NIST Technical Note 2053

Users Guide to Type277 Loosely-Coupled Integration of TRNSYS with Java

Farhad Omar

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2053

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Technical Note 2053

Users Guide to Type277 Loosely-Coupled Integration of TRNSYS with Java

Farhad Omar
Energy and Environment Division
Engineering Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2053

August 2019

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Technical Note 2053
Natl. Inst. Stand. Technol. Tech. Note 2053, 28 pages (August 2019)
CODEN: NTNOEF

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2053

Software Disclaimers

Copyright Notice for External Libraries

Table of Contents

List of Figures

1. Purpose

The purpose of this document is to provide instructions for using a newly developed TRNSYS component (Type277). Type277 utilizes socket communication to provide loosely-coupled integration between TRNSYS and Java-based components. Bi-directional flow of information is facilitated using the client-server architecture of Java. Type277 is compiled as both a 32-bit and 64-bit dynamic link library to support co-simulation with TRNSYS17 and TRNSYS18 projects. An application of Type277 is demonstrated by modifying an example TRNSYS project. The Java project and files, TRNSYS modified example projects, and Type277 dynamic link libraries are being made available to the public through the NIST GitHub site.

2. Introduction

This section describes the use of NIST-developed software (Type277) for performing co-simulation by facilitating the exchange of data between a local server and a model developed with the TRaNsient SYstem Simulation tool (TRNSYS) [1]. A detailed description of the co-simulation environment is provided in [2]. Figure 1 shows a schematic representation of the co-simulation environment, describing the interaction of a TRNSYS model and a local server. Figure 1 also shows that Type277 enables a TRNSYS model to participate in a co-simulation with a third-party software tool through the local server. An example of co-simulation between TRNSYS and MATLAB is described in Section 4.2. The connection between TRNSYS and the local server is accomplished through socket communication, while the connection between the local server and the third-party software can be established using proxies or other available libraries.

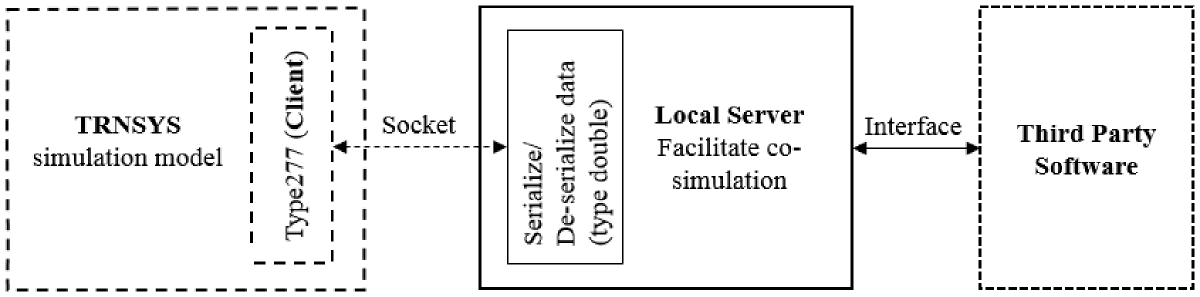


Figure 1. A schematic representation of the co-simulation environment

The co-simulation environment shown in Figure 1 was implemented in a Java project, which includes all the necessary folders and files used for loosely-coupled integration of TRNSYS with a Java environment. The key idea behind this approach is to demonstrate the use of Type277 and enable users to modify the code to meet their needs. Figure 2 shows a Unified Modelling Language (UML) class diagram of the Java project without the interface to a third-party software tool.

