SystemJ Quick-Start Guide
Version 2.0

By now you've installed the SystemJ Development kit for Eclipse and likely heard about its strengths in designing and programming concurrent and distributed software systems, so what now?

If you just want to jump straight in to development this 'Quick-Start' guide is what shows you how to use the toolset in Eclipse, or else, if you want a more detailed demonstration of SystemJ itself please have a look at the Detailed SystemJ guide. Please note that these guides are available in several locations; the website (http://www.systemjtechnology.com/index.php?option=com_content&view=article&id=62&Itemid=73) and in the Eclipse documentation at Help>Help Contents>SystemJ Developer Guide>Development Guides. Additionally, there is an FAQ available at http://systemjtechnology.com/index.php?option=com_content&view=article&id=48&Itemid=41. Should you have any further questions on what is needed or should be explained better please send an email to info@systemjtechnology.com, or file a bug report at http://www.systemjtechnology.com/trac/systemjcompiler/newticket and we will get back to you as soon as possible.

As per the installation guide, by this point you are now assumed to have a licensed version of SystemJ installed on your machine.

To begin with, there is a conventional structure to which SystemJ projects adhere as shown in Figure 1. Firstly, the .sysj file should be kept at the root of project whereas the Java packages can be placed in the src folder. A SystemJ project is organised in this manner primarily for clarity due to the typical project configuration; one or two .sysj files and several Java packages Following this is the naming convention which helps the tools to relate different file types to one another. For instance, the .xml file which defines the network structure used by SC.sysj should be called SC.xml; the tools have been designed to automatically search for files which satisfy this convention when the respective .sysj file is launched. The same convention applies to the .sjio debugging file.

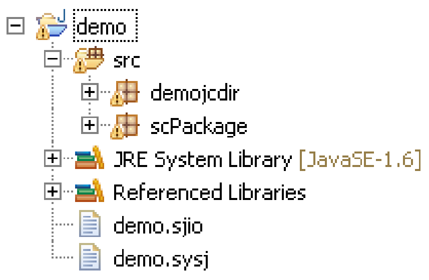
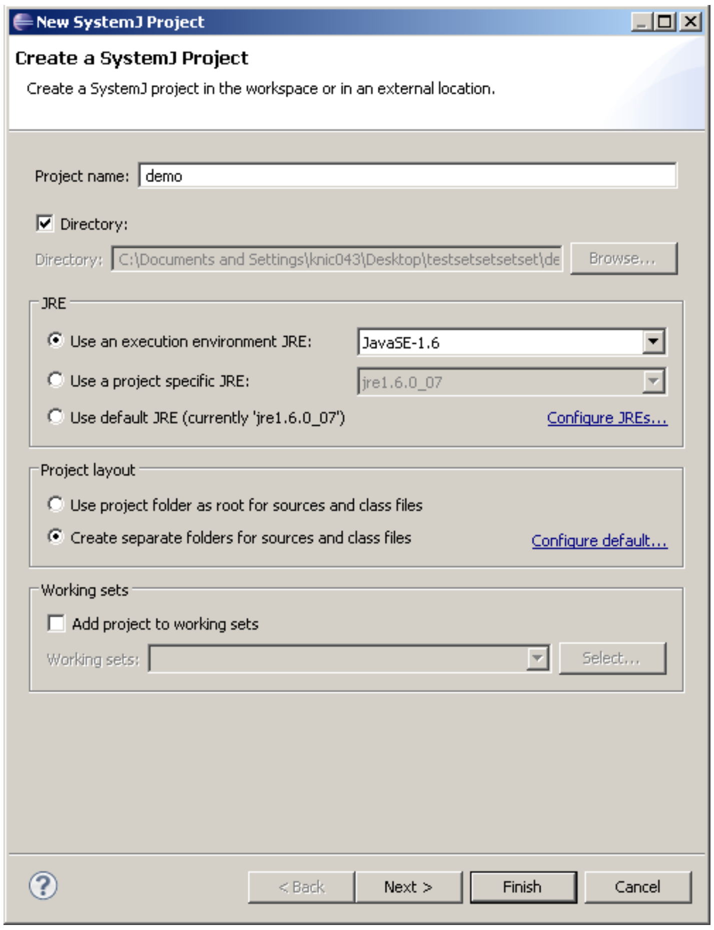


