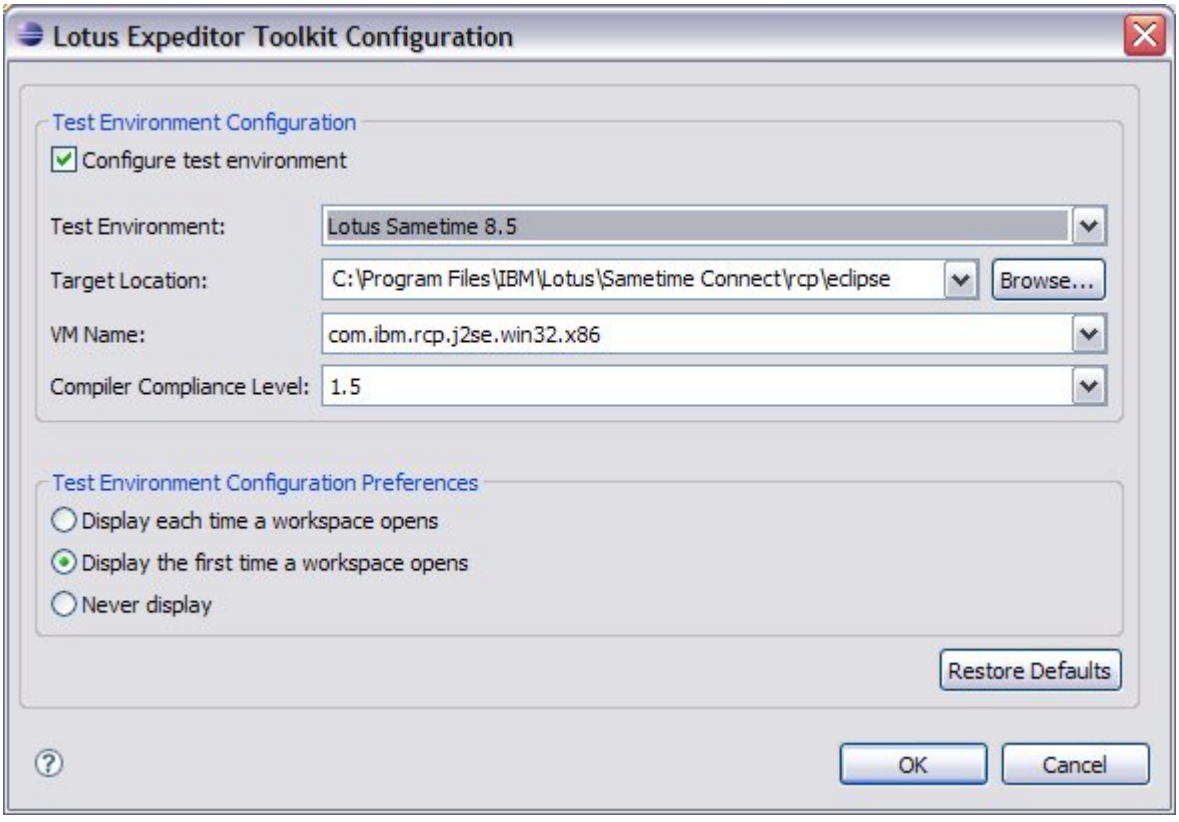

5. Again verify the install, accept the license agreement and click Finish.

### 4.1.5 Configure the toolkit

After you restart the IDE platform, you are presented with the toolkit configuration dialog box again. This time you will have the option to select IBM Sametime Connect after you've completed the steps below.

Follow these steps to configure the toolkit.

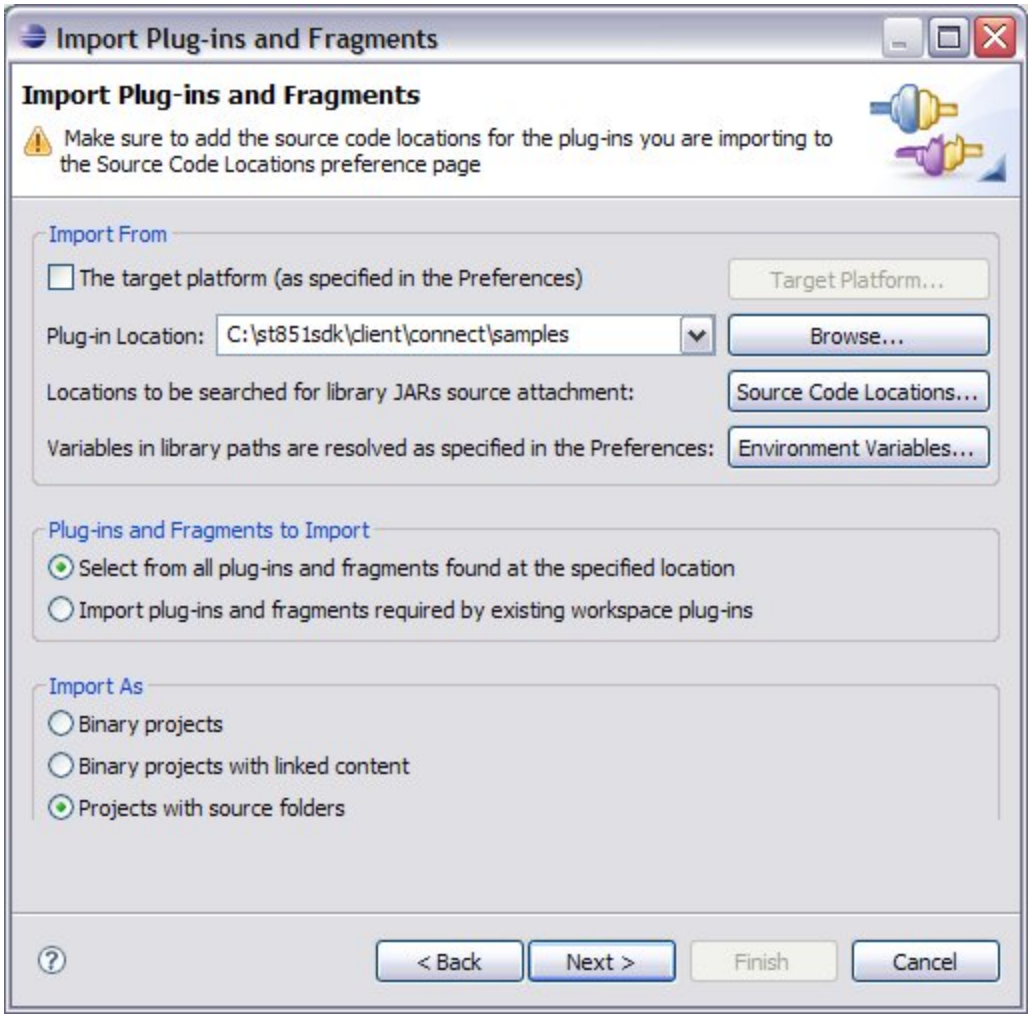
1. Select Lotus Sametime 8.5 in the Test Environment drop-down box.
2. Click Browse, and then select the rcp\eclipse directory underneath the IBM Sametime Connect installation directory (For example: C:\Program Files\IBM\Sametime Connect\rcp\eclipse) in the Target Location field and click OK.
3. Leave the default selected VM and Compiler Compliance Level.
4. Click OK.
5. It is also recommended that you select the option "Attempt to configure the toolkit the first time a workspace opens" in the Auto-Configuration Preference dialog box. This selection causes the configuration dialog box to be presented only the first time the toolkit is used in a workspace. You can change this preference later in any workspace by choosing Windows – Preferences – Client Services. When you click OK, the IDE is reconfigured for development against IBM Sametime 9.



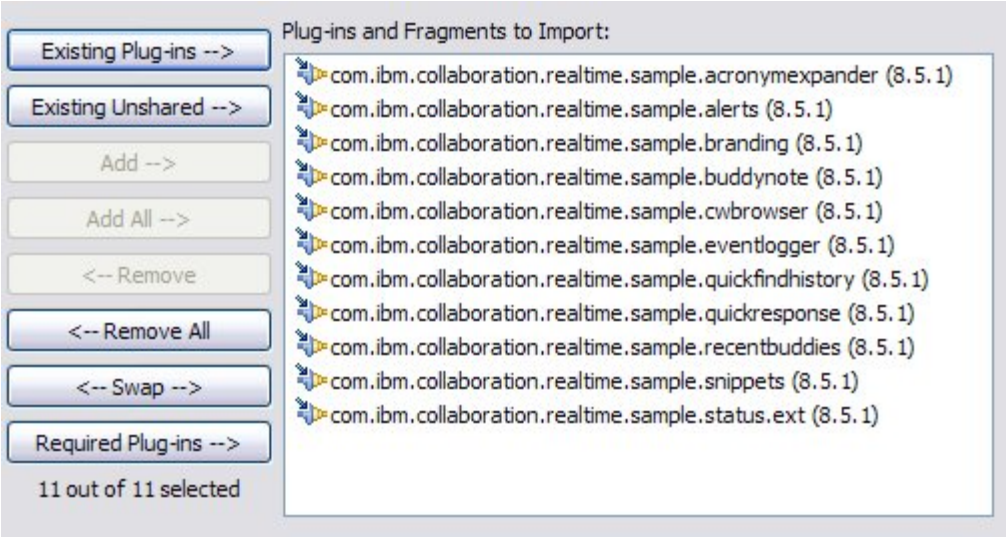

### 4.1.6 Import the sample plug-ins

Now that you've configured the runtime environment and target platform, you're ready to import the sample plug-ins. Follow the steps below to do so.