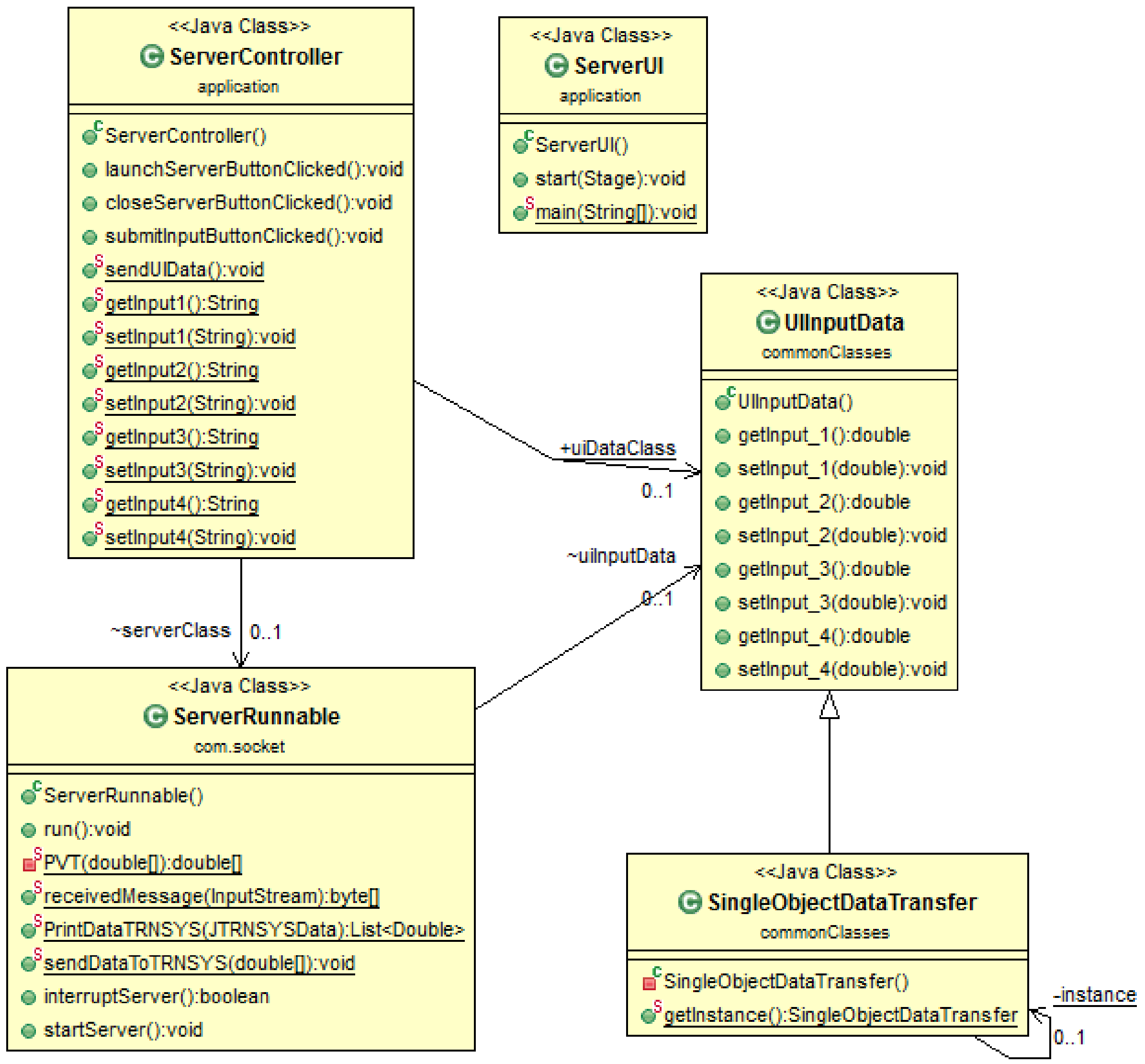


Figure 2. A UML class diagram of the JAVA classes used for implementing a local server located in the project repository

The UML class diagram shown in Figure 2, describes the relationship of various classes used to support the loosely-coupled integration of TRNSYS and Java. The `ServerUI` is the main class for rendering the graphical user interface (GUI) and creating an instance of the controller class (`ServerController`). The `ServerController` class controls the functionality of the GUI and instantiates an object of the `ServerRunnable` class, which implements the server and facilitates the exchange of data between TRNSYS and Java. The instructions for accessing TRNSYS data in `ServerRunnable` class are presented in Section 4.1. The `ServerController` class also instantiates an object of `UIInputData` class to capture the user inputs (e.g., `getInput_1()` shown in Figure 2) from the GUI and transfer them to other classes using a Singleton object.