Figure 1: Project Structure

Now that you understand the expected project layout you can create one yourself using the SystemJ project wizard shown in Figure 2. You can provide only a name and click finish to generate a project with the default settings or you can page through and customise it further. If the project generated is lacking something that you require you can create that file through the .sysj,

.xml and ,sjio wizards as you see fit. Like the project wizard, you can stick with the defaults or customise it as long as you maintain the naming conventions as shown in Figure 3.


New SystemJ Project

Create a SystemJ Project

Create a SystemJ project in the workspace or in an external location.

Project name: demo

[✓] Directory:

Directory: C:\Documents and Settings\knic043\Desktop\testsetsetsetset\de

JRE

(●) Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: jre1.6.0_07

( ) Use default JRE (currently 'jre1.6.0_07')

Project layout

( ) Use project folder as root for sources and class files

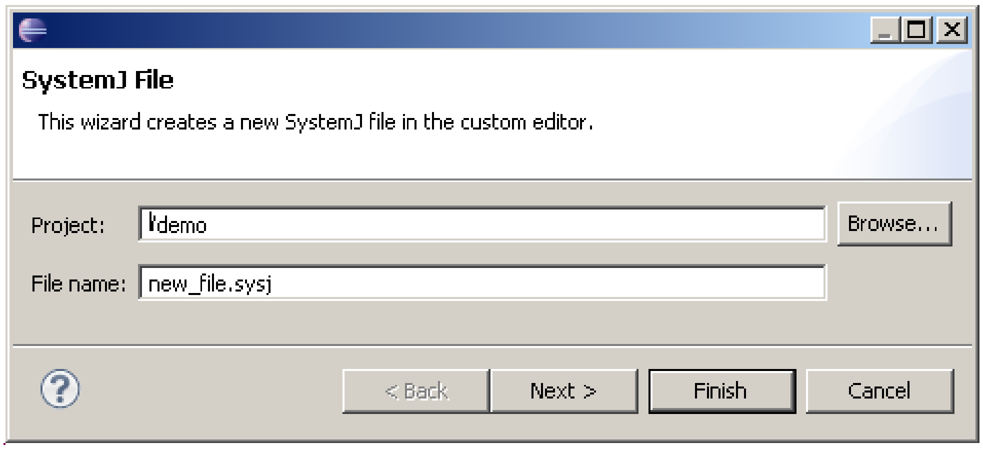
(●) Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

Working sets:

