# Table of Contents

1. Preparing for the Tutorial ........................................................................................................ 1  
   System Requirements ......................................................................................................... 1  
   *Apache Ant* ..................................................................................................................... 1  
   *JacORB 2.2.1* .................................................................................................................. 1  
   *OpenCCM 0.9.0* .............................................................................................................. 1  
   Installing the OpenCCM Platform ..................................................................................... 2  
   Tips on Installing OpenCCM and JacORB ....................................................................... 3  
2. Overview ................................................................................................................................. 4  
   *Cadena* .............................................................................................................................. 4  
   *Example Project* ............................................................................................................... 4  
3. Getting Started ....................................................................................................................... 5  
   *Eclipse* ............................................................................................................................... 5  
   *Creating the project* .......................................................................................................... 5  
   Known Issues with OpenCCM Wizard ............................................................................... 6  
4. Module Editor .......................................................................................................................... 7  
   Opening the Editor ............................................................................................................. 7  
   *The Server Component* .................................................................................................. 7  
      *Creating the server component type* ........................................................................... 7  
      *Creating the interface types* ....................................................................................... 8  
      *Adding ports to the server* ......................................................................................... 9  
   *The Client Component* .................................................................................................. 9  
      *Creating the client component type* ......................................................................... 10  
      *Adding ports to the client* ....................................................................................... 10  
5. Implementation ....................................................................................................................... 11  
   Implementation Skeleton Generation ............................................................................. 11  
   Server implementation ..................................................................................................... 11  
   Client implementation ...................................................................................................... 12  
6. Scenario Editor ....................................................................................................................... 16  
   Opening the editor ............................................................................................................. 16  
   *Creating the instances* .................................................................................................. 17  
   *Creating connections between the instances* .............................................................. 17  
   The Tennessee Two Step Method ................................................................................... 17  
   The Oklahoma Mesa Method ......................................................................................... 19  
   The Arizona Blue and Yellow Method ........................................................................... 20  
   The Texas Brazos Method ............................................................................................... 20  
7. Scenario Execution ............................................................................................................... 22  
   Starting the system ........................................................................................................... 22  
   Stopping the Scenario ...................................................................................................... 22  
   Glossary ............................................................................................................................... 23  
Bibliography ............................................................................................................................ 26
List of Figures

1.1. Confirm OpenCCM Platform Install ................................................................. 3
2.1. The Cadena meta-modeling language .............................................................. 4
2.2. The chat system ............................................................................................... 4
3.1. Eclipse Workbench ......................................................................................... 5
4.1. Module editor .................................................................................................. 7
4.2. Completed types ............................................................................................... 10
5.1. Java editing within Cadena ............................................................................. 11
5.2. Client interface ............................................................................................... 13
6.1. Scenario editor ................................................................................................ 16
6.2. Add CCMInterfaceConnector ....................................................................... 18
6.3. Port Right Click ............................................................................................. 18
6.4. Component Port Binding ............................................................................... 18
6.5. Port Right Click ............................................................................................. 19
6.6. Component Port Binding ............................................................................... 19
6.7. Port Right Click ............................................................................................. 20
6.8. Port Right Click ............................................................................................. 20
6.9. Port Right Click ............................................................................................. 21
6.10. Component Port Binding ............................................................................ 21
7.1. Running system .............................................................................................. 22
Chapter 1. Preparing for the Tutorial

In this chapter, we will guide you through the process of installing the software required to complete the example provided in this tutorial. We assume that you have already read the Cadena 2.0: Install Guide and have completed the installation of Java, Eclipse, and Cadena. This guide will walk you through installing OpenCCM and the Cadena OpenCCM Platform. It also has some tips and tricks included.

System Requirements

Before you can run the tutorial, you must first install the following software packages. We present the basics here, for additional tips on installation, execution, and troubleshooting, please see the section called “Tips on Installing OpenCCM and JacORB”. We will now install:

1. Apache Ant
2. JacORB 2.2.1
3. OpenCCM 0.9.0

It should be noted that while performing these installations and running the resulting executables, the authors used the Gentoo GNOME (2.12.3) Linux platform and the Microsoft Windows XP Professional platform.

Apache Ant

To compile OpenCCM, you will need Apache Ant version 1.6 or above. If you do not already have it, you can obtain it from the Apache website [http://ant.apache.org/]. Alternatively, you can use the Ant build file from within Eclipse using the Ant plugin that is included with Eclipse.