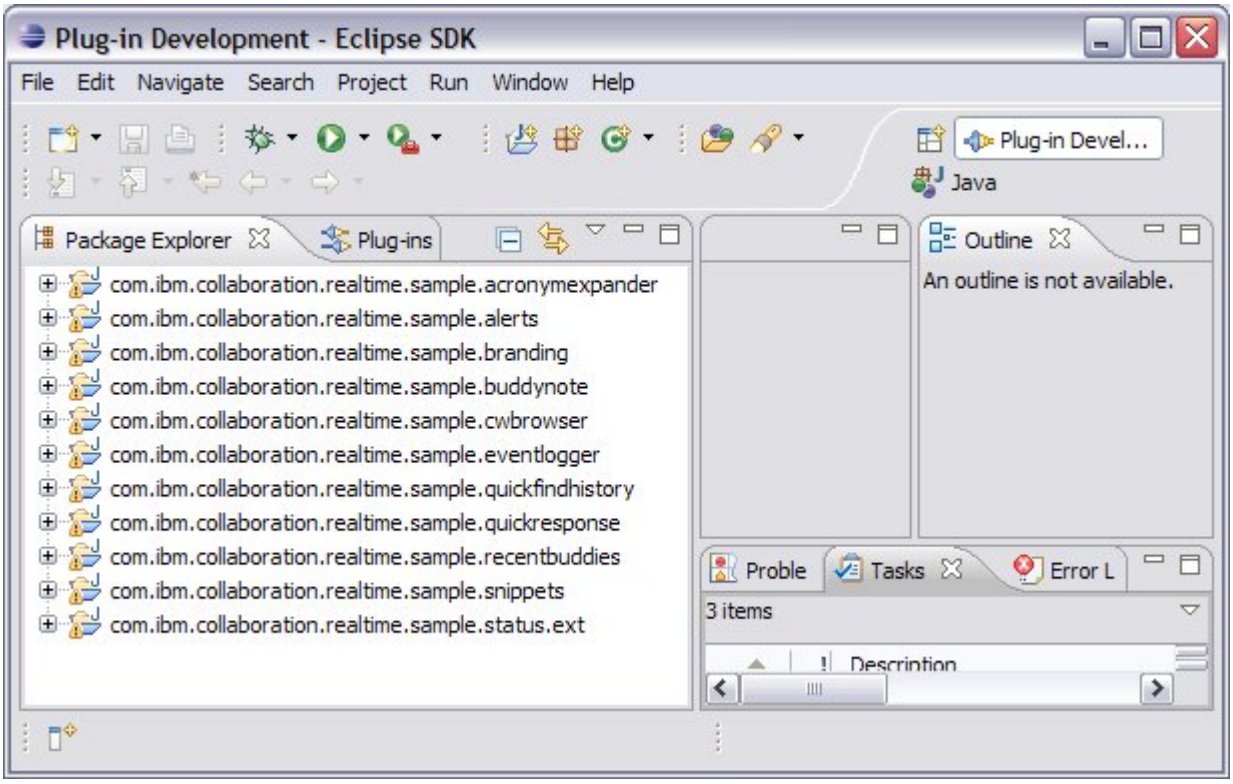
1. Start Eclipse if it isn't running.
2. Open the plug-in development perspective using the plug-in icon on the perspective bar or by clicking **Window > Open Perspective > Other > Plug-in Development.**
3. **Click File > Import... > Plug-in Development > Plug-ins and Fragments** to import the sample plug-ins from the IBM Sametime SDK to your development environment.
4. Click Next.
5. In the **Import From** field, de-select "**The target platform"** and browse to the client connect samples subdirectory of the st9sdk directory in which you installed the IBM Sametime 9.0 SDK , for example, C:\st9sdk\client\connect\samples.
6. In the Import As field, select Projects with source folders. Your screen should look like the one below:




7. Click **Next**.
8. Click **Add All** to import all the sample plug-ins, then click **Finish**.




**Result**: The plug-ins for IBM Sametime are added to your workspace, which should look like the one below.




### 4.1.7 Create a launch configuration

The last step you need to perform to set up your development environment is to create a launch configuration. The launch configuration is used to run or debug the plug-ins using the IBM Sametime Connect target platform you created earlier.

Follow the steps below to create a launch configuration.

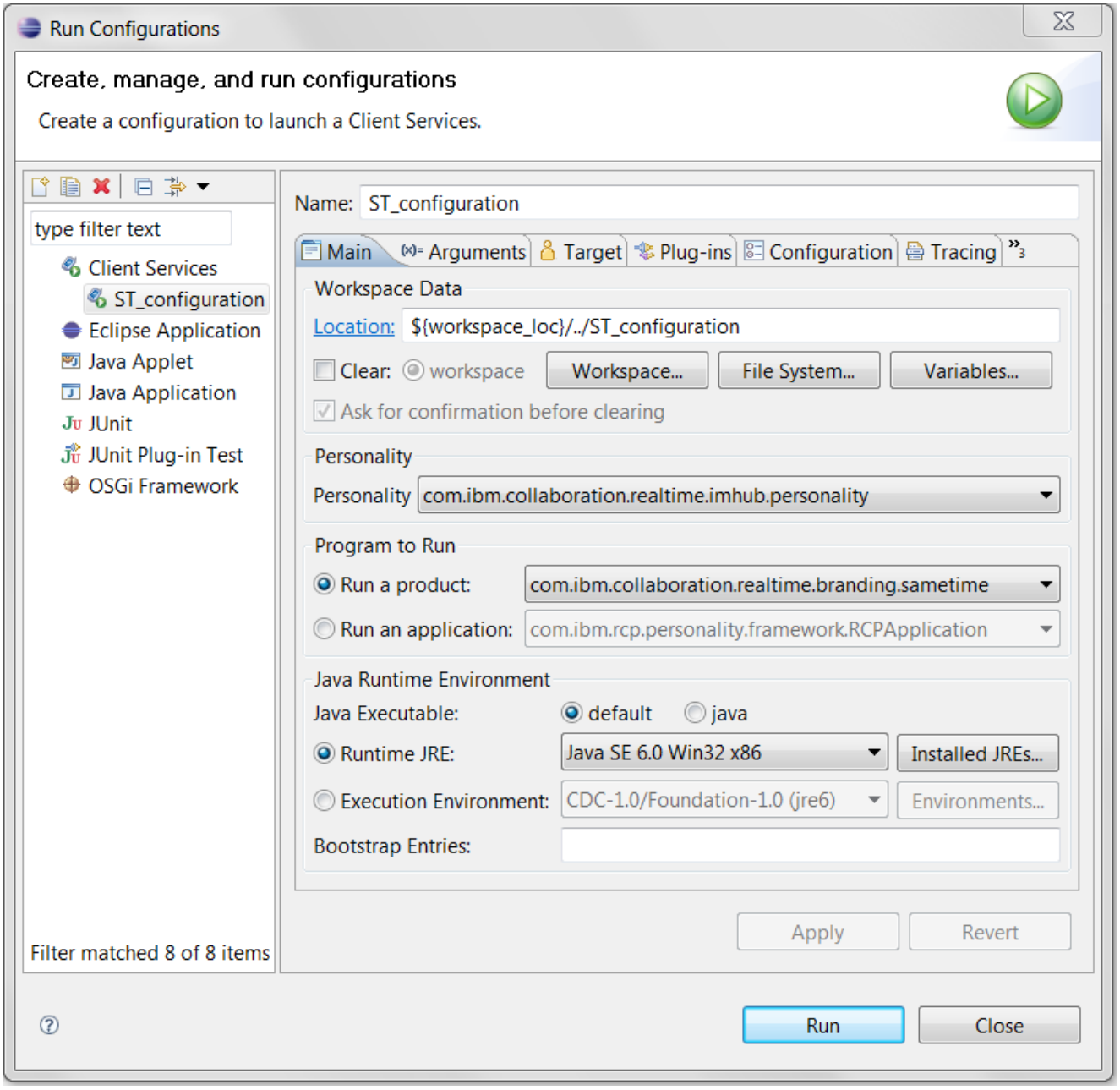
1. Start Eclipse if it isn't running.

2. In the plug-in perspective, click **Run > Run Configurations…** in order to configure the runtime environment.
3. Select **Client Services** on the configuration list.
4. Click the **New** icon (the leftmost icon above the configuration list).
5. Replace the default configuration name in the **Name** field with a new one, for example, **ST_Configuration**
6. Replace the default location in the **Location** field with a new workspace, for example, **${workspace_loc}/../ST_Configuration**

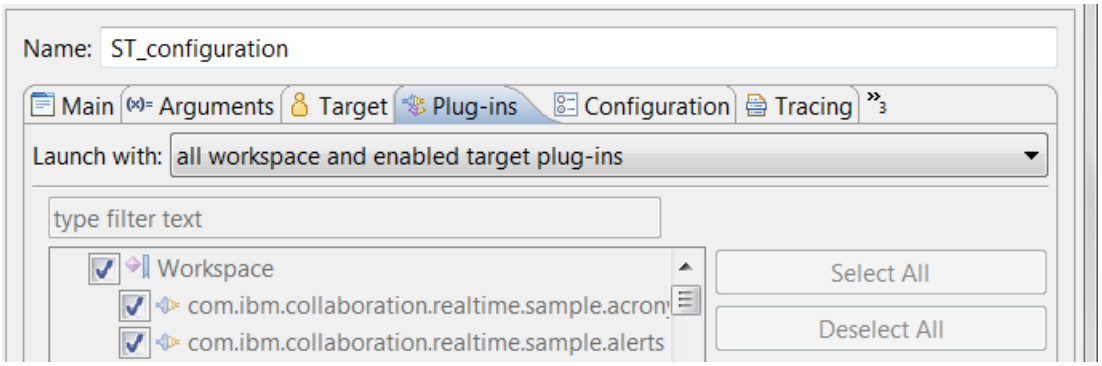
   **Note:** The workspace you specify here is the one that IBM Sametime Connect will use to store product settings when you launch it from Eclipse. This is not the same workspace that IBM Sametime Connect uses when you launch it outside of the Eclipse IDE. You should not select the same workspace directory that you are using for Eclipse configuration settings (the one you selected in previous steps).

7. All settings necessary for Sametime are now pre-populated.

   Your screen should look like the one below:




8. Select the **Plug-ins** tab and make sure that **Launch with all workspace and enabled external plug-ins** is selected.



9. Click **Apply** to save your changes, then click **Run** to launch the configuration.

   **Result:** IBM Sametime Connect is launched from the plug-in development environment of Eclipse with the sample plug-ins you imported earlier. To run the plug-ins in debug mode, use **Run > Debug…** to launch the configuration from the Eclipse main menu.

## 4.2 Planning a plug-in

Before you jump into building a plug-in, it helps to plot out a few things:

- Determine what you want the plug-in to be able to do.
- Find out if there are already extension points that your plug-in can hook into to perform these tasks.
- Determine if these extension points are Eclipse based or part of the APIs for IBM Sametime Connect.

You should also keep in mind that with Eclipse, you should separate UI functionality from data handling. For example, in creating the Recent Buddies plug-in, we considered some of the following questions:

- Where should the Recent Buddies functionality appear?
- What kinds of information about my buddies should appear in the Recent Buddies area?
- What kinds of actions can I perform with the person names in Recent Buddies? Which actions are standard and which are unique to Recent Buddies?
- How will the application know the user has initiated or received a chat?
- Where will the Recent Buddies list be stored?
- How will the Recent Buddies list be retrieved?
- What is Recent Buddies' memory and speed impact?

In all the excitement over the extensibility of IBM Sametime Connect, it is easy to forget the last point: What is the cost of all these new plug-ins? Recall that “IM” stands for instant messaging and responsiveness is critically important. When designing a plug-in you should also take into account the potential load it may place on your user’s computer and their network infrastructure.