2.1. System Requirements

1. Personal computer able to run Microsoft Windows 7 or 10 versions;
2. Java Standard Edition (SE) 8 for Windows (JRE 1.8) or Java Development Kit (JDK 1.8);
3. JavaFX 11 libraries (included in the project, see Appendix) or earlier versions;
4. MATLAB Java application programming interface `matlabcontrol-4.1.0.jar` (included in the project, see Appendix); and
5. Google's Protocol Buffer library `protobuf-java-3.6.1.jar` (included in the project, see Appendix).

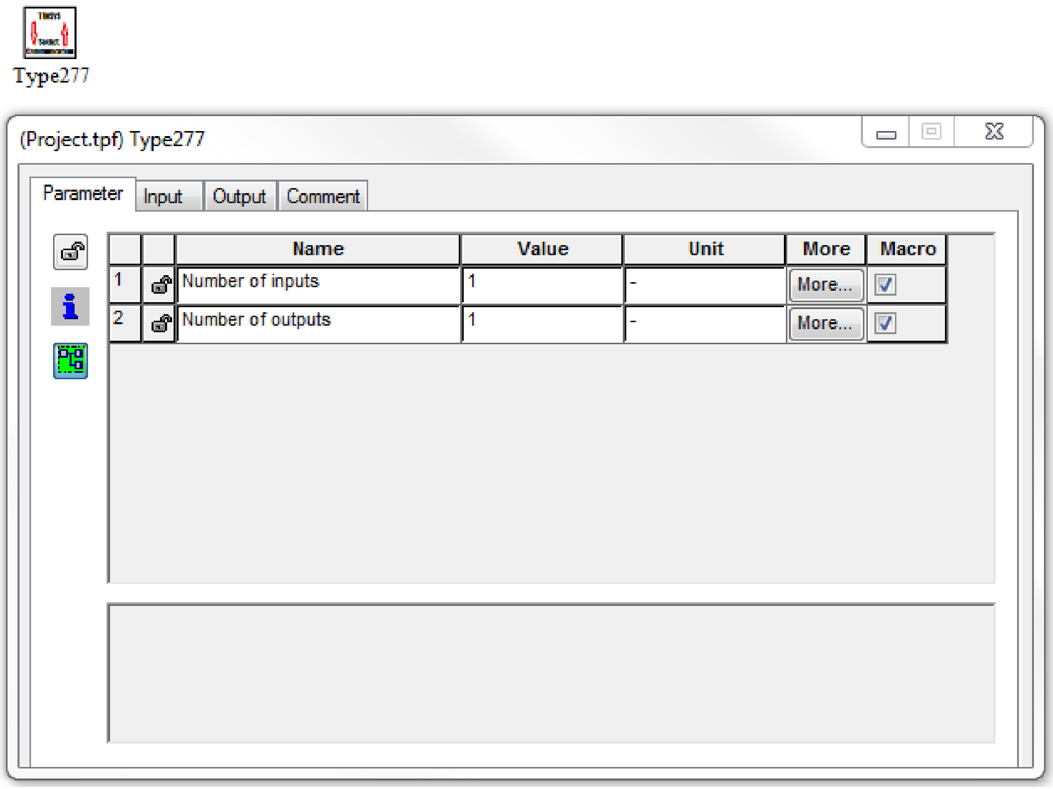
Note that an implementation of protocol buffers used in this project is not compatible with JDK 11. This project was created and tested using JRE 1.8 or JDK 1.8 for Windows. Also, note that MATLAB software must be available in your system to use the example described in Section 4.2.