To build an Apache Ant build file (build.xml by default) from within Eclipse, just open the file in Eclipse and run as an Ant build file. Whether you download and install Ant yourself or use it from within Eclipse, you have to be sure that your JAVA_HOME environment variable is set because Ant will use it to locate the JDK on your system. For more information on checking or setting your JAVA_HOME environment variable see the tips section of the Cadena 2.0: Install Guide.

From now on, we assume Ant is available on your system.

JacORB 2.2.1

You are now ready to install JacORB 2.2.1. OpenCCM requires an ORB package, and when installing OpenCCM, you will refer OpenCCM to the new JacORB directory. JacORB 2.2.1 can be obtained from jacob.org [http://www.jacob.org/download.html].

On this download site, look for the heading "JacORB 2.2.1". For the purposes of the tutorial, download the compact version appropriate for your platform. We suggest that you download the gzipped tar-archive jacORB_2_2_1-compact.tar.gz for Linux or the zipped archive jacORB_2_2_1-compact.zip for Windows.

To install JacORB, simply extract the contents of the archive into the directory of your choice.

OpenCCM 0.9.0

You should place OpenCCM 0.9.0 into the same directory in which you installed JacORB 2.2.1. OpenCCM 0.9.0 can be obtained from the OpenCCM website [http://openccm.objectweb.org/doc/0.9/]
Preparing for the Tutorial

install_guide.html]. From the "Project Links" list on the left of the browser window, choose "Download". From there you will open the "ObjectWeb Forge Site" link. We suggest that you download OpenCCM-0.9.0.tar.gz or OpenCCM-0.9.0.zip for Linux or Windows respectively.

We now summarize the installation steps below. If you choose to follow the instructions on the webpage, follow them up to and not including the installation instructions (stop after the compilation step).

Preparing OpenCCM for Cadena

Extract the contents of the OpenCCM archive into the directory containing JacORB 2.2.1 (installing them in separate directories should work just as well).

If you're using the command line, enter the OpenCCM-0.9.0/openccm and run ant with no arguments.

If your running Ant from within Eclipse, while the Ant build file build.xml has the focus in the Eclipse editor window, right click in the editor and select Run As # Ant Build.

Ant will warn you that you need to edit the build.properties file. To edit the file:

- Select the ORB used to compile and execute the OpenCCM platform: Uncomment the ORB.name=JacORB-2.2 line and only that line. Leave the rest commented.

- Set the ORB.home.dir variable indicating the directory where the used ORB is installed. For example:

  ORB.home.dir=/software/OpenCCM/JacORB_2_2_1

When in Windows, be sure to use "/s"s and not "\"s when specifying the path for the ORB.home.dir variable.

Now that you have edited the properties file you can run Ant again. Note that compiling OpenCCM may take a fair amount of time depending on the power of your system. We have seen this take up to 20 minutes to complete. Once the compilation process is finished you're ready to move on to the installation of the Cadena OpenCCM Platform. For the purposes of this tutorial, you do not have to install OpenCCM, you must only compile it. From this point forward, we assume that you have compiled and not installed OpenCCM.

Installing the OpenCCM Platform

Now that you have an OpenCCM installation available on your local machine you can install the OpenCCM Platform plugin for Cadena. To do this, you should use the directions in the Cadena 2.0: Install Guide to direct you through the process of installing the 2 plugins that make-up the OpenCCM Platform in Cadena:

- Cadena CCM Platform 2.0.*

- Cadena OpenCCM Platform 2.0.*

In Figure 1.1, “Confirm OpenCCM Platform Install” you will see the Search Results dialog where the features are selected.
Tips on Installing OpenCCM and JacORB

The following are tips regarding OpenCCM installation and problems that may be encountered when using OpenCCM and possible fixes.

- **You are unable to open the Module and Scenario Specifications.**

  If attempting to open the Module and Scenario Specifications produces the error message *Unable to create this part due to an internal error...* then you may not be running the Eclipse workbench and the installed plugins with Java 5. For instance, you may be using an older version such as JDK 1.4. Cadena and Cadena OpenCCM require Java 5. Check to ensure that you are running Eclipse with Java 5. As mentioned in the Cadena 2.0: Install Guide, you can force Eclipse to run on a particular JRE using the command line option `-vm`.