Load may occur from a variety of things. Plug-ins that are mainly UI code produce little or no network load. But if they use many special libraries they may increase the memory footprint. Good coding practice and simple “before and after” examination of memory use can allow you to track implications. Chat messages themselves are quite light in terms of network load. But if you are automating lots of file or image transfers, you may be creating network stress conditions. At design time it is good to ask yourself “what if everyone did this at once?”

## 4.3 Create the Recent Buddies plug-in

In order to create a new plug-in you need to perform the following steps:

1. Create the plug-in.

2. Add the required dependencies.
3. Declare the extensions and extension points.
4. Write the classes that implement the interface or extend the abstract class defined by the extension point.
5. Install and test the plug-in.

The following outlines the general steps we followed to create the Recent Buddies plug-in.

1. Launch the Eclipse plug-in development platform.
2. Click File > New > Project > Plug-in Project > Next.
   **Result:** The New Plug-in Project wizard opens.
3. Type **com.ibm.collaboration.realtime.sample.recentbuddies** in the Project name field. Leave the rest of the default choices.
4. Click **Next**.
   **Result:** The Plug-in Content window opens.
5. Type a descriptive name in the **Plug-in Name** field. Keep the rest of the default values.
   **Note**: The sample plug-ins use the naming conventions followed by the Sametime development team, which is why the plug-in id begins with com.ibm.collaboration.realtime.
6. Click Finish.
   **Result:** The Overview panel opens.

## 4.4 Adding the Recent Buddies plug-in dependencies

Here are the noteworthy plug-in dependencies that the Recent Buddies plug-in uses:

Table listing plug-in dependencies used by the Recent Buddies plug-in.

| PLUG-IN | DESCRIPTION |
|---|---|
| *core | Defines core classes like ServiceHub. |
| *directory | Directory services for retrieving person-related information. |
| *imhub | Defines Sametime's Contact list and main window that contains the area for additional applications. |
| *messages | Manages message handler events. |
| *people | Listens for on-line people status. |
| com.ibm.rcp.realtime.livenames | Defines interfaces for enabling interactive chat menu actions of Person-like instances. |

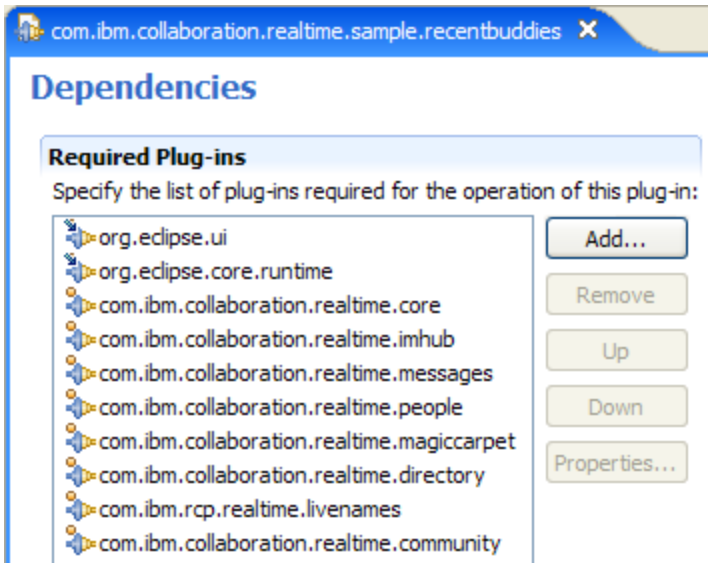
**Note:** To save space,the asterisk (*) is used to replace the first part of long package names such as the com.ibm.collaboration.realtime prefix.

These steps outline how to add the dependencies required for the Recent Buddies plug-in and to declare its extensions.

1. Click the **Dependencies** tab of the Recent Buddies plug-in manifest.
2. Click **Add**.

3. Add the following plug-ins:
   org.eclipse.core.runtime, org.eclipse.ui, *core, *imhub, *messages, *magiccarpet (implementation-only classes and interfaces), *directory, *community, *people, com.ibm.rcp.livenames

   **Result**: The screen should look like the following image:




4. Click the **Extensions** tab.
5. Click **Add**.
6. Add the following extensions:
   *core.serviceListeners
   *messages.MessageHandlerListener
   com.ibm.rcp.ui.shelfViews
   org.eclipse.ui.views
   org.eclipse.ui.popupMenus
7. Click **Save** to save the plug-in at this point.

## 4.5 Key Recent Buddies classes

Table listing key Recent Buddies classes.

| CLASS | FUNCTION |
|---|---|
| RecentBuddiesView | Adds the Recent Buddies view to the available shelves when Sametime is started. This class also determines what information is displayed and how it is laid out as well as the right-click actions that can be performed on members of this list. |
| RbMessageHandlerAdapter | Handles and processes the messages from the notification service. |

## 4.6 Contributing a view shelf to the main window

The views class answers the question of how the application will look and where it appears. It also needs to keep track of the status of chat partners who are on the Recent Buddies list.

RecentBuddiesView is referred to in the Recent Buddies plug-in extension manifest as follows:

```
<extension
   point="org.eclipse.ui.views">
   <view
      allowMultiple="false"
      class="com.ibm.collaboration.realtime.sample.recentbuddies.ui.RecentBuddiesView"
      name="Recent Buddies"
      icon="images/ST_Ext_Recent_Buddies.png"
      id="com.ibm.collaboration.realtime.sample.recentbuddies.ui.RecentBuddiesView"/>
</extension>
```

com.ibm.collaboration.realtime.sample.recentbuddies.ui.RecentBuddiesView extends the ViewPart class as required by the shelfViews extension point.

ShelfView is defined in the following code:

```
<extension
   point="com.ibm.rcp.ui.shelfViews">
   <shelfView
      id="com.ibm.collaboration.realtime.sample.recentbuddies.shelfview"
      view="com.ibm.collaboration.realtime.sample.recentbuddies.ui.RecentBuddiesView"
      region="BOTTOM"
   showTitle="true"/>
</extension>
```

The RecentBuddiesView is primarily composed of a JFace TableView control which will be populated with the names of recent IBM Sametime Connect chat partners. The JFace table view uses "helper" classes that handle common code such as mapping between model classes (for example, RecentBuddy) and the strings and images expected by the SWT's Table control. The following code defines the content and label providers we will use with this. It has been abbreviated for space reasons:

---

```
private class BuddiesTableContentProvider implements
      IStructuredContentProvider {
   public Object[] getElements(Object inputElement) {
      return ((RecentBuddyList) inputElement).getRecentBuddies().toArray();
   }

   public void dispose() {}
   public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {}
}

private class BuddiesTableLabelProvider implements ITableLabelProvider {
   public Image getColumnImage(Object element, int columnIndex) {
      if (columnIndex == RecentBuddyListConstants.STATUS_COLUMN)
            return getDefaultPersonaImage();
      else
            return null;
   }

   public String getColumnText(Object element, int columnIndex) {
      RecentBuddy rb = (RecentBuddy) element;

      if (columnIndex == RecentBuddyListConstants.NAME_COLUMN){
            return rb.getNickname();
      } else if (columnIndex == RecentBuddyListConstants.LAST_CHAT_COLUMN) {
            if (rb.getLastChatTime() != 0) {
                  return DateFormat.getInstance().format(
                        new Date(rb.getLastChatTime()));
            } else {
                  return "";
            }
      } else if (columnIndex == RecentBuddyListConstants.CHAT_COUNT_COLUMN) {
                  return Integer.toString(rb.getChatCount());
      }

      return null;
   }

   public boolean isLabelProperty(Object element, String property) {
      return true;
   }

   public void removeListener(ILabelProviderListener listener) {}
   public void addListener(ILabelProviderListener listener) {}
   public void dispose() {}
}
```

---

The following code creates and populates the primary table control. Most of the details are in RecentBuddiesView's private methods, which can be reviewed in the source code provided with the IBM Sametime SDK. The excerpt below gives an outline of the steps involved.

```
public void createPartControl(Composite parent) {
      Composite comp = new Composite(parent, SWT.NONE);
      comp.setLayout(new FormLayout());
      createTable(comp);
   }

   private void createTable(Composite parent) {
      if (buddiesTableCtl != null)
            buddiesTableCtl.dispose();

      buddiesTableViewer = new TableViewer(parent, SWT.MULTI
                  | SWT.FULL_SELECTION | SWT.VERTICAL);
      buddiesTableCtl = buddiesTableViewer.getTable();

      // Recent Buddies uses a table to represent a group
      // of Person instances. MyBusinessCard is useful
      // for representing just one user (like BuddyNote).
      createTableColumns();
      createTableContextMenu();
      createTableHelpers();
      setTableLayout();
      addTableListeners();
      updateSortIndicator();


      buddiesTableViewer.setInput(RecentBuddyList.getInstance());
      parent.layout();
   }
```

## 4.7 How to be notified of important events

Recent Buddies' MessageHandlerAdapter class, RbMessageHandlerAdapter, subscribes to messages from the message bus and is specified within the extension manifest as the class associated with the *messages.MessageHandlerListener extension point.

RbMessageHandlerAdapter is referred to in the Recent Buddies plug-in extension manifest as follows:

```
<extension
   point="com.ibm.collaboration.realtime.messages.MessageHandlerListener">
   <messageHandler
   class="com.ibm.collaboration.realtime.sample.recentbuddies.RbMessageHandlerAdapter"/>
</extension>
```

The MessageHandlerAdapter works in conjunction with a DefaultMessageHandler subclass. This class defines handleMessage methods for each type of Message subclass; you intercept event messages by overriding the corresponding handleMessage methods.

In the Recent Buddies plug-in, the DefaultMessageHandler subclass is called RbMessageHandler.

The MessageHandlerAdapter class, RbMessageHandlerAdapter looks like the following code:

```
import com.ibm.collaboration.realtime.messages.MessageHandler;
import com.ibm.collaboration.realtime.messages.MessageHandlerAdapter;

public class RbMessageHandlerAdapter extends MessageHandlerAdapter {

   public RbMessageHandlerAdapter(MessageHandler handler) {
      super(handler);
   }


   public RbMessageHandlerAdapter() {
      super(new RbMessageHandler());
   }
}
```

In most cases, the handler adapter code is as shown above with only the name of the instantiated handler, RbMessageHandler, changing to a specific class that will process selective incoming message events. In a more complex implementation, the adapter would be a good place for event-related initialization code. The DefaultMessageHandler subclass, RbMessageHandler, receives and processes the messages you are interested in. The following code sample is an abbreviated version of the subclass:

```
public class RbMessageHandler extends DefaultMessageHandler {
   public void handleDefaultMessage(Message message) {
      // do nothing
   }

   public void handleMessage(ImTextConnectionOpenSuccessMessage message) {
      //...
}

public void handleMessage(ManyToManyUserEnterExitMessage message) {
      //...
}

public void handleMessage(PartnerStatusUpdateMessage message) {
      //...
}

// cont'd
```

The DefaultMessageHandler has no-op handleMessage(…) implementations whose parameter indicates the specific message; as an extender of the *messages.MessageHandlerListener extension point, your subclass can override these methods to react to important events within IBM Sametime Connect.