3. Using Type277 in TRNSYS

TRNSYS is a modular and extendable simulation environment that consists of a suite of software tools designed to accommodate transient simulation of multi-zone buildings and other thermal systems. The main user interface is Simulation Studio, in which users can setup projects by graphically connecting model components. Each component is mathematically described in the TRNSYS simulation engine and has a corresponding graphical representation (proforma) in Simulation Studio. A proforma is a black-box description of inputs, outputs, and parameters. TRNSYS components are commonly referred to as Types and are identified by a number which relates a component to the model of that component written as a subroutine. An advantage of the TRNSYS modular architecture is its ability to support the integration of user-defined types, such as Type277.

Type277 is a dynamic-linked library (DLL) designed to facilitate the loosely-coupled integration of a simulation model in TRNSYS with components written in other programming languages, e.g., Java and MATLAB. Type277 utilizes socket communication to establish a connection between TRNSYS and a local server; effectively converting a TRNSYS model into a client of the server. To ensure a reliable exchange of information between a server and the client, the data of the primitive type Double are serialized using Google's protocol buffers [3]. Type277 is written in C++ and compiled into 32-bit and 64-bit DLLs. This document presents a step-by-step instruction for using Type277 in a TRNSYS model while exchanging data with a server implemented in Java. Alternative implementation of the local server in C++ and Python are also made possible by serializing data with protocol buffers. Relevant C++ and Python libraries are included with the Java project download from NIST's GitHub site.

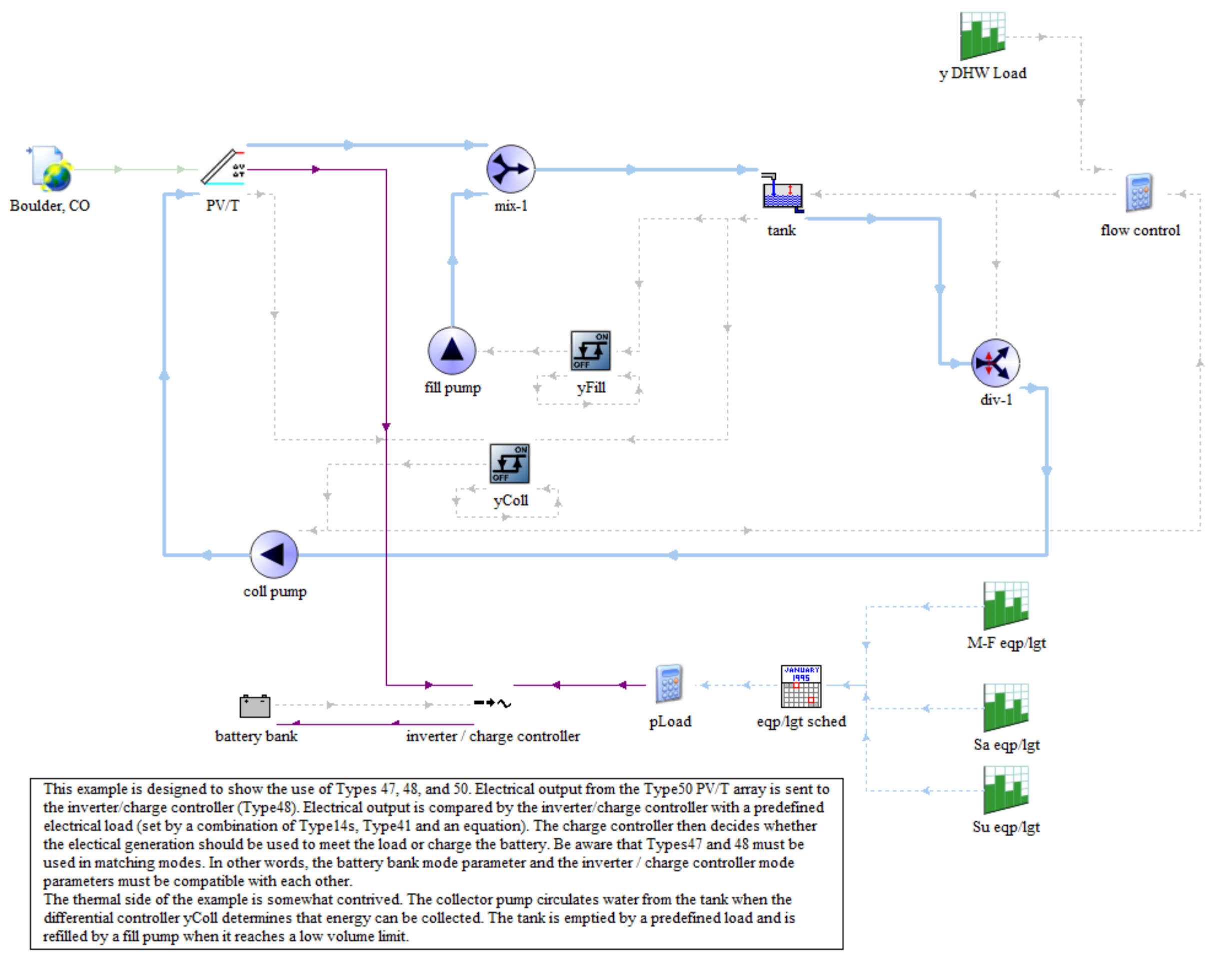
Like all standard types in TRNSYS, Type277 has a proforma as shown in Figure 3. As can be seen from Figure 3, the value associated with the first parameter sets the number of inputs and the value associated with the second parameter sets the number of outputs. Type277 uses these parameters to set the size of a vector (1 x n dimensional array) for exchanging data of the primitive type Double. The input data are sent to the server and the returned values can be connected to the Inputs of other types in a TRNSYS project.