- **Ant complains that the `build.properties` file is incorrectly set**

  If you get an error message similar to *WARNING: <ORB.home.dir> in the <build.properties> file is incorrectly set!!!*, you have unarchived JacORB_2_2_1-compact.tar.gz in Linux, and you are trying to run `ant` after setting up the `build.properties` file, you may need to change the file permissions within the JacORB directory before `ant` can make sense of the directory you have provided. To change the permissions in the JacORB directory, run the following commands from within the JacORB directory.

  ```
  • find . -type d -exec chmod 775 {} \;
  • find . -type f -exec chmod 664 {} \;
  ```

  Alternatively, you might be able to use the operating system to set the permissions for the directory and all of its sub-directories.

  You should be able to compile OpenCCM successfully now by running `ant` again.
Chapter 2. Overview

Cadena

At its core, Cadena is a component meta-modeling tool. It provides a three tier approach to achieve this functionality. The top tier, or style tier, allows users to describe the component platform that is going to be used. The middle tier, or module tier, allows users to create component types corresponding to the description developed in the style tier. The bottom tier, or scenario tier, allows users to create instances of the types defined in the module tier, and then create connections between those instances.

Once a new style has been created, extensions based on the style can be plugged into Cadena to customize the tool for that platform. A number of different extensions have been created for the OpenCCM style in an effort to provide a complete development environment. Some of these extensions are: a new OpenCCM project wizard, language checkers, code generators, and customized graphics.

Figure 2.1. The Cadena meta-modeling language

Example Project

The rest of this tutorial is developed around a simple chatting application. The application has one chat server and multiple chat clients. The server uses asynchronous events to notify clients of new chat messages while the clients use synchronous method calls to send new messages to the server.

Figure 2.2. The chat system
Chapter 3. Getting Started

Eclipse

Eclipse is a highly extensible, Java™-based open-source IDE. The Eclipse user interface consists of multiple views (for viewing resources and structure) and editors (for editing resources) grouped together as perspectives.

Figure 3.1. Eclipse Workbench

Figure 3.1, “Eclipse Workbench” above shows an editor and several views that you will use in this tutorial. The Navigator view is used for convenient navigation of project files and folders. The Outline view is linked with the editor and displays hierarchical structure of the edited file or other additional information. In this example, the editor is supported by the Properties view. The Properties view shows additional information about the currently selected entity.

Note: For a complete tutorial on Eclipse alone (without reference to Cadena or the OpenCCM plugin), take a look at the online documentation [http://www.eclipse.org/documentation/main.html].

Creating the project

Once the OpenCCM environment is successfully installed, a new OpenCCM project can be created within Cadena.

• Select "File # New # Project..." from the main menu.
• The new project dialog should appear. From the tree on the left, select "Cadena # OpenCCM Project" and then click the "Next" button.

• The first page of the wizard should appear. To continue, a valid project name must be entered. Type in "chat" for the project name and then click the "Next" button to continue.

• The second page of the wizard should appear. To continue, the path to the OpenCCM installation must be specified (this path must be in the form of "\OpenCCM-x.x.x\openccm"). Once a valid path has been selected, click the "Finish" button.

Once the above steps have been successfully completed, a new OpenCCM project will be created and should be present in the navigator view on the left. The wizard automatically creates the necessary directory structure and it also creates the main module and scenario for the project.

**Known Issues with OpenCCM Wizard**

The following are problems that may be encountered while creating the new OpenCCM project.

• **The wizard completes without error but the module and scenario files are not created.**

  This issue usually occurs when Cadena is run in Eclipse 3.2 (instead of 3.1.2). We suggest you double-check the release of Eclipse you are using and potentially un-install and install the correct version. If this isn't the issue, please let the developers know about this through the forums.

• **There is no "OpenCCM Project" available.**

  This is usually because the OpenCCM platform plugin was not installed. Double-check the list of plugins (in the Help | About Eclipse SDK | Plug-in Details list) to make sure it is listed. If this plugin is installed the CCM plugin should also be installed.
Chapter 4. Module Editor

The module editor is used to define the component types used in the system. For the chat example, there are two component types: a chat server component type and a chat client component type.

Opening the Editor

When an OpenCCM project is created with the new project wizard, a main module file is also created within the project. To open the main module for the chat example:

- From the navigator view on the left, browse to the module file: "chat/specification/module".
- Double click on the "chat.module" file.

The module editor has two tabs. The "Overview" tabs shows what external modules are imported by this module and allows additional modules to be imported. The "Table" tab displays all of the elements that compose the module. Right-clicking within the table displays a pop-up menu. The pop-up contains actions which allow elements to be added to the module, removed from the module, and various aspects of the elements to be modified.

When a component type is selected within the table view, the graphical representation of the component type is displayed within the Outline view. This preview of the component is helpful in making sure that the component "looks" right.