## 4.8 Overview of the plug-in manifest

Eclipse plug-ins are located in their own directory within the Eclipse installation. Each plug-in has two files that define how the plug-in works to the platform runtime:

- plugin.xml
- MANIFEST.MF

A plug-in extension manifest is broken into four parts that may or may not all be used by the plug-in:

- Dependencies (MANIFEST.MF)
- Runtime (MANIFEST.MF)

- Extensions (plugin.xml)
- Extension Points (plugin.xml)

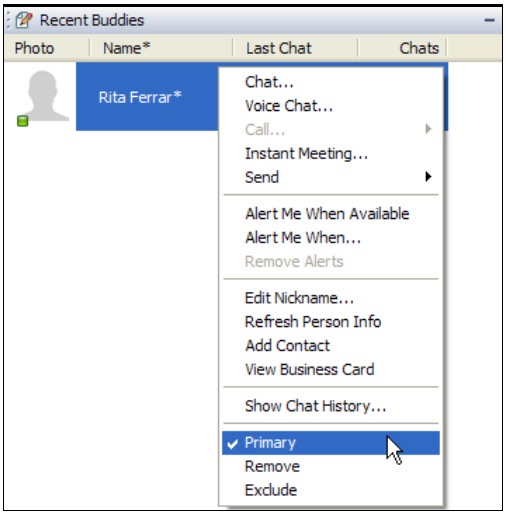
Although these specifications are stored in separate files, for the plug-in developer's convenience they are shown in an editor with pages corresponding to major sections. The Eclipse Plug-in Development Environment manifest editor updates the correct file based on the programmer's modifications, using the editor or its code-completion wizards.

## 4.9 Modify the Recent Buddies plug-in extension manifest file

The following code shows part of the Recent Buddies plug-in extension manifest once all of the extension classes have been added.

```
<plugin>
   <extension
      point="org.eclipse.ui.views">
      <view
            allowMultiple="false"
            class="com.ibm.collaboration.realtime.sample.recentbuddies.ui.RecentBuddiesView"
            name="Recent Buddies"
            icon="images/ST_Ext_Recent_Buddies.png"
            id="com.ibm.collaboration.realtime.sample.recentbuddies.ui.RecentBuddiesView"/>
   </extension>

   <extension
      point="com.ibm.rcp.ui.shelfViews">
      <shelfView
            id="com.ibm.collaboration.realtime.sample.recentbuddies.shelfview"
            view="com.ibm.collaboration.realtime.sample.recentbuddies.ui.RecentBuddiesView"
            region="BOTTOM"
            showTitle="true"/>
   </extension>

   <extension
      point="com.ibm.collaboration.realtime.messages.MessageHandlerListener">
      <messageHandler
   class="com.ibm.collaboration.realtime.sample.recentbuddies.RbMessageHandlerAdapter"/>
   </extension>

   <extension
      point="org.eclipse.ui.popupMenus">
      <objectContribution
            adaptable="false"
            id="com.ibm.collaboration.realtime.sample.recentbuddies.personselections"
            objectClass="com.ibm.collaboration.realtime.livenames.PersonSelection">
// cont'd
```

The org.eclipse.ui.popupMenus extension above specifies an object-specific menu action contribution to PersonSelection. The RecentBuddy instances shown in the Recent Buddies table implement the PersonSelection interface, thus the registered context menu of its table will display the standard Person-related menu choices as well as the three Recent Buddies-specific choices:




By default, menu actions added via org.eclipse.ui.popupMenu object contributions are added wherever an object fulfilling the specified objectClass interface appears (in this case PersonSelection). To limit the display of the contributed Primary, Remove, and Exclude choices to the Recent Buddies list, the <filter name="context" value="RecentBuddies"/> element was added to the extension definition and a getAdapter method added to the RecentBuddy class:

```
public Object getAdapter(Class adapterClass) {
  Object adapter = null;
  if (adapterClass.equals(IActionFilter.class)) {
      LiveNameService svc = (LiveNameService)
      ServiceHub.getService(LiveNameService.SERVICE_TYPE);
      adapter = svc.getLiveNameActionFilter("RecentBuddies");
  }
  return adapter;
}
```

As explained in the IActionFilter online help accompanying the Eclipse SDK, the workbench will retrieve the filter from the selected object by testing to see if it implements IActionFilter. If that fails, the workbench will ask for a filter through the IAdaptable mechanism. Since RecentBuddy implements IAdaptable.getAdapter, it has the opportunity to provide an action filter initialized with its context ("RecentBuddies" parameter in the svc.getLiveNameActionFilter invocation above). The action filter returned by the RecentBuddy.getAdapter method confirms the context and the contributed menu actions are included; objects other than RecentBuddy that also implement PersonSelection, such as Person instances shown in the contact list, will not return an action filter initialized with the "RecentBuddies" context and therefore will not show these contributed menu actions.

See *How do I use IAdaptable and IAdapterFactory?* In the Eclipse Official FAQs for more details.

# Chapter 5. A quick how-to guide

Once you have seen how to build an initial plug-in for IBM Sametime Connect, you might want to take a look at some of the other plug-in samples. Like the Recent Buddies plug-in, these samples might be more basic than you need; however, they provide a solid look into how to implement some frequently requested tools. A lot of these samples can be found in the Snippets sample plug-in

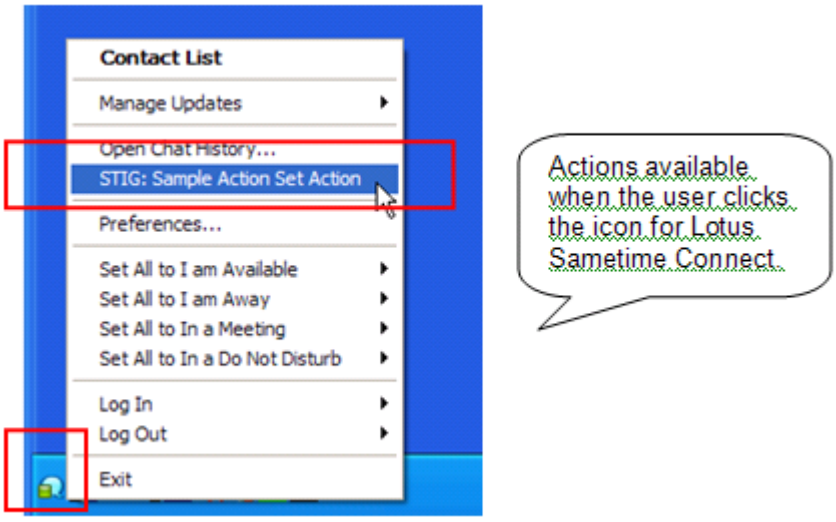
The Snippets plug-in exposes some of the following functionality:

- Adding an action to the system tray icon
- Adding right mouse-click actions to a selected person or group
- Adding menu items to the chat window action bar
- Adding a toolbar action to the contact list window
- Adding a toolbar action to the chat window.

## 5.1 Adding an action to the IBM Sametime Connect system tray icon

The following code adds a new action that is generated when a user right-clicks the icon for IBM Sametime Connect. The icon is displayed in the system tray, typically near the time and date display in the bottom right corner of the screen.

The following image shows the system tray action, named **STIG: Sample Action Set Action**.



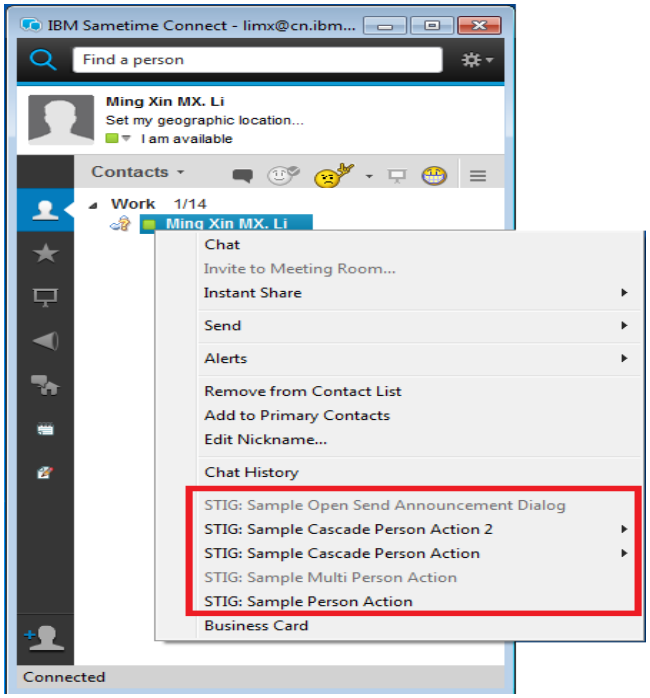

The following plug-in manifest excerpt shows how to add to the action set.

```
<!-- action set must have com.ibm.collaboration.realtime prefix to appear among system tray
choices -->
<extension
   point="org.eclipse.ui.actionSets">
   <actionSet
      description="STIG: Actions for Sample Plug-in"
      id="com.ibm.collaboration.realtime.sample.snippets.actionset"
      label="STIG: Sample Action Set"
      visible="false">
      <action

   class="com.ibm.collaboration.realtime.sample.snippets.actions.SampleActionSetAction"
   id="com.ibm.collaboration.realtime.sample.snippets.actions.sampleactionsetaction"
   label="STIG: Sample Plug-in Action"
   menubarPath="sametime/plugins"
   tooltip="STIG: Show Sample plug-in action"/>
   </actionSet>
</extension>
```

## 5.2 Adding right-mouse click actions to a selected person or group

You can add actions to the existing set of selected partner menu options using the com.ibm.rcp.realtime.livenames.LiveNameSelection extension.

The following image shows an example of person or group actions. The custom-built actions are preceded by **STIG:**



The following extension shows how to add actions to the context menu of a chat partner:

```
<extension
   point="org.eclipse.ui.popupMenus">
   <objectContribution
      adaptable="false"
      id="com.ibm.collaboration.realtime.sample.snippets.livenamecontribution"
      objectClass="com.ibm.rcp.realtime.livenames.LiveNameSelection">
      <action
            class="…sample.snippets.actions.SampleLiveNameActionDelegate"
            enablesFor="1"
            id="com.ibm.collaboration.realtime.sample.snippets.actions.samplepersonaction"
            label="STIG: Sample Person Action"
            style="push"/>
      <action
            class="… sample.snippets.actions.SampleMultiLiveNameActionDelegate"
            enablesFor="2+"

   id="com.ibm.collaboration.realtime.sample.snippets.actions.samplemultipersonaction"
            label="STIG: Sample Multi Person Action"
            style="push"/>
// omitted for brevity
   </objectContribution>
</extension>
```

## 5.3 Adding a toolbar action to the contact list window(Embedded Only)

You can add icons to the main user interface; the tools for this are provided by standard Eclipse extensions.