		Name	Value	Unit	More	Macro
1		Number of inputs	1	-	More...	✓
2		Number of outputs	1	-	More...	✓

Figure 3. Type277 proforma

To demonstrate the use of Type277 in a TRNSYS model, the PVT example TRNSYS project was modified by replacing the pLoad Equation shown in Figure 4 with Type277. The goal is to calculate pLoad in Java and MATLAB and return the calculated values back to TRNSYS.



This example is designed to show the use of Types 47, 48, and 50. Electrical output from the Type50 PV/T array is sent to the inverter/charge controller (Type48). Electrical output is compared by the inverter/charge controller with a predefined electrical load (set by a combination of Type14s, Type41 and an equation). The charge controller then decides whether the electical generation should be used to meet the load or charge the battery. Be aware that Types47 and 48 must be used in matching modes. In other words, the battery bank mode parameter and the inverter / charge controller mode parameters must be compatible with each other.
The thermal side of the example is somewhat contrived. The collector pump circulates water from the tank when the differential controller yColl determines that energy can be collected. The tank is emptied by a predefined load and is refilled by a fill pump when it reaches a low volume limit.

Figure 4. The PVT example model in TRNSYS [1]

This is a simple calculation, but the purpose of using the PVT example is to demonstrate the process of exchanging data between TRNSYS and Java. More complex calculations can be performed in a similar manner. A case study for assessing the performance of six residential energy management control algorithms through co-simulation of TRNSYS and MATLAB facilitated by Type277 is presented in [2], [4].

3.1. Downloading JTRNSYS Project

This section provides instructions for downloading a Java project that contains all the necessary folders and files used for loosely-coupled integration of TRNSYS and the Java environment. It is assumed that the 64-bit version of TRNSYS 18 has been installed in the C:\TRNSYS18 folder.