Figure 4.1. Module editor

The Server Component

The first component type is the chat server. The chat server is responsible for accepting chat messages from individual clients and then broadcasting those messages to all of the clients that are connected to it.

Creating the server component type

To create the server component:

- Right click within the table.
• Select "Add Component Type # CCMComponent" from the pop-up menu.

• Enter "ChatServer" for the name.

• Click the "Finish" button.

The server component is now created and should show up within the table and in the Outline view.

Creating the interface types

At this point, the server component is unable to communicate with the clients as it has no ports. Before any ports can be added to the component, the interface types that will be used by the ports must be defined.

Two interface types are necessary for this project. The first type of interface is used by the clients to send messages to the server while the second type of interface is used by the server to notify clients about new chat messages.

To create the first interface type:

• Right click in the Interface Type table.

• Select "Add Interface Type # CCMInterface" from the pop-up menu.

• Enter "SendMessage" for the name.

• Click the "Finish" button.

The "SendMessage" interface is now created and should show up within the Interface Type table.

Now that the "SendMessage" interface exists, an operation needs to be added to it in order to allow the clients to send their messages. To add the operation:

• Double click on the "SendMessage" interface within the module table. The properties view should now be visible below the module editor with the interface's properties displayed within it.

• Double click on the "operations" property to set a value to it.

• Right click on "operations" and select "Add Element".

• A new element should now be present as a child of the "operations" property. Expand the newly created element: operations # 0

For each operation of an interface type, the name must be set, the return type must be set, and parameters may optionally be added. To set the name of the operation:

• Click to the right of the "Name" property to activate the name field editor.

• Enter "sendMessageOp".

Setting the return type of the operation is done in a similar way:

• Click to the right of the "return-type" property. A drop-down selection box should appear.

• Select "void" from the drop-down selection.

In order to actually pass some data to the server, two parameters must be added to the operation. The first parameter identifies the sender of the message, while the second parameter contains the content of the message.

The two parameters are added in the same way that the operation is added to the interface, by right clicking on the parameters property and selecting "Add Element". The name of the first parameters should
be "sender", the type is "string", and the attribute is "in". The name of the second parameter should be 
"message", the type is "string", and the attribute is "in".

The next step is to create the interface used to notify clients about new messages. To create this interface:
• Right somewhere click within the Interface Type table
• Select "Add Interface Type # CCMEvent"
• Enter "HandleMessage" for the name
• Click the "Finish" button

Two state members must be added to the event type. The first state member tells the receiver who sent 
the message, while the second state member contains the contents of the message.

The two state members are added in the same way that the "sendMessageOp" operation and its parameters 
are added to the "SendMessage" interface. The name of the first attribute is "sender", the type is "string", 
and the visibility is "public". The name of the second attribute is "message", the type is "string", and the 
"visibility" is "public".

Adding ports to the server

Now that the necessary interfaces are created, ports can be added to the server component. To add the 
first port:
• Right click on the "ChatServer" component within the table view.
• Select "Add Port # provides" from the pop-up menu.
• Use "sendMessage" for the name of the port.
• To select the type of the interface, click the "Browse" button and select the "SendMessage" interface 
type from the dialog.
• Click the "Finish" button.

The same steps are followed to add the second port:
• Right click on the "ChatServer" component within the table view.
• Select "Add Port # publishes" from the pop-up menu.
• Use "handleMessage" for the name of the port.
• To select the type of the interface, click the "Browse" button and select the "HandleMessage" interface 
type from the dialog.
• Click the "Finish" button.

The two ports should now show up as children of the server component type in the table view as well as 
being visible in the outline view.

The Client Component

The second component type is the chat client. This chat client provides a graphical user interface to the end user. The chat client handles sending the user’s messages to the server and displaying messages that are received from the server.
Creating the client component type

The client component is created in the same way that the server component is created with the exception of the name change to "ChatClient".

Adding ports to the client

The interface types that the client utilizes have already been created in the previous section. So, at this point the ports just need to be added. To add the first port:

- Right click on the "ChatClient" component within the table view.
- Select "Add Port # uses" from the pop-up menu.
- Use "sendMessage" for the name of the port.
- To select the type of the interface, click the "Browse" button and select the "SendMessage" interface type from the dialog.
- Click the "Finish" button.

The same steps can be followed to add the second port:

- Right click on the "ChatClient" component within the table view.
- Select "Add Port # consumes" from the pop-up menu.
- Use "handleMessage" for the name of the port.
- To select the type of the interface, click the "Browse" button and select the "HandleMessage" interface type from the dialog.
- Click the "Finish" button.