The toolbar consists two types of actions: primary actions and secondary actions. Primary actions are placed in the primary toolbar location as buttons. Secondary or less frequently used actions are displayed as menu items when user opens the "More" drop down button.

Actions will be added to toolbar as primary actions by default. A new attribute "isSecondary" is introduced to define a secondary action in plugin.xml. If “isSecondary” is set to true, that action will be placed in the “More” drop down menu.

The following image shows two toolbar contribution examples: one action button is added to the toolbar primary action location; another one is added to the “More” drop down menu as a secondary action.

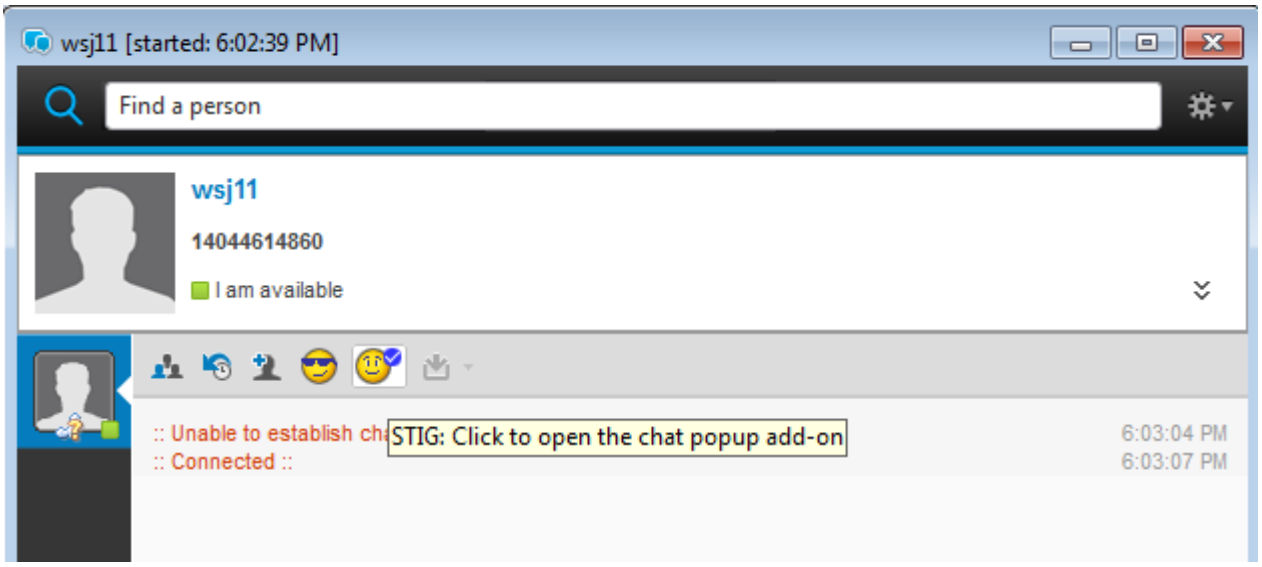
The following extension shows how to contribute an action to the contact list primary and secondary toolbar location.

```
<extension
point="org.eclipse.ui.viewActions">
    <viewContribution
         id="com.ibm.collaboration.realtime.sample.snippets.viewactions"
         targetID="com.ibm.collaboration.realtime.imhub">

         <action
class="com.ibm.collaboration.realtime.sample.snippets.actions.SampleBuddyListCounterAction"
              enablesFor="0"
              icon="images/biggrin.gif"
              disabledIcon="images/biggrinOff.png"
       id="com.ibm.collaboration.realtime.sample.snippets.actions.samplebuddylistdelegate"
```

```
            label="STIG: SampleBuddyListDelegate"
            style="push"
            isSecondary="true"
            toolbarPath="buddylist/secondary/additions"
            tooltip="STIG: SampleBuddyListCounterAction / SampleBuddyListDelegate"/>

         <action
      class="com.ibm.collaboration.realtime.sample.snippets.actions.SampleBuddyListAction"
            disabledIcon="images/EmoticonThumbsUpOff.png"
            enablesFor="2"
            icon="images/EmoticonThumbsUp.gif"
      id="com.ibm.collaboration.realtime.sample.snippets.actions.samplebuddylistaction"
            label="STIG: SampleBuddyListAction"
            style="push"
            toolbarPath="buddylist/additions"
            tooltip="STIG: Select two people to do something (SampleBuddyListAction)"/>

    </viewContribution>
</extension>
```

## 5.4 Adding a toolbar action to the chat window

You can use the *chatwindow.chatAction extension to add more tool bar options to the chat window user interface. The following image shows an example of a new toolbar that opens the chat area, preceded with **STIG**.




The following extension creates a pop-up add-on, which is displayed beneath the chat window send area (file transfer uses this feature to show its progress; Quick Response uses it to show a list of possible choices).

```
<extension
   point="com.ibm.collaboration.realtime.chatwindow.popupAddOn">
   <popupAddOn
      class="com.ibm.collaboration.realtime.sample.snippets.ChatPopupAddOn"
      id="com.ibm.collaboration.realtime.sample.snippets.chatpopupaddon"
      name="STIG: Sample Popup Addon"/>
</extension>
```

# Chapter 6. Sample plug-ins

This chapter presents the sample plug-ins that are included in the IBM Sametime SDK.

The following table, which you saw in section 2.5  lists the plug-in samples as well as what they do and the extensions they use.

Table listing plug-in samples, their purpose, and the extensions they use.

| PLUG-IN | PURPOSE | EXTENSIONS |
|---|---|---|
| Recent Buddies | Stores a list of most recent chat partners into a type of preferred users list. | *core.serviceListeners<br>*messages.MessageHandlerListener<br>com.ibm.rcp.ui.shelfViews<br>org.eclipse.ui.views<br>org.eclipse.ui.popupMenus (object contributions) |
| Buddy Note | Adds an application window or tab to the mini-apps area for typing text notes about selected buddies. | *messages.MessageHandlerListener<br>com.ibm.rcp.ui.shelfViews<br>org.eclipse.ui.views |
| Quick Response | Creates an action button that opens a set of chat responses that can be added to. | *chatwindow.popupAddOn<br>*chatwindow.chatAction<br>org.eclipse.ui.preferencePages |
| Acronym Expander | Automatically converts common acronyms, like LOL and BRB to “laughing out loud” and “be right back”. | *messages.MessageHandlerListener<br>org.eclipse.ui.preferencePages |
| Snippets | Provides a grab bag of small code samples. | *chatwindow.chatAction<br>*chatwindow.chatArea<br>*chatwindow.nwayListExtension<br>*chatwindow.popupAddOn<br>*messages.MessageHandlerListener<br>org.eclipse.ui.viewActions<br>org.eclipse.ui.actionSets<br>org.eclipse.ui.actionSetPartAssociations<br>org.eclipse.ui.popupMenus<br>org.eclipse.ui.views |
| Branding | Contributes branding areas to the login dialog, contact list, and chat window. | *ui.stbranding |

| | | |
|---|---|---|
| Chat Window Browser | Contributes a chat window pop-up addon which contains a browser to view website pages referenced in a 1-1 or nway chat window. | *chatwindow.popupAddOn<br><br>*messages.MessageHandlerListener |

## 6.1 Installing the sample plug-ins

The sample plug-ins can be found in the st9sdk\client\connect\samples directory of the IBM Sametime 9.0 SDK. The plug-ins are packaged as JAR files, which contain both source and binary code. You can import these plug-in JAR files into your Eclipse workspace so that you can look or modify the source code, or you can simply use the plug-ins JARs as provided with IBM Sametime Connect.

The procedure for importing the plug-ins into your Eclipse workspace was described in Chapter 4, *Extending IBM Sametime Connect By Example*. If you intend to make any changes to the sample source code, you should save a copy of the original samples somewhere.

To use the sample plug-ins in the SDK with the IBM Sametime Connect client, you must install them using an update site. To install the samples, use the provided sample update site which can be found in:

<*SDK Location*>/client/samples/com.ibm.collaboration.realtime.samples.updatesite

To install the samples in the provided update sites, perform the following steps:

1. Start the Sametime client and log in
2. Open the Tools -> Plug-ins -> Install Plug-ins... menu
3. Select Search for new features to install and click Next
4. Click the New Local Site... button (Click New Remote Site if the update site is on a web server)
5. Navigate to and select the directory (or enter the web address if remote)
6. Click OK on the Edit Local Site dialog popup
7. Make sure the site you just created is selected in the "Sites to include in search" window
8. Click Finish
9. A feature update window will open, select all the samples you wish to install and click Finish
10. Accept the license agreement (just a template) and click Next
11. Click Finish
12. Click Yes to restart the client when prompted

## 6.2 The Buddy Note plug-in

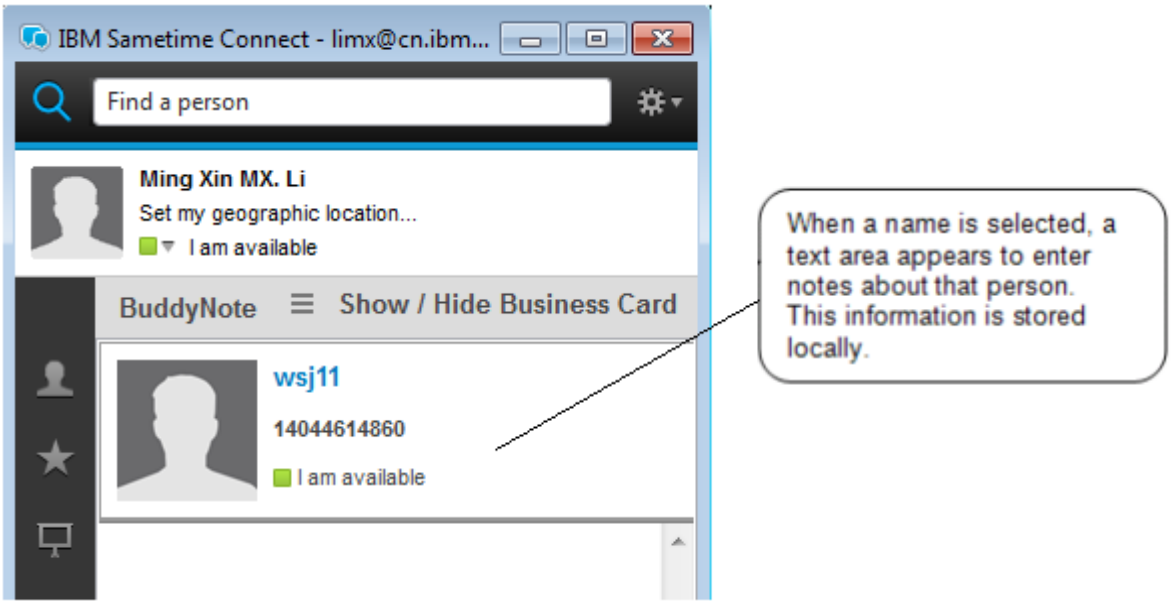
The Buddy Note application provides a way to store notes about the people in your contact list. Whenever a contact is selected their buddy note is displayed. These notes are stored locally.