1. Download the zip file or clone JTRNSYS project from https://github.com/usnistgov/JTRNSYS.git;
2. Extract *jtrnsys-master.zip* or clone to your desired location, e.g., C:\jtrnsys-master;
3. In the *jtrnsys-master* folder:

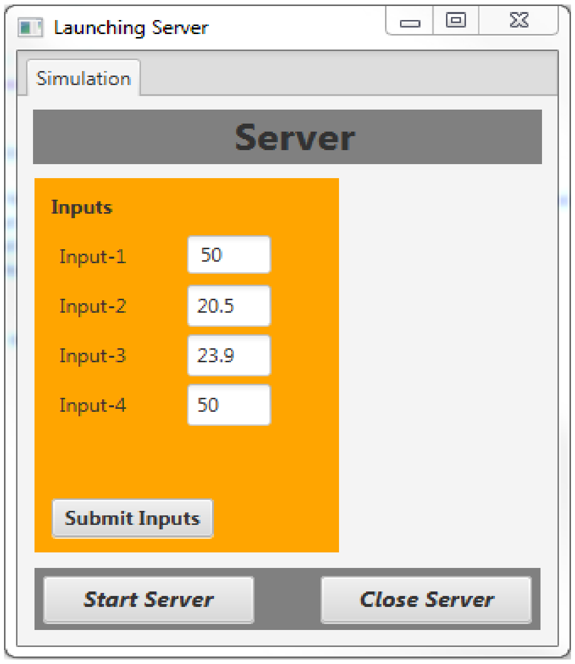
a. Navigate to the "TRNSYS 64bit" folder and copy the "Trnsys Socket Client" folder into the "C:\TRNSYS18\Studio\Proformas" folder;
b. Navigate to the "TRNSYS 64bit\PVT Example" folder and copy the PVTType277.tpf file into the "C:\TRNSYS18\Examples\PVT" or into another desired folder such as "C:\TRNSYS18\MyProjects";
c. Navigate to the "TRNSYS 64bit\Type277DLLs\ReleaseDLL" and copy the Type277.dll file into the release folder in "C:\ TRNSYS18\UserLib\ReleaseDLLs"; and
d. Navigate to the "TRNSYS 64bit\Type277DLLs\DebugDLL" and copy Type277.dll file into the debug folder in "C:\ TRNSYS18\UserLib\DebugDLLs".

To use the 32-bit version of Type277, follow step 3, but copy files from the "TRNSYS 32bit" folder instead of the "TRNSYS 64bit" folder. The 32-bit version of Type277 will work with both TRNSYS17 and the 32-bit version of TRNSYS18.

3.2. Co-Simulation between TRNSYS and Java

To perform co-simulation between TRNSYS and Java navigate to the jtrnsys-master folder and follow these instructions:

1. Double click the ServerUILaunch.bat file or type *java -jar "LaunchServerUI.jar"* in Windows command prompt to launch the GUI as shown in Figure 5; and
2. Double click the Start Server button to launch the server. Do not close the server, it must remain active to accept a client's connection. Note that this action may cause a windows firewall security alert, allow the process to proceed.



Input-1	50
Input-2	20.5
Input-3	23.9
Input-4	50

Figure 5. User interface for starting the server

As can be seen from Figure 5, the Server GUI provides a simple interface for capturing user input data, starting the server, and finally terminating the server and closing the GUI. The four generic input fields are provided as a mechanism for users to submit their desired values. For example, users can set heating and cooling setpoints or define threshold limits for controlling humidity levels. These inputs are captured as text and parsed as the primitive type Double in `ServerController` class. The number of input fields can be extended as needed either graphically by manipulating the Server GUI or editing the `JServer.fxml` file included in the JTRNSYS project. To capture user inputs, double click the Submit Inputs button. In the current implementation, the data from the GUI is stored in an object of `UIInputData` class. A method, `sendUIData()` in `ServerController` class, is provided for sending the GUI input data through an instance of a Singleton object. Note that this is an additional functionality and not necessary to launch the server or communicate with TRNSYS.