At this point, the component types are fully defined and the next phase of the system development can begin.

Figure 4.2. Completed types
Chapter 5. Implementation

Once the component types have been defined, the implementation of the components can begin. The OpenCCM plugin for Cadena provides code generation facilities for components while the Eclipse environment provides Java development support. These two features combine to provide a powerful environment for implementing OpenCCM systems.

Figure 5.1. Java editing within Cadena

Implementation Skeleton Generation

Before any work can begin on the components' implementations, their code skeletons must be generated. To do this:

1. Right click on the "chat" project within the resource navigator
2. Select "OpenCCM Actions->Generate Component Code" from the pop-up menu.

Some users have reported issues with this step in the tutorial. The complaint is that the code was not actually generated. Sometimes this is accompanied by a statement that WindowsXP's firewall prompted them about blocking an application. When the code generation step is run, it makes use of the OpenCCM framework. That framework uses a network port to communicate over. This is what WindowsXP is complaining about. So, if a firewall is installed and is blocking the traffic, this step might not work properly.

The skeletons should now be generated. To browse the generated source code:

1. Browse to "chat/src/org.objectweb.ccm.chat.cif" in the navigator view on the left.
2. Double click on any of the Java files within that directory to view the implementation.

Now that the skeleton code is generated we can start implementing the server.

Server implementation

The implementation of the server is fairly simple. All it needs to do is take the individual messages that it receives from each client and rebroadcast it to all of the clients.
Open up the implementation:

1. Browse to the implementation directory as described above.

2. Double click on "ChatServerImpl.java" to open up the Java editor.

3. Replace the generated sendMessageOp source code with this:

   ```java
   /**
    * @generated */
   public void sendMessageOp(
     java.lang.String sender,
     java.lang.String message) {
     System.err.println("<" + sender + "> : " + message);
     HandleMessage outgoingMessage = new HandleMessageImpl();
     outgoingMessage.sender = sender;
     outgoingMessage.message = message;
     get_context().push_handleMessage(outgoingMessage);
   }
   ```

All of the work for the server happens within the sendMessageOp method. This method is invoked by clients when they want to have the server broadcast a message.

4. Remove the JavaDoc code in line #2.

When the OpenCCM plugin generates the skeleton implementations, it does not simply replace existing code, but instead merges the new code in with the old code. The @generated directive is used to identify blocks of code that can be replaced by generated code.

5. Some notes about the code:

   • 7: This line is mainly for debugging purposes. It prints all incoming messages to the console.

   • 9-11: These lines handle the construction of the new outgoing event. The payload of the message is set as well.

   • 13: The event that was just constructed is pushed out. All components that are subscribed to the server will receive a copy of the message.

**Client implementation**

The implementation of the client is a little more complicated than the server as it provides a graphical user interface for the user. The interface allows the user to set their name, send messages to the server, and it displays messages that it receives from the user.
Figure 5.2. Client interface

1. Open the Client implementation.

2. Browse to the implementation directory as described above.

3. Double click on `ChatClientImpl.java` to open up the Java editor.

4. Add the `ActionListener` interface to the ChatClient implementation.

   In order for the component to handle the button events for the "send" button, the component's class needs to implement the `ActionListener` interface:

   ```java
   public class ChatClientImpl
   extends org.objectweb.ccm.chat.ChatClientSessionComposition.ComponentImpl
   implements ActionListener {
   ```

5. Replace the generated `configuration_complete()` method source code with this:

   The next item of business is setting up the GUI and making it visible. All of the GUI setup is done within the `configuration_complete` method:

   ```java
   private JTextArea mainMessageArea;
   private JTextField nameText;
   private JTextField newMessageText;

   /**@generated */
   public void configuration_complete()
   throws org.omg.Components.InvalidConfiguration {
   super.configuration_complete();
   // Set up the main window
   JFrame mainFrame = new JFrame();
   mainFrame.setSize(300, 300);
   mainFrame.setLocation(100, 100);
   mainFrame.getContentPane().setLayout(new BorderLayout());
   mainMessageArea = new JTextArea();
   mainFrame.getContentPane().add(mainMessageArea, BorderLayout.CENTER);
   ```
Implementation

```java
23: bottomPanel.setLayout(new BorderLayout());
24:
25:   // Add the user name area
26:   nameText = new JTextField("anonymous");
27:   bottomPanel.add(nameText, BorderLayout.WEST);
28:
29:   // Add the new message text area
30:   newMessageText = new JTextField();
31:   bottomPanel.add(newMessageText, BorderLayout.CENTER);
32:
33:   // Add the send message button
34:   JButton sendButton = new JButton("Send");
35:   sendButton.addActionListener(this);
36:   bottomPanel.add(sendButton, BorderLayout.EAST);
37:
38:   mainFrame.getContentPane().add(bottomPanel, BorderLayout.SOUTH);
39:
40:   // Show the main window
41:   mainFrame.setVisible(true);
42: }
```