The Buddy Note plug-in demonstrates key functionality, such as how to perform the following actions:

- Declare an extension to add an application to the shelf view area

- Track the selection of a person
- Persist data locally

The following images show the user interface with the Buddy Note plug-in.




### 6.2.1 The Buddy Note plug-in extension manifest

The following is an excerpt from the Buddy Note extension manifest, plugin.xml.

```
<plugin>

    <extension
       point="com.ibm.collaboration.realtime.messages.MessageHandlerListener">
       <messageHandler
    class="com.ibm.collaboration.realtime.sample.buddynote.BuddyNoteMessageHandlerAdapter"/>
    </extension>

    <extension
       point="org.eclipse.ui.views">
       <view
              class="com.ibm.collaboration.realtime.sample.buddynote.BuddyNoteView"
              icon="images/ST_Ext_Buddy_Note.png"
              id="com.ibm.collaboration.realtime.sample.buddynote.BuddyNoteView"
              name="BuddyNote"/>
    </extension>

    <extension
       point="com.ibm.rcp.ui.shelfViews">
       <shelfView
              id="com.ibm.collaboration.realtime.sample.buddynote.shelfView"
              region="BOTTOM"
              showTitle="true"
              view="com.ibm.collaboration.realtime.sample.buddynote.BuddyNoteView"/>
    </extension>

</plugin>
```

Table listing classes used in the Buddy Note plug-in.

| CLASS | FUNCTION |
|---|---|
| BuddyNoteView | This class extends the ViewPart class, which builds the application shelves, |

| | |
|---|---|
| | defines what content appears in the window, and how it will be laid out. |
| BuddyNoteMessageHandler Adapter | Handles and processes the messages from the notification service. |

### 6.2.2 BuddyNoteView class

In order to build a plug-in that associates text notes with selected chat partners, the BuddyNoteView class extends the ViewPart class.

BuddyNoteView adds the following functionality to the ViewPart class.

- Creates a text control for text to be created and stored
- Listens for UI events
- Updates the content of this area when the user selects a person.
- Associates the entered text with the selected user.

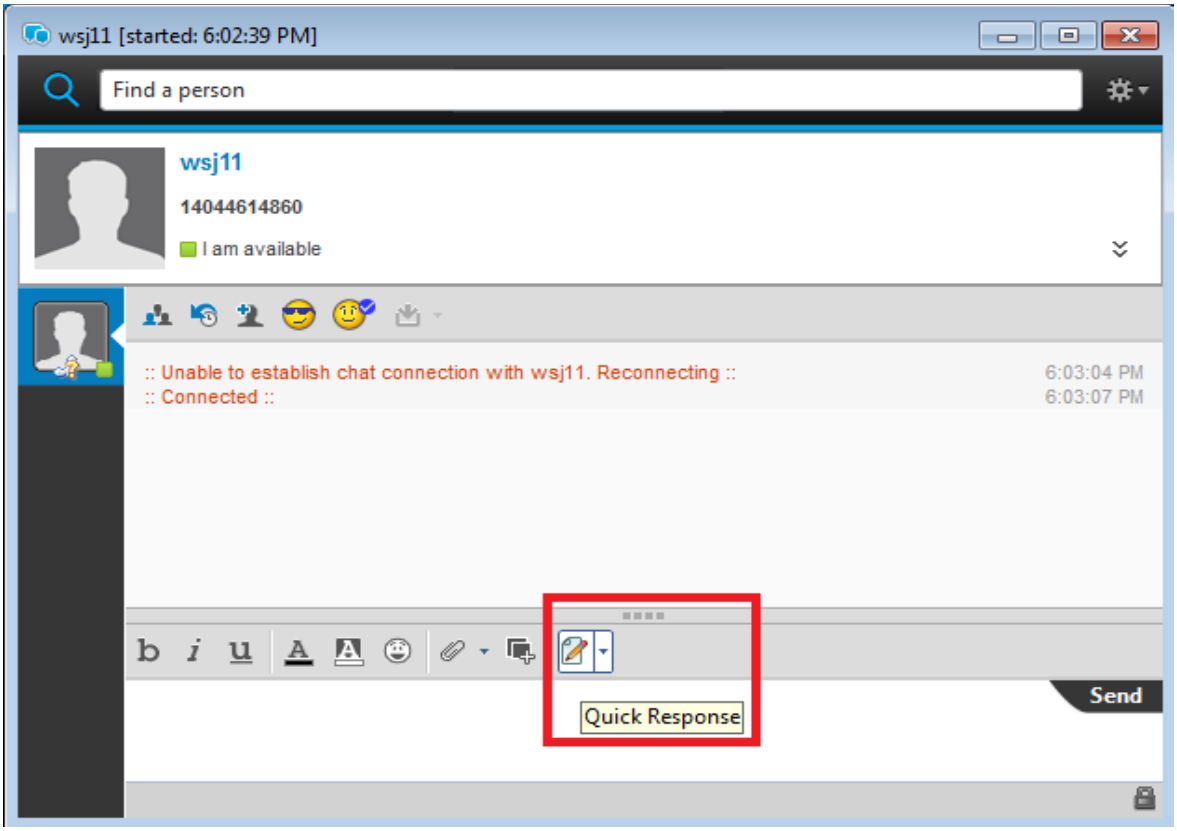
## 6.3 Introducing the Quick Response plug-in

The Quick Response plug-in provides the end-user with a one-click button for inserting pre-typed responses, for example “I’m finishing another call and will be right with you."

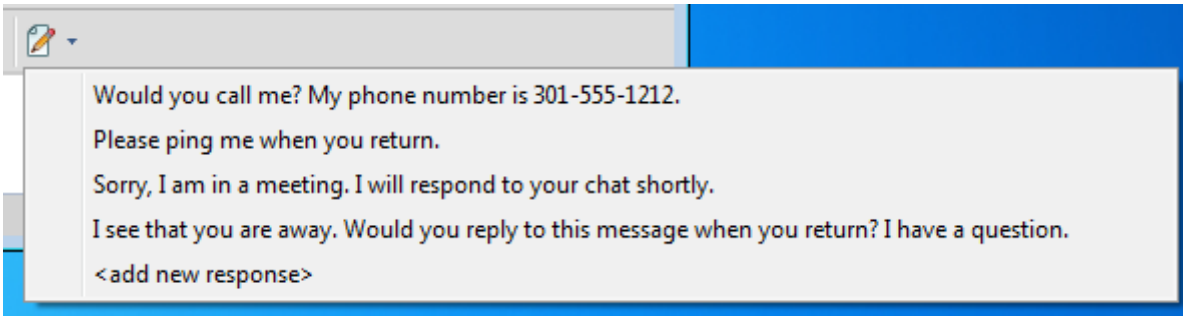
For the programmer, the Quick Response plug-in demonstrates:

- pop-up add-on pane
- chat window integration
- modifying the user interface by adding an icon to the action bar and adding drop-down menu choices

The following image shows an example of the Quick Response plug-in opening in a new window.




The following image shows the Quick Response quick click option.



### 6.3.1 Underlying model of Quick Response

Table listing the key classes used by the Quick Response plug-in and their functions.

| CLASS | FUNCTION |
|---|---|
| QrPopupAddOn | Defines a pane that shows below the chat send area; the pane includes a scrollable list control presenting all the possible responses rather than the maximum 50 in the drop-down menu. |
| QrOpenPopupAddOnAction | Adds the action to the existing general actions in the main toolbar as a drop-down menu |
| QuickResponsePreferencePage | Preference page for adding, removing, and editing quick responses. |

### 6.3.2 The Quick Response Plug-in extension manifest

The following is the Quick Response Plug-in manifest file.

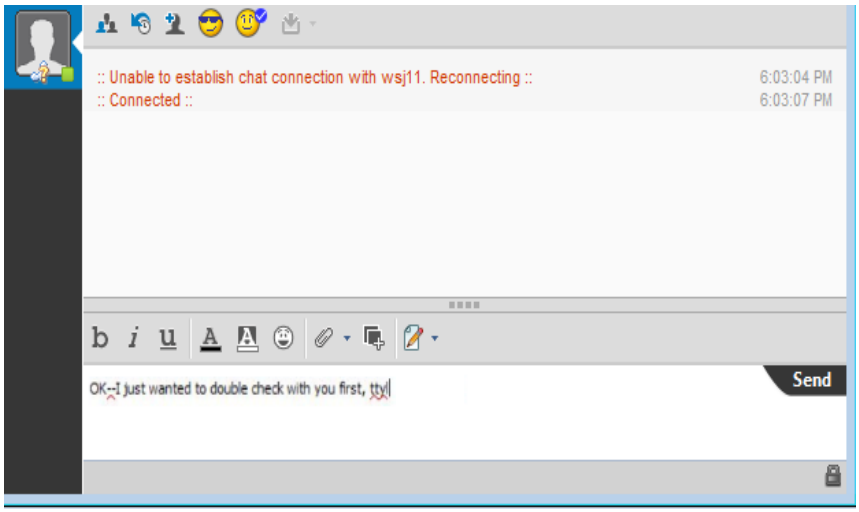
```
<plugin>
   <extension
      point="com.ibm.collaboration.realtime.chatwindow.popupAddOn">
      <popupAddOn
             class="com.ibm.collaboration.realtime.sample.quickresponse.QrPopupAddOn"
             id="com.ibm.collaboration.realtime.sample.quickresponse"
             name="Quick Response"/>
   </extension>

   <extension
      point="com.ibm.collaboration.realtime.chatwindow.chatAction">
      <chatAction
      class="com.ibm.collaboration.realtime.sample.quickresponse.QrOpenPopupAddOnAction"
             displayName="Quick Response"
             id="com.ibm.collaboration.realtime.sample.quickresponse.openquickresponse"
             image="images/quickresponse.png"
             path="format/end"
             showsFor="both"
             tooltipText="Quick Response"
             type="format"/>
   </extension>

   <extension
      point="org.eclipse.ui.preferencePages">
      <page
   class="com.ibm.collaboration.realtime.sample.quickresponse.QuickResponsePreferencePage"
             id="com.ibm.collaboration.realtime.sample.quickresponse.page1"
             name="Quick Response"/>
   </extension>

</plugin>
```