Figure 2: SystemJ Project Wizard


SystemJ File

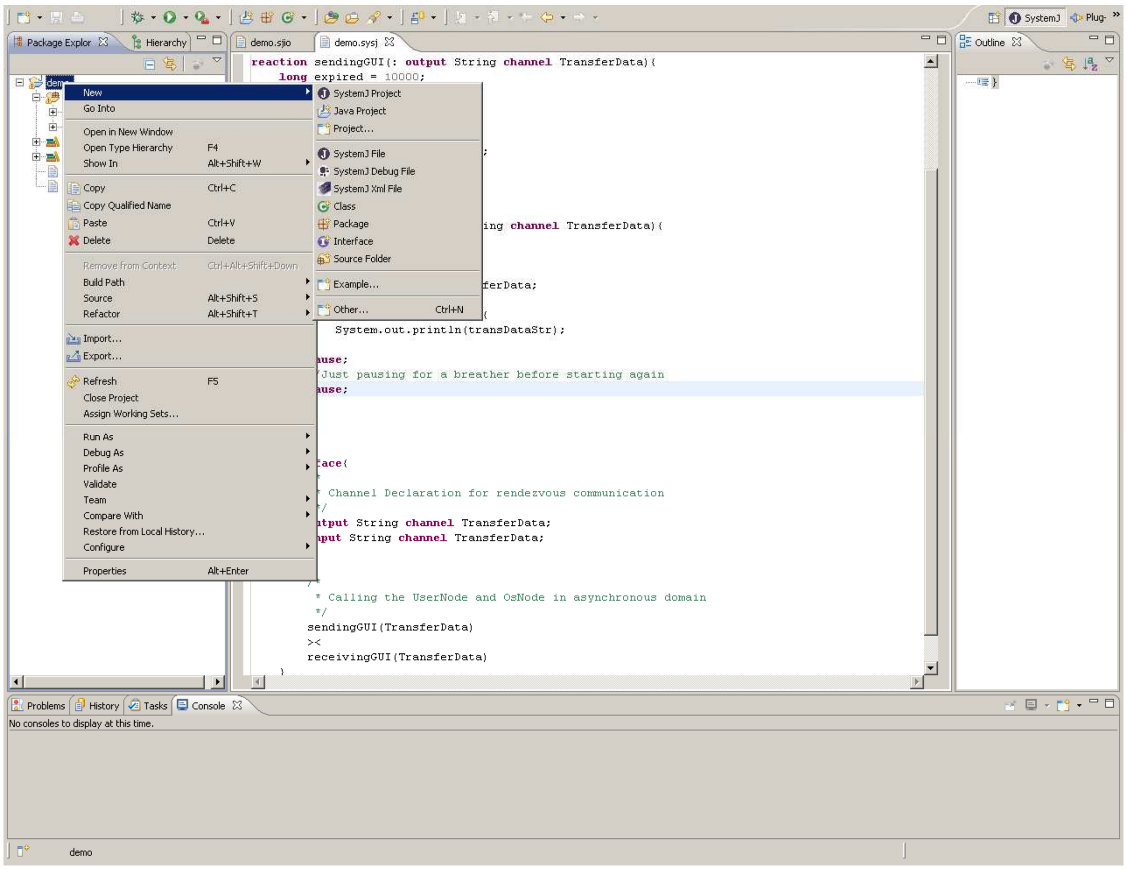
This wizard creates a new SystemJ file in the custom editor.

Project: /demo

File name: new_file.sysj

Figure 3: SystemJ File Wizard

As shown in Figure 4 the demo project makes use of the non-networked template and includes a demo.sysj and a demo.sjio file which are automatically opened in the SystemJ Code Editor when the project is generated and the user is prompted to open the SystemJ perspective. The code editor offers syntax highlighting among other useful features which are complemented well by the perspective that adds some shortcuts to the file and projects menus as well as organising the views on the workbench in a way conducive to SystemJ development.