6. Some notes about the code:

- 1-3: These fields are part of the GUI that are referenced in other parts of the component.
- 5: Again, the @generated directive needs to be removed in order to prevent the code generator from replacing the custom method body the next time skeletons are generated.
- 11-42: In this section the GUI is setup as a normal Swing application. On line 35, the component is registered as an action listener so it can handle the "Send" button being pressed.

7. Add the code for the `actionPerformed()` method.

Now that the GUI is setup, the button press events for the "send" button can be handled:

```java
1: public void actionPerformed(ActionEvent arg0) {
2:   // Send a new message to the server
3:   SendMessage sendMessage =
4:     get_context().get_connection_sendMessage();
5:   sendMessage.sendMessageOp(
6:     nameText.getText(), newMessageText.getText());
7:   newMessageText.setText(" ");
8: }
```

8. Some notes about the code:

- 3-4: The component's context is used to fetch the interface that the "sendMessage" port is connected to.
- 5-6: The user name and new message text is fetched directly from the GUI components, and the "sendMessageOp" is called.
• 7: Presumably, the user is done with the sent message, so the new message text field is cleared.

9. Add the code for the **push**() method.

The final aspect of the component is the handling of incoming messages. **push(...)** methods are automatically created for each of the event types the component consumes.

```java
1: /**@generated
2: */
3: public void push(
4:   org.objectweb.ccm.chat.HandleMessage event) {
5:   // Display the new message in the main message area
6:   mainMessageArea.append(
7:     "<" + event.sender + ">  " +
8:     event.message + "\n");
9: }
```

10. Some notes about the code:

• 1: Remove the @generated directive.

• 6: For each incoming message, take the contents of the message and add it to the main message area so that the user can see it.
Chapter 6. Scenario Editor

Once the component types have been defined, the system can be assembled into a scenario. The system assembly declares instances of the component types and declares how these instances are connected together to provide a complete application.

Opening the editor

When an OpenCCM project is created, a main scenario file is automatically created as well. To open the default scenario file:

- From the navigator view on the left, browse to the scenario directory: chat/specification/scenario
- Double click on the chat.scenario file.

The scenario editor has three tabs. The "Overview" tab shows what modules are imported by this scenario and allows users to import additional modules (only types from imported modules are visible). The "Table" tab displays all of the elements that compose the scenario. Right-clicking within the table will display a pop-up menu. That pop-up displays actions which allow elements to be added to the scenario, removed from the scenario, and various aspects of the elements to be modified. The "Graph" tab displays a graphical representation of the same scenario. Elements can be dragged around and positioned to make the graph more understandable. Like the "Table" tab, elements can be right clicked on to make various modifications to them.

Figure 6.1. Scenario editor
Creating the instances

The first component instance is the server. To create the server:

1. Switch to either the "Table" view, or the "Graph" view.
2. Right click somewhere within the main editor area.
3. Select "Add Component Instance # CCMComponent" from the pop-up menu. The new component instance wizard should appear.
4. For the name, use "server"
5. To select the type for the instance, click on the "Browse..." button. Select the "ChatServer" type from the dialog that appears.

The new server component should show up in both the "Graph" and "Table" tabs.

You can create any number of clients but we suggest creating two for the purposes of testing at the end of this tutorial. The clients are made in the same way that the above server was created, except the type of the clients is "ChatClient". Each of the clients should also have a unique name. In this case we will use "Client1" and "Client2".

Creating connections between the instances

Cadena offers four ways to create connections between components. This lets you choose which method is the best for you. The four methods are:

- The Tennessee Two Step Method (aka, Table-Add-Connector)
- The Oklahoma Mesa Method (aka, Table-New-Connection)
- The Arizona Blue and Yellow Method (aka, Graph-Add-Connector)
- The Texas Brazos Method (aka, Graph-New-Connection)

Try out each method and see which one works best for you. Then use each method to create all the necessary connections.