## 6.4 Acronym Expander

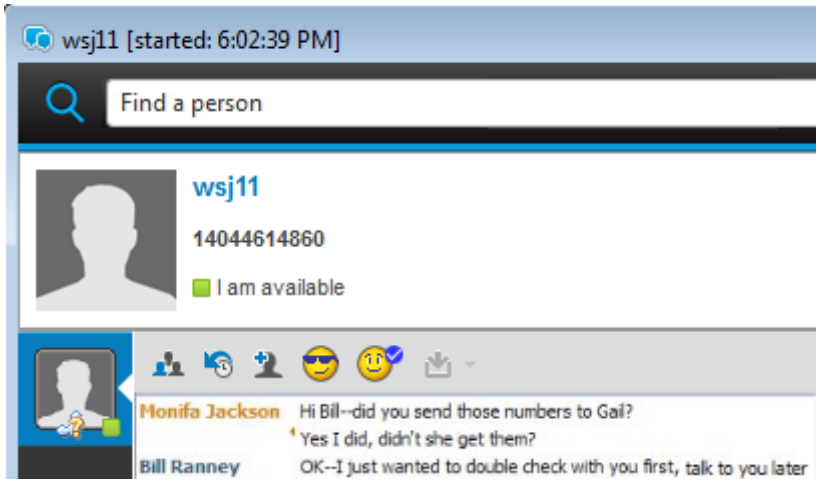
Acronym Expander is a plug-in that will automatically convert incoming acronyms such as LOL and BRB to their full text phrase, such as “laughing out loud” and “be right back”.

The following images show the how acronyms look before and after they are activated with the Acronym Expander plug-in.

Before:




After:



### 6.4.1 Underlying model of the Acronym Expander

Table listing the key classes used by the Acronym Expander plug-in and their functions.

| CLASS | FUNCTION |
|---|---|
| AcronymExpanderMessageEventCallback | Callback handler of incoming messages that monitors incoming text messages before they are passed to the registered message handlers.<br><br>See com.ibm.collaboration.realtime.messages.MessageHandlerListener extension point. |
| AcronymExpanderPreferencePage | Acronym preference page for adding / removing / editing the paired entries and enabling expansion (for example, TTFN = ta ta for now). |

### 6.4.2 The Acronym Expander plug-in extension manifest

The following is the Acronym Expander plug-in manifest file.

```
<plugin>
   <extension
      point="org.eclipse.ui.preferencePages">
      <page
class="com.ibm.collaboration.realtime.sample.acronymexpander.AcronymExpanderPreferencePage"
            id="com.ibm.collaboration.realtime.sample.acronymexpander.page"
            name="Acronym Expander"/>
   </extension>

   <extension
      point="com.ibm.collaboration.realtime.messages.MessageHandlerListener">
      <messageHandlerCallback
      class="com.ibm.collaboration.realtime.sample.acronymexpander.AeMessageHandlerCallback"
      id="com.ibm.collaboration.realtime.sample.acronymexpander.messageHandlerCallback1"/>
   </extension>

</plugin>
```

## 6.5 Branding

The branding application provides a way to specify logos and related branding information in IBM Sametime Connect in the login dialog, contact list, and chat window. The branding UI defines for a simple image that appears at the top of the login dialog, a website URL whose content is shown in the contact List, and a composite widget displayed next to the chat window's business card that displays a graphic and opens a web browser when clicked.

**Note:** Branding is only supported in standalone Sametime client. It is not supported in Notes embedded Sametime client.

To change the default branding in Sametime, you must tell the Sametime UI plug-in to use your custom branding plug-in instead of the default. This is done by setting the `stBranding` preference in the `com.ibm.collaboration.realtime.ui` plugin-in to the ID of your custom branding extension. For example, to use the branding sample in the SDK, the property must be set as follows:

```
com.ibm.collaboration.realtime.ui/stbranding=com.ibm.collaboration.realtime.sample.
branding.custom
```

This can be done during install time by editing the `plugin_customization.ini` file within the deploy directory of the install CD (or network-install directory).

It can also be done afterwards by provisioning the plug-in preference to Sametime Connect, see technote #1261055:

http://www-1.ibm.com/support/docview.wss?rs=477&uid=swg21261055

A default `plugin_customization.ini` file has been included in the branding sample. After installing this sample into a client, follow either of the two steps above to enable it. To test this sample in the development environment, add the following in the "Program arguments:" section of the Arguments tab in the Run / Debug configuration window (see section 4.1.7):

```
-plugincustomization [Location of Branding sample source]/plugin_customization.ini
```

## 6.6 Chat Window Browser

This plug-in contributes a chat window pop-up addon which contains a browser to view website pages referenced in a 1-1 or n-way chat window. It demonstrates how users can enhance the collaborative experience by recognizing website URLs and showing them immediately below the chat window Send area.

In the sample implementation, the keyword @openURL must proceed a URL in the chat transcript to automatically open that website in the pop-up add-on. Of course, the URLs from sites that allow embedded content could easily be translated to their more effective webpage oriented presentation to embedded presentation. For example, if the URL www.example.com/videos?v=123 is detected in a received text message where "v" is the video ID for example.com's video content, the extending plug-in could automatically translate this URL into the embedded equivalent for display within this pop-up add-on. Typically such websites indicate precisely the URL for top-level and embedded content, such as:

```
<object embed src="http://www.example.com/" ...>
```

Such embedded HTML content can be written to a local HTML file in the file system temporary space so it can be rendered by this pop-up add-on as file://temp/example_com_videos_v_123. The implementation of such a plug-in extension is left as an exercise for the reader. The sample demonstrates keyword usage. For example, preceding a URL with the keyword @openURL will automatically display it in your chat partner's window, if they have this plug-in installed. This notion can be extended to more business-oriented uses to increase the productivity of your IM community. For example, "@customer AAA" to look up and display information about customer AAA you wish to ask a colleague about, or @pmr 12345 to display a problem report you wish to discuss with your co-worker.

## 6.7 Snippets

The Snippets plug-in is a grab bag of small code samples. They don't perform any practical function, but serve as simple examples of how to extend the user interface with new menu items, toolbar buttons, and n-way chat list extensions. The contributed actions are prefixed with **STIG** to help you find them quickly among the standard user interface elements.

## 6.8 Conclusion

The introduction to this guide stated that the IBM Sametime Connect client provides extensibility features that go far beyond those in previous IBM Sametime releases. Now that you've read the guide and worked with the accompanying samples in the IBM Sametime SDK, you'll no doubt agree with that statement.

The IBM Sametime Connect client and other IBM managed client products provide unprecedented opportunities for you to build solutions that extend and integrate with IBM client products.

# Appendix A. Using the Java Toolkit to send Rich Text and Binary Data

The Sametime Connect APIs toolkit currently does not provide any mechanism to send rich text or binary data like images to a client. Support for this is being considered for a future release of the SDK. In the meantime, the Java Toolkit can be used to send this type of message to clients. This topic provides information on sending rich text and binary data to the Sametime Connect Client using the Sametime Java Toolkit.

**Note**: The following assumes you are familiar with the general concepts of using the Java Toolkit and in particular using the Java Toolkit to send instant messages. For more information on using the Java Toolkit, see the Java Toolkit Developer's Guide.

The basis of sending rich text, images, and mixed content (text + images) involves sending data over the IM channel in a particular format:

## Sending data over an IM session

Sending data involves invoking the sendData(...) method on the IM session to the recipient with the appropriate arguments.

The method detail for **IM.sendData()** is as follows:

```
public void sendData(boolean encrypted, int dataType, int dataSubType, byte[] data)
```

In the code that calls the sendData method:

The **dataType** for sending binary data is: `27191`

The **dataSubType** for sending binary data is: `0`

The **data** byte array includes another subset of indicators for: data type, data subtype and a bytes indicator.

The data byte array can be constructed as follows:

```
byte[] data = null;
  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  DataOutputStream dataStream = new DataOutputStream(baos);
  try
  {
      dataStream.writeUTF(messageType);
      dataStream.writeUTF(messageSubType);
      dataStream.write(messageBytes);
      data = baos.toByteArray();
  }
  catch (IOException e)
  {
      // Handle exception
  }
```

where messageType, messageSubType and messageBytes can be the following:

Table describing Message types and subtypes.

| MESSAGE TYPE | MESSAGE SUBTYPE | BYTE OR SEQUENCE | DESCRIPTION | USAGE NOTES |
|---|---|---|---|---|
| data | richtext | 0xEE | Tells client that current chat partner is capable of receiving rich text | When this is received, the chat window enables rich text controls and formatting |
| data | richtext | 0xDF | Tells client that current chat partner is not capable of receiving rich text | When this is received, the chat window disables rich text controls and formatting |
| data | command | 0xDE | Informs client of start of mixed content (text intermixed with binary data sent as a clump and processed after end message is received). | Once this message is received, all subsequent messages are buffered and will be displayed at once. |
| data | command | 0x82 | Informs client that all mixed content has been received and processing can begin. | When this is received, the buffered messages are processed and displayed at once. |
| data | image | byte[] | Byte[] will be binary data for an image file. It should be written directly to a file (image type is not sent) | |

Once you have received the imOpened() event for the IM session (refered to from now on as `imSession`), you can invoke **sendData**:

```
imSession.sendData(true, 27191, 0, data);
```

For convenience we can wrap this process in a method and refer to it later:

```
    public void sendDataMessage(String messageType, String messageSubType, byte[]
messageBytes)
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dataStream = new DataOutputStream(baos);
        try
        {
            dataStream.writeUTF(messageType);
            dataStream.writeUTF(messageSubType);
            dataStream.write(messageBytes);
        }
        catch (IOException e)
        {
            throw new AssertionError("sendDataMessage failed");
        }

        imSession.sendData(true, 27191, 0, baos.toByteArray());
    }
```

The following sections show how to use the mechanism of sending data to achieve rich text, image, and mixed content messages.

## Sending rich text

There are three steps involved in sending rich text:

Send an acknowledgement to the IM recipient that you are capable of sending rich text.

Format the message inside HTML span tags with desired font attributes.

Send the message over the IM channel

First you need to tell the recipient client that you are capable of sending rich text. Once you do that, the client will assume all messages from you will be rich text. Using the sendDataMessage(...) method we created and the details in the table above we can send the acknowledgement as follows:

```
// Send rich text acknowledgement
sendDataMessage("data", "richtext", new byte[] { (byte) 0xEE} );
```

Now that the client knows rich text is coming, you can format your text and send it on.