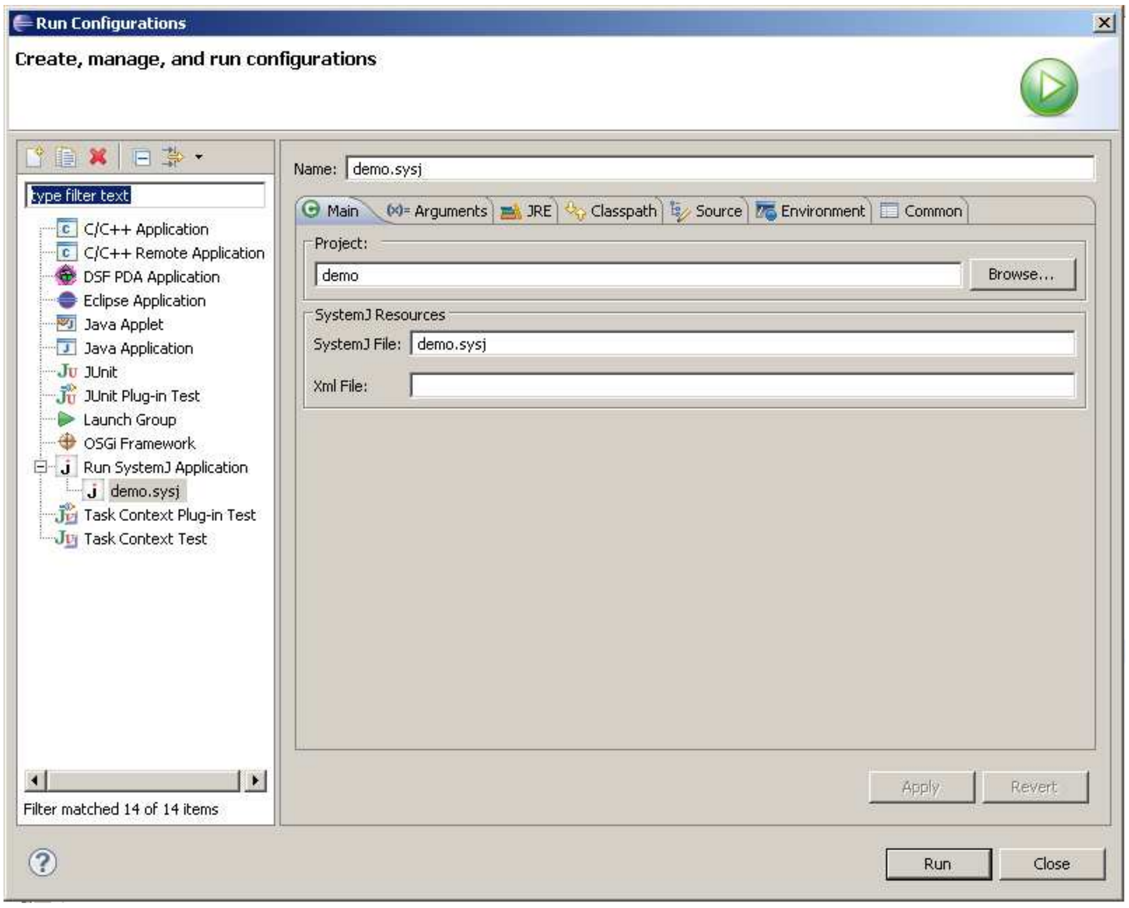


```
reaction sendingGUI(: output String channel TransferData){
    long expired = 10000;
                                    ;
                                    ing channel TransferData){
                                    ferData;
                                    {
            System.out.println(transDataStr);
        use;
        /Just pausing for a breather before starting again
        use;
        ace(
        *
        * Channel Declaration for rendezvous communication
        */
        tput String channel TransferData;
        put String channel TransferData;
      /*
       * Calling the UserNode and OsNode in asynchronous domain
       */
      sendingGUI(TransferData)
      ><
      receivingGUI(TransferData)
    }
```

No consoles to display at this time.

Figure 4: SystemJ Perspective – Includes ‘Outline’ to the right, ‘Code Editor’ in the middle and the shortcuts on the ‘right-click > New’, or ‘File > New’ menu, on the left.

And finally, the most important tools are those which enable the running and debugging of SystemJ projects. Within Eclipse the process of a launch configuration is shortcut>delegate>launch, and through the tools you, the user, can take action at either of the first two levels. The shortcuts have been made context specific and are available at the project level, the respective compiled directory level and at the .sysj file itself under ‘Run As > SystemJ Application’ and ‘Debug As > SystemJ Application’. Moreover, what a shortcut does is essentially configuring a delegate with default parameters and then launching them. Should you want more control over the launch configuration you can easily alter the parameters at the Run > Run Configurations.. (Figure 5) or Debug>Debug Configurations.. (Figure 6) options respectively.