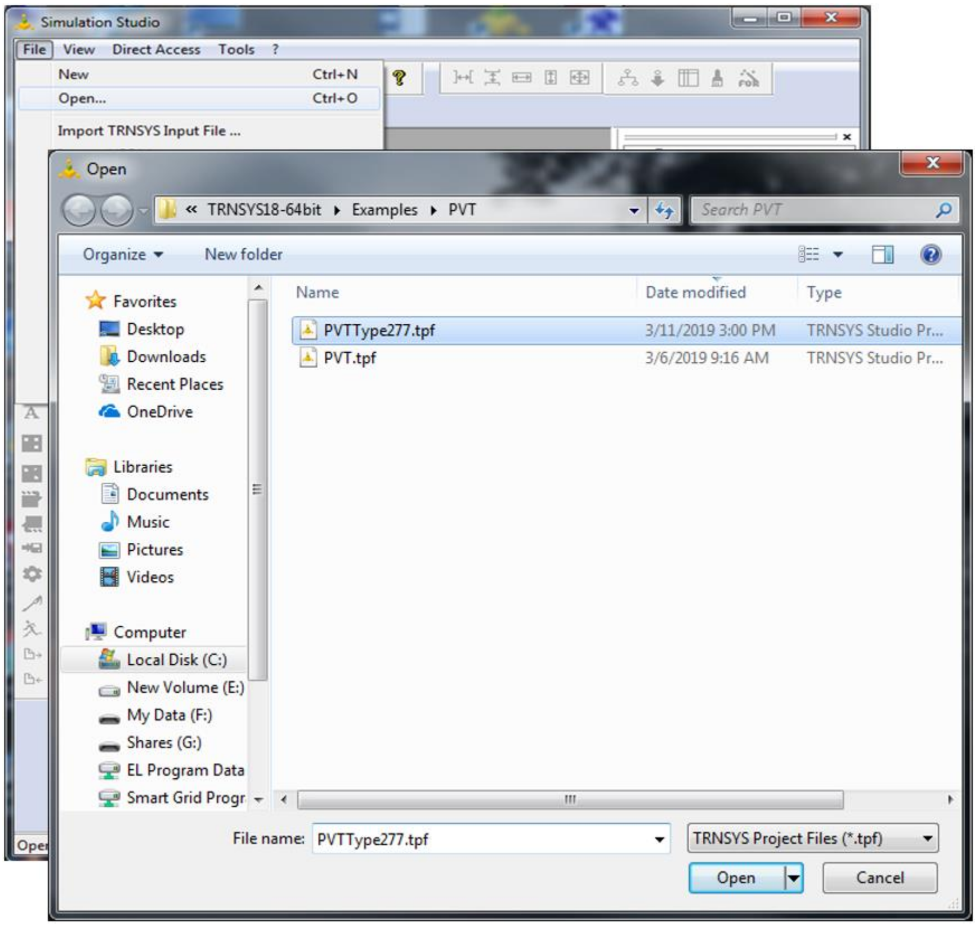
After launching the server, it is ready to listen for a client connection on port 1345, which is used by Type277 to communicate with the local server. The client is a TRNSYS simulation model that uses Type277. The port number (1345) was arbitrarily chosen, but it is hard-coded in Type277.

The next step is to simulate the modified version of the PVT project with Type277 as explained in the following section. Note that before simulating the TRNSYS model, the server must be running. In the absence of having an active server, TRNSYS will not throw an error if the model is simulated. Instead, the simulation will progress slowly, where the value for pLoad will be equal to zero.

3.2.1. Simulating the PVT Model in TRNSYS

The purpose of this section is to load the modified version of the PVT model (PVTType277) into the Simulation Studio and simulate it.

1. Open the 64-bit or 32-bit installation of Simulation Studio in TRNSYS18;
2. Select *File* → *Open…* from the menu and navigate to the folder where the PVT example for Type277 was copied in Step b of Section 3.1;
3. Select the PVTType277.tpf file and click *Open* as shown in Figure 6.