Format your rich text message as desired. An example follows:

```
String richTextStart = "<span style=\"font-size:8pt;font-
family:Tahoma;color:#000000;font-style:normal;\" class=\"left\">";
String richTextEnd = "</span>";
String myRichTextMessage = "<b>Hello!</b>";
String messageToSend = richTextStart + myRichTextMessage + richTextEnd;
```

Now send the rich text message on the IM session as text. The method details for **IM.sendText()** is as follows:

```
public void sendText(boolean encrypted, String text);
```

So you can send your message:

```
imSession.sendText(true, messageToSend);
```

## Sending mixed content including images

Sending mixed content includes sending multiple messages of varying type in one instant message to the client. Types can include text, rich text, images, etc. NOTE: The Connect Client expects all images to be sent as part of a mixed content message. The process of sending mixed content involves sending a start indicator, one or more messages (text or binary), and a stop indicator. Using the sendDataMessage(...) method we've created and the details in the table above we can send mixed content using the following example steps:

```
// Send mixed content start indicator
sendDataMessage("data", "command", new byte[] { (byte) 0xDE} );

// Send text
m_im.sendText(false, "First piece of text");
m_im.sendText(false, "Second piece of text");

// Send mixed content stop indicator
sendDataMessage("data", "command", new byte[] { (byte) 0x82} );
```

Sending images in particular requires sending an HTML image tag and then sending the image data as bytes (using the correct subtype). Using the sendDataMessage(...) and the table above, we can send an image like this:

```
// First load the image
File image = new File("C:/myImage.gif");
FileInputStream imageStream = new FileInputStream(image);
byte[] imageBytes = new byte[imageStream.available()];
imageStream.read(imageBytes);

// Now send the HTML image tag and the image data
m_im.sendText(false, "<img src=\"\"/>");
sendDataMessage("data", "image", imageBytes);
```

A full example of sending mixed content including an image:

```
// First load the image
File image = new File("C:/myImage.gif");
FileInputStream imageStream = new FileInputStream(image);
byte[] imageBytes = new byte[imageStream.available()];
imageStream.read(imageBytes);

// Send mixed content start indicator
sendDataMessage("data", "command", new byte[] { (byte) 0xDE} );

// Send some text
m_im.sendText(false, "First piece of text - before image");

// Now send the <img tag and the image data
m_im.sendText(false, "<img src=\"\"/>");
sendDataMessage("data", "image", imageBytes);

// Send some more text
m_im.sendText(false, "Second piece of text - after image");

// Send mixed content stop indicator
sendDataMessage("data", "command", new byte[] { (byte) 0x82} );
```

# Appendix B. Livename Extended Status API

## Livenames Extended Status API

This livename extended status API allows you to display custom status icons in livenames in the buddy list and elsewhere in the user interface. The custom icons appear next to (to the left of) the IBM Sametime status icons.

The livename extended status API includes two plug-ins:

1. com.ibm.collaboration.realtime.status.ext_9.0.jar

This is the extended status implementation plug-in, which is included in the Sametime Connect 9 client. In previous Sametime releases, this plug-in was provided in the SDK – see *Enabling Extended and Telephony Status* at the end of this appendix for more information.

2. com.ibm.collaboration.realtime.sample.status.ext_9.0.jar

This is the extended status sample plug-in, including source code. This plug-in can be found in the samples subdirectory of the IBM Sametime Connect Toolkit

The extended status implementation plug-in provides an extension point for you to introduce new status icons to a livename, wherever that livename appears in the IBM Sametime Connect user interface. The primary use for this feature is to display a status icon for a user's telephone status (for example, on hook or off hook), next to the status icon for the user's IBM Sametime presence. Although it's possible to display any number of additional status icons, the practical limit is 3 icons, including the Sametime presence icon. The recommended size for each status icon is 16 x 16 pixels. Inconsistent image sizes will cause other icons to stretch or shrink.

In order to leverage the status extension framework in IBM Sametime, you need to

1. Implement the com.ibm.collaboration.realtime.status.ext.StatusProvider interface,

2. Use the com.ibm.collaboration.realtime.status.ext.statusProviders extension point to register your implementation.

The interface and extension point are defined in the extended status implementation plug-in.

The StatusProvider implementation returns the custom status image for the given LiveName object in its getStatusImage method. A federated status controller combines the status images of all registered extended status providers, and combines those images into a single, consolidated status icon. If the consolidated icon for a given pattern of statuses has already been created, it is reused. The StatusProvider interface also defines a getStatusText method, however, this method is not currently utilized by the federated status controller.

Status extenders can be passive or active. A passive status extender provides image only in response to Sametime presence status changes. An active status extender refreshes the livename icon actively, in

response to the extended status events that the status provider is monitoring. The monitoring of extended status events is external to IBM Sametime Connect; it is up to the status extender to detect status event changes, for example, by connecting to a separate telephony presence server. Once a change is detected, the status extender can refresh the display of the user's status icons.

See the extended status sample plug-in in this toolkit for more information about extending livename status. The sample plug-in provides a StatusProvider implementation that displays a telephony status icon next to the Sametime presence icon. You can use this sample as the basis for an implementation that responds to telephony status change events.

To use the livename extended status sample plug-in with IBM Sametime Connect, follow the instructions for installing the sample plug-ins in Chapter 6.

## Telephony Status API

In addition to the extended status plug-in, the IBM Sametime Connect 9.0 client includes the telephony status plug-in, com.ibm.collaboration.realtime.telephony.status, which uses the capabilities of the livename extended status API to display telephony status icons on livenames in the buddy list and elsewhere. Like the extended status plug-in, the telephony status plug-in was provided in the IBM Sametime SDK in previous Sametime releases – see *Enabling Extended and Telephony Status* below for more information.

The telephony status plug-in works in conjunction with the IBM Sametime Telephony Presence Adapter in the Sametime Community Server Toolkit. The Telephony Presence Adapter is used by both IBM Sametime Unified Telephony and third-party telephony service providers to integrate the Sametime server with telephony presence systems. The IBM Sametime server obtains telephony status from the Telephony Presence Adapter and publishes status changes via user attributes. The telephony status client plug-in responds to telephony status user attributes by displaying the appropriate telephony status icon next to the Sametime presence icon.

In environments where third-party telephony presence is used, telephony status can be enabled on the client as described in the next section.

## Enabling Extended or Telephony Status

Starting in IBM Sametime 8.5.1, the extended status and telephony status plug-ins are included in the IBM Sametime Connect client, so it is no longer necessary to deploy these plug-ins to customers in order to make use of these features. In previous releases, these plug-ins were only included in the IBM Sametime SDK and in the IBM Sametime Unified Telephony client add-on.

Since these plug-ins change the appearance of livenames in the contact list, they are disabled by default, so that they do not impact IBM Sametime 8.5.x users who do not need these features. The extended status feature, or both the extended status and telephony features, can be enabled using client preferences. For more information on how to enable these plug-ins, and how they are used by IBM Sametime Unified Telephony, see the following topic in the IBM Sametime 8.5.1 Information Center:

http://publib.boulder.ibm.com/infocenter/sametime/v8r5/topic/com.ibm.help.sametime.v851.doc/config/config_client_sut_ext_tel.html

Note: Because previous releases of the IBM Sametime Connect client do not include these plug-ins, if you deploy client plug-ins that require extended status or telephony status to older clients, you will also need to deploy the appropriate versions of the extended status and telephony status plug-ins. You can find these plug-ins in previous releases of the IBM Sametime SDK.

# Appendix C. How to convert miniApp to shelfView

In the Sametime 7.5 release, the sample plug-ins were created as mini-application tabs, instead of the shelves used in Sametime 8.x. If you used the miniApps class you can modify your plug-in to use the shelfView instead.

Follow these steps to convert a miniApps plug-in to one that uses the shelfView.

1. Change your AbstractMiniApp class to ViewPart (just reuse the class and edit it)
2. If you have an AbstractMiniApp.init() method, you can move the innards to a ViewPart.init(viewsite) method
3. Override the ViewPart.createPartControl method to just call your AbstractMiniApp.createControl method.
4. Change your plugin.xml to use the shelfViews extension point:
   a. Remove the miniApp ext pt
   b. Add a views extension for your view:

```
<extension point="org.eclipse.ui.views">
      <view
      class="com.ibm.something.package.whatever.MyMiniApp"
      icon="icon/ST_Logo_12x12.png"
      id="com.ibm.something.package.whatever.myminiapp"
      name="%something"/>
</extension>
```

   c. Add a shelfViews extension using that view:

```
<extension point="com.ibm.rcp.ui.shelfViews">
      <shelfView
      id="com.ibm.something.package.whatever.myshelfview"
      region="TOP" <-- this says where the thing goes, TOP, MIDDLE, BOTTOM
      showTitle="true"
      view="com.ibm.something.package.whatever.myminiapp/>
      <!-- this is the ID of the view, NOT the CLASS -->
</extension>
```

And you're done!

AbstractMiniApps will continue to work in 8.0x but is deprecated. The benefit of using shelfView is that you can reuse your miniapp inside IBM Notes or the Expeditor application of your choice.

# Appendix D. How to access the Java Toolkit Sametime Session Object

While developing plug-ins for IBM Sametime Connect, it may be necessary to access the lower level APIs available in the Sametime Java Toolkit. This can be accomplished by getting a handle to the STSession object. Sessions are stored at the Community level, so there is one session per Sametime Community. The STSession protocol objects are wrapped in RtcSession objects, so we must get the RtcSession from the Community, then get the protocol session and cast it to a STSession object:

```
import com.ibm.collaboration.realtime.im.community.Community;
import com.ibm.collaboration.realtime.im.community.CommunityService;
import com.ibm.collaboration.realtime.servicehub.ServiceHub;
import com.ibm.collaboration.realtime.RtcSession;
import com.lotus.sametime.core.comparch.STSession;
…
…
…

String communityId = "….";
CommunityService commSvc = (CommunityService)
ServiceHub.getService(CommunityService.SERVICE_TYPE);Community
Community community = commSvc.getCommunity(communityId);
RtcSession rtcSession = community. getRtcSession();
STSession stSession = (STSession) rtcSession.getProtocolSession();
```

Once you have the STSession object you can make use of all the APIs available in the Sametime Java Toolkit.

# Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
5 Technology Park Drive
Westford Technology Park
Westford, MA 01886

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only. All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp.
Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## *Trademarks*

These terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM

AIX

DB2

DB2 Universal Database Domino

Domino

Domino Designer

Domino Directory

i5/OS

iSeries

Notes

OS/400

Sametime

System i

WebSphere

AOL is a registered trademark of AOL LLC in the United States, other countries, or both.

AOL Instant Messenger is a trademark of AOL LLC in the United States, other countries, or both.

Google Talk is a trademark of Google, Inc, in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft, and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.