Run Configurations

Create, manage, and run configurations

Name: demo.sysj

Project: demo

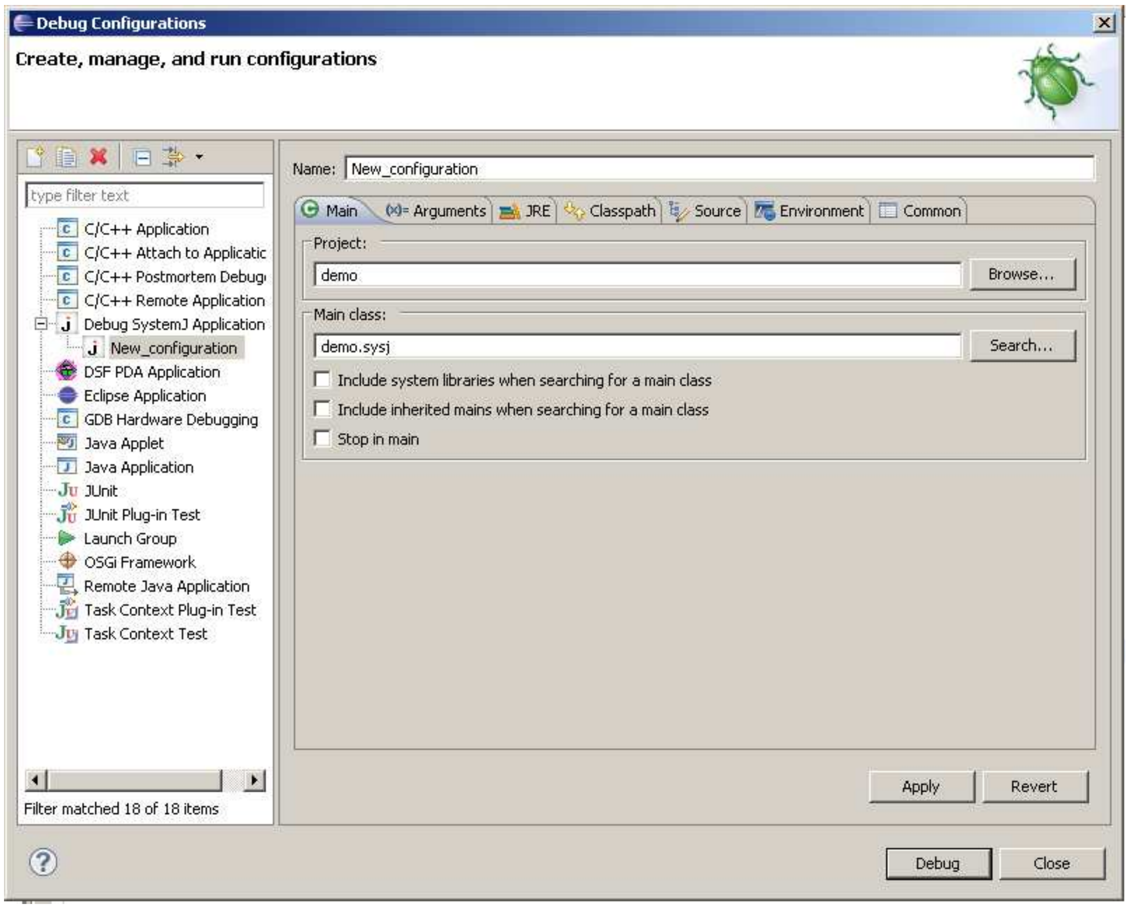
SystemJ Resources

SystemJ File: demo.sysj

Xml File:

Figure 5: Run Delegate

Take note though, that the debug launch type does not require an .xml due to the manner in which it is compiled. Because of this, as seen in the images, the two delegates differ slightly in what settings need to be configured for a launch.