The Tennessee Two Step Method

This method takes place in the Table View. You will have to specify both the client and the server under the "Binding" drop-down menu.

1. Make sure you are in the Table View.
2. Right click anywhere on the "Connections" table and go to "Add Connector- >CCMInterfaceConnector" as seen in Figure 6.2, “Add CCMInterfaceConnector”.
Figure 6.2. Add CCMInterfaceConnector

3. The **Port Binding Roles** window will pop up. Click on the **clientSide** under **Name** and set the binding to **ChatClient1.sendMessage** as seen in Figure 6.3, “Port Right Click”.

Figure 6.3. Port Right Click

4. Now click on **serverSide** and set the binding to **ChatServer.sendMessage** as seen in Figure 6.4, “Component Port Binding”.

Figure 6.4. Component Port Binding
5. Click on **Finish**.

Now a connection has been made between the Server and the Client1. You can switch to the **Graph View** to see the connection visually.

## The Oklahoma Mesa Method

This method takes place in the **Table View**. You will only have to specify the server under the "Binding" drop-down menu.

1. Make sure you are in the **Table View**.

2. In the **instances** table expand **ChatClient1** and right click on **handleMessage**.

3. Click on New Connection For **Port->CCMEventConnector.consumerSide** as seen in Figure 6.5, “Port Right Click”.

   **Figure 6.5. Port Right Click**

   ![Port Right Click](port_right_click.png)

4. Set **producerSide** binding to **ChatServer.handleMessage** as seen in Figure 6.6, “Component Port Binding”.

   **Figure 6.6. Component Port Binding**

   ![Component Port Binding](component_port_binding.png)

5. Click **Finish**.

Now another connection has been made between the Server and Client1. You can switch to the **Graph View** to see the connection visually.
The Arizona Blue and Yellow Method

This method takes place in the **Graph View**. You will have to specify both the client and the server under the "Binding" drop-down menu.

1. Make sure you are in the **Graph View**.

2. Right click any where in the **Graph View** window and select "Add Connector->CCMInterfaceConnector" as seen in Figure 6.7, “Port Right Click”.

![Figure 6.7. Port Right Click](image)

3. In the **Bind Connector Roles** window set the **clientSide** Binding to **ChatClient2.sendMessage**. Then set the **serverSide** Binding to **ChatServer.sendMessage** and click **Finish** as seen in Figure 6.8, “Port Right Click”.

![Figure 6.8. Port Right Click](image)

You have now created another connection between the Server and Client2.

The Texas Brazos Method

This method takes place in the **Graph View**. You will only have to specify the client under the "Binding" drop-down menu. This is the easiest way to make a connection.

1. Make sure you are in the **Graph View**.

2. Right click on the "**handleMessage**" port of the **server instance** and it will show a pop-up menu.(context menu)
3. Select **New Connection for Port -> CCMEventConnector.producerSide** from the pop-up menu. A wizard dialog appears which will walk you through setting up the new connection. This can be seen in Figure 6.9, “Port Right Click”.

**Figure 6.9. Port Right Click**

4. In bindings select the client you want to have on the consumer side as seen in Figure 6.10, “Component Port Binding”.

**Figure 6.10. Component Port Binding**

You have now created a connection between the Server and Client2. This is the last connection that needs to be made for the program to function properly. You can now continue to the Scenario Execution.
Chapter 7. Scenario Execution

Once all of the previous steps are completed, the scenario can be executed.

Starting the system

To start the system:

1. Right click on the scenario that was created in the navigator view on the left. In this case it is called "chat.scenario" in chat/specification/scenario.

2. Select "OpenCCMActions->Start OpenCCM Scenario" from the popup menu.

The console view should now appear underneath the editor view. A couple of things should be displayed in the console. First, the result of running an Ant script that creates all of the necessary archives. This should only take a few seconds and after the step is complete, the console is cleared. Next, the OpenCCM platform is started. A number of lines should scroll by on the console. The last line should be "The demonstration chat is ready to be used ...". The main windows of the chat clients should then appear and they can now be used to send messages back and forth.

Figure 7.1 shows that you can change the name of each chat client and send messages back and forth.