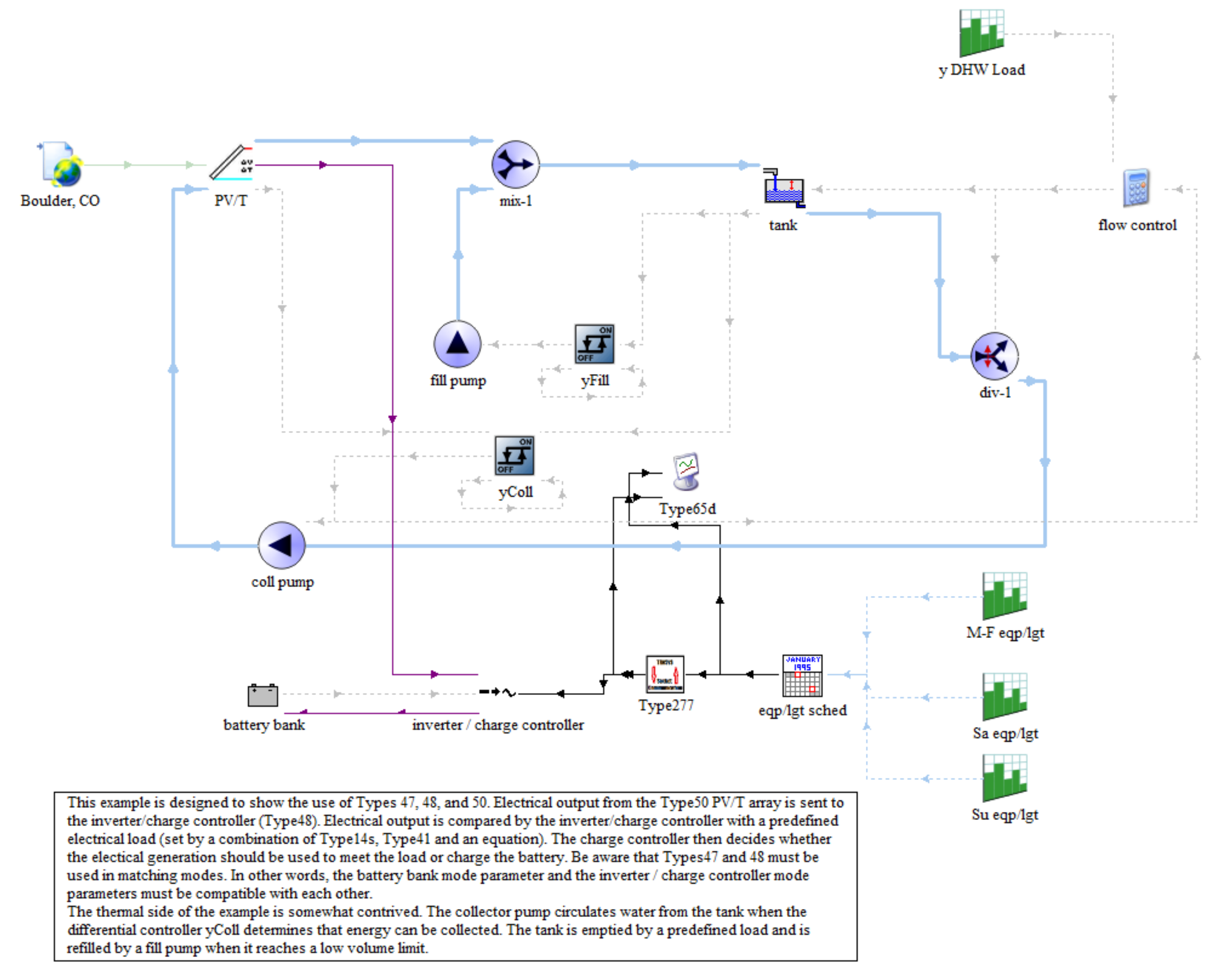


Name	Date modified	Type
PVTType277.tpf	3/11/2019 3:00 PM	TRNSYS Studio Pr...
PVT.tpf	3/6/2019 9:16 AM	TRNSYS Studio Pr...

File name: PVTType277.tpf
TRNSYS Project Files (*.tpf)

Figure 6. Open the PVTType277 model

The PVT model with Type277, shown in Figure 7, is loaded into Simulation Studio.