Debug Configurations

Create, manage, and run configurations

Name: New_configuration

Project: demo

Main class: demo.sysj

☐ Include system libraries when searching for a main class

☐ Include inherited mains when searching for a main class

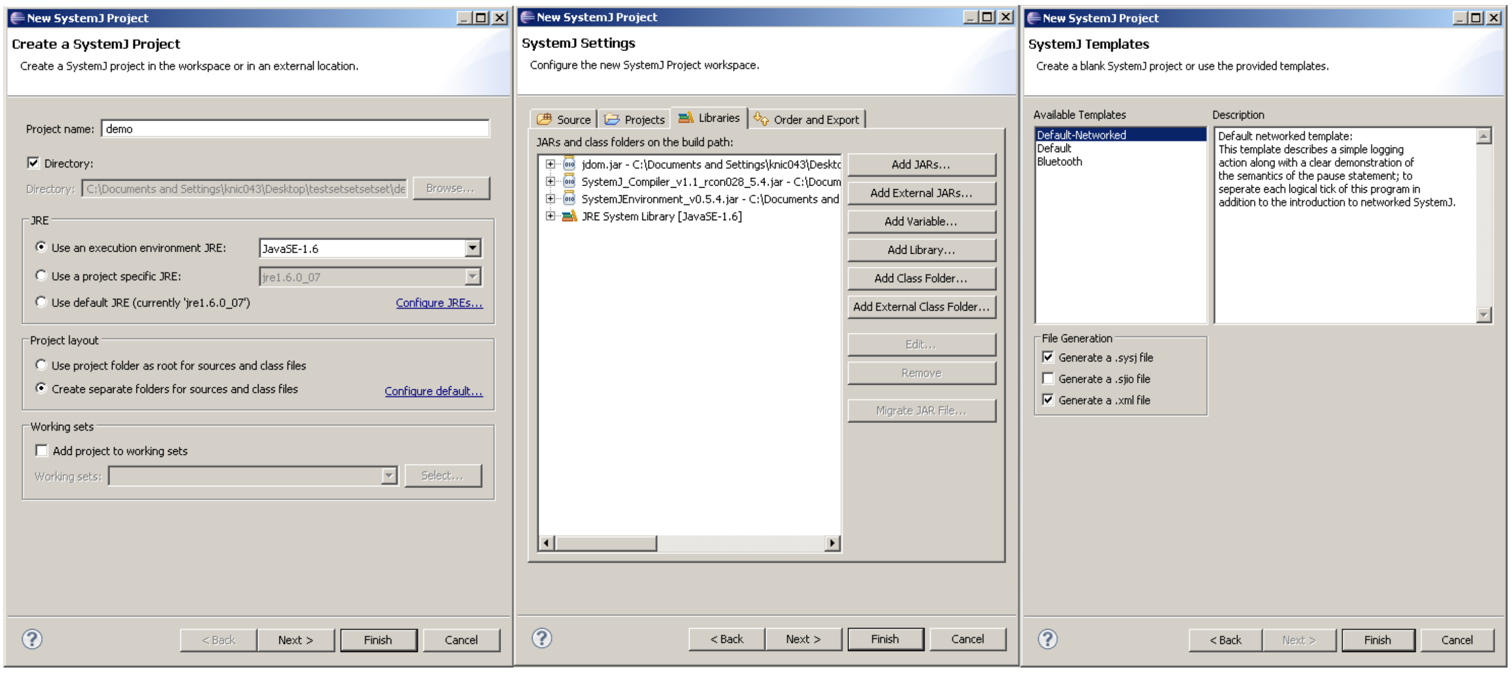
☐ Stop in main

Filter matched 18 of 18 items

Figure 6: Debug Delegate

SystemJ can also be used for programming networked systems. The project setup for network programming differs slightly compared to the aforementioned project setup. To explain network setup we revisit the setup from the start. Figure 10 shows the three initial dialog boxes that are used to setup the main project components. The first two dialog boxes set the name of the project and the libraries as previously seen in Figure 2.Essentially the difference lies in the third dialog box as, compared to the one in Figure 2, the designer needs to tick the ‘Generate a .xml file’ option along with the .sysj file option. SystemJ uses an external XML (Extension Markup Language) file to bind the channels and interface signals with the underlying physical layer.

Figure 11 shows an example SystemJ program with two clock-domains acting as a server and client communicating with each other via channels bound to the underlying physical network layer. The top left screen shows the SystemJ program code, the top right shows the corresponding XML file, the bottom left highlights the TransferId output channel binding for clock-domain UserNode and the bottom right highlights the TransferId input channel binding for clock-domain OsNode.



New SystemJ Project

Create a SystemJ Project

Create a SystemJ project in the workspace or in an external location.

Project name: demo

[✓] Directory:

Directory: C:\Documents and Settings\knic043\Desktop\testsetsetsetset\de

JRE

(●) Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: jre1.6.0_07

( ) Use default JRE (currently 'jre1.6.0_07')

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

New SystemJ Project

SystemJ Settings

Configure the new SystemJ Project workspace.

JARs and class folders on the build path:

- jdom.jar - C:\Documents and Settings\knic043\Desktc
- SystemJ_Compiler_v1.1_rcon028_5.4.jar - C:\Docum
- SystemJEnvironment_v0.5.4.jar - C:\Documents and
- JRE System Library [JavaSE-1.6]

New SystemJ Project

SystemJ Templates

Create a blank SystemJ project or use the provided templates.

Available Templates: Default-Networked, Default, Bluetooth

Description: Default networked template:
This template describes a simple logging action along with a clear demonstration of the semantics of the pause statement; to seperate each logical tick of this program in addition to the introduction to networked SystemJ.

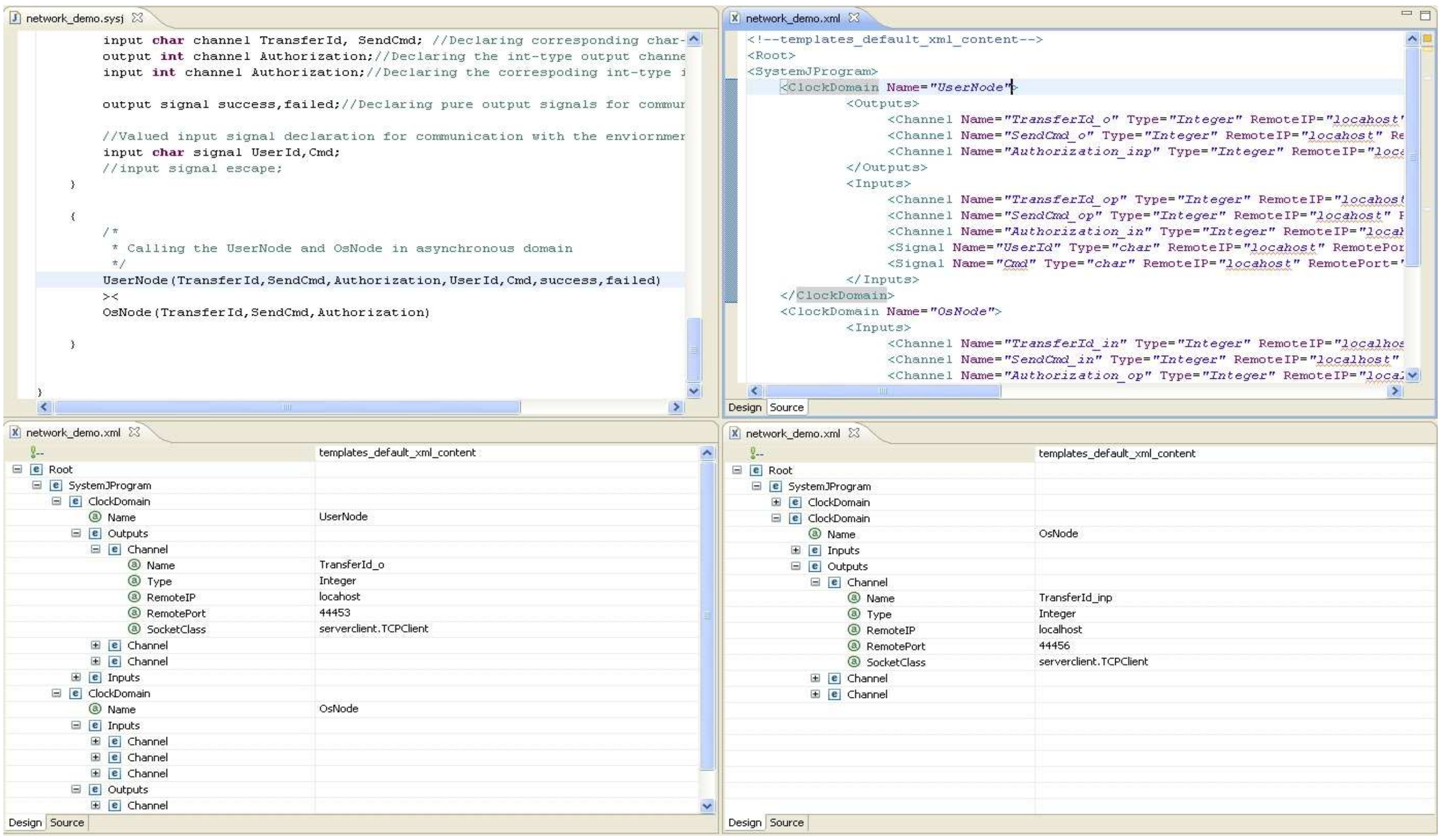
File Generation

[✓] Generate a .sysj file

[ ] Generate a .sjio file

[✓] Generate a .xml file

Figure 10: Setting up a SystemJ project for programming networked systems



network_demo.sysj

```
        input char channel TransferId, SendCmd; //Declaring corresponding char-
        output int channel Authorization;//Declaring the int-type output channe
        input int channel Authorization;//Declaring the correspoding int-type i

        output signal success,failed;//Declaring pure output signals for commur

        //Valued input signal declaration for communication with the enviornmer
        input char signal UserId,Cmd;
        //input signal escape;
    }

    {
        /*
         * Calling the UserNode and OsNode in asynchronous domain
         */
        UserNode(TransferId,SendCmd,Authorization,UserId,Cmd,success,failed)
        ><
        OsNode(TransferId,SendCmd,Authorization)

    }

}
```

network_demo.xml

```
<!--templates_default_xml_content-->
<Root>
<SystemJProgram>
    <ClockDomain Name="UserNode">
            <Outputs>
                <Channel Name="TransferId_o" Type="Integer" RemoteIP="locahost'
                <Channel Name="SendCmd_o" Type="Integer" RemoteIP="locahost" Re
                <Channel Name="Authorization_inp" Type="Integer" RemoteIP="loca
            </Outputs>
            <Inputs>
                <Channel Name="TransferId_op" Type="Integer" RemoteIP="locahost
                <Channel Name="SendCmd_op" Type="Integer" RemoteIP="locahost" F
                <Channel Name="Authorization_in" Type="Integer" RemoteIP="local
                <Signal Name="UserId" Type="char" RemoteIP="locahost" RemotePor
                <Signal Name="Cmd" Type="char" RemoteIP="locahost" RemotePort="
            </Inputs>
    </ClockDomain>
    <ClockDomain Name="OsNode">
            <Inputs>
                <Channel Name="TransferId_in" Type="Integer" RemoteIP="localhos
                <Channel Name="SendCmd_in" Type="Integer" RemoteIP="localhost"
                <Channel Name="Authorization_op" Type="Integer" RemoteIP="local
```

network_demo.xml

```
                                              templates_default_xml_content
Root
  SystemJProgram
    ClockDomain
      Name                                    UserNode
      Outputs
        Channel
          Name                                TransferId_o
          Type                                Integer
          RemoteIP                            locahost
          RemotePort                          44453
          SocketClass                         serverclient.TCPClient
        Channel
        Channel
      Inputs
    ClockDomain
      Name                                    OsNode
      Inputs
        Channel
        Channel
        Channel
      Outputs
        Channel
```

network_demo.xml

```
                                              templates_default_xml_content
Root
  SystemJProgram
    ClockDomain
    ClockDomain
      Name                                    OsNode
      Inputs
      Outputs
        Channel
          Name                                TransferId_inp
          Type                                Integer
          RemoteIP                            localhost
          RemotePort                          44456
          SocketClass                         serverclient.TCPClient
        Channel
        Channel
```

Figure 11: An Eclipse session split into four windows.