Figure 7.1. Running system

Stopping the Scenario

Once you are done experimenting with the clients you should stop the scenario. To do this you will:

1. Right click on the "chat" project in the navigator view on the left.

2. Select "OpenCCMActions->Stop OpenCCM Project" from the popup menu.
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cadena</td>
<td>An Eclipse-based extensible integrated modeling and development framework for component-based systems.</td>
</tr>
<tr>
<td>TinyOS</td>
<td>An open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks.</td>
</tr>
<tr>
<td>nesC</td>
<td>An extension to the C programming language designed to embody the structuring concepts and execution model of TinyOS.</td>
</tr>
<tr>
<td>Eclipse</td>
<td>An open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. When we refer to Eclipse it is usually as an IDE or platform and not the project or community.</td>
</tr>
<tr>
<td>workspace</td>
<td>An Eclipse term that refers to the central hub for all user data. This is a specific folder/directory. A good quote from the Eclipse website is &quot;you can think of the platform workbench as a tool that allows the user to navigate and manipulate the workspace&quot;.</td>
</tr>
<tr>
<td>project</td>
<td>An Eclipse term that refers to a specific type of resource in the workspace. To be more specific, a workspace contains a collection of projects. Projects contain files and folders.</td>
</tr>
<tr>
<td>Module File</td>
<td>A Cadena term that refers to a file that contains a Cadena Module.</td>
</tr>
<tr>
<td>Scenario File</td>
<td>A Cadena term that refers to a file that contains a Cadena Scenario.</td>
</tr>
<tr>
<td>Scenario</td>
<td>A Cadena term that refers to a collection of instances (component, scenario, and connector) that define a modeled application.</td>
</tr>
<tr>
<td>Module</td>
<td>A Cadena term that refers to the description of the types available in the model which will be used at the Scenario tier. Modules contain definitions of Types that are used to define Scenario instances.</td>
</tr>
<tr>
<td>Style</td>
<td>A Cadena term that refers to the description of the platform that will be modeled at the other tiers of Cadena (module and scenario tiers). In other words, the style helps define a language to use in the Module tier. Styles contain definitions of Kinds (and Meta-Kinds) that are used to define Module Types.</td>
</tr>
<tr>
<td>nesC Interface</td>
<td>A TinyOS/nesC term that refers to a collection of methods (or method signatures) with a name. In nesC, components (modules and configurations) provide and use interfaces.</td>
</tr>
<tr>
<td>nesC Module</td>
<td>A TinyOS/nesC term that refers to a component that holds logic. This uses and provides interfaces, commands, and events. It also holds the logic that maps to the defined interfaces, commands, and events.</td>
</tr>
<tr>
<td>nesC Configuration</td>
<td>A TinyOS/nesC term that refers to a component that does not hold logic. A configuration defines a collection of components (modules and configurations)</td>
</tr>
</tbody>
</table>
and connectors as well as an optional collection of interfaces, commands, and events that it uses and provides. This holds no logic.

<p>| Nature | An Eclipse term that refers to flags set on Eclipse projects. These flags help Eclipse behave in a prescribed way. For example, certain actions, features, and builders are only available in projects with certain natures. For example, the Cadena Specification Path can only be defined in a project with a Cadena nature. |
| Specification Path | A Cadena term that refers to the path Cadena uses to find the model specifications available in a project. This includes three distinct paths for styles, modules, and scenarios. |
| Interface Type | ... |
| Component Type | ... |
| Component Instance | ... |
| Scenario Instance | ... |
| TinyOS Module | A Cadena/nesC term that refers to a Cadena Module that is set to use the nesC style. |
| TinyOS Scenario | A Cadena/nesC term that refers to a Cadena Scenario that is set to use the nesC style. |
| Architectural Definition Language (ADL) | ... |
| Product-Line Development | ... |
| Software Product Lines (SPL) | ... |
| Middleware | ... |
| Type | ... |
| Service | ... |
| Meta Model | ... |
| Component | ... |
| Interface | ... |
| Connector | ... |
| Meta Kind | ... |
| Kind | ... |
| Platform | ... |
| Port Option | ... |
| Role | ... |
| Interface Kind | ... |</p>
<table>
<thead>
<tr>
<th>Glossary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Kind</td>
<td>...</td>
</tr>
<tr>
<td>Connector Kind</td>
<td>...</td>
</tr>
<tr>
<td>Instance</td>
<td>...</td>
</tr>
<tr>
<td>Level</td>
<td>...</td>
</tr>
<tr>
<td>Layer</td>
<td>...</td>
</tr>
<tr>
<td>Assembly</td>
<td>...</td>
</tr>
</tbody>
</table>
Bibliography


[nesC:URL] “nesC Web Site”.

[TinyOS:URL] “TinyOS Web Site”.

[Cadena:URL] “{\sc Cadena} Web Site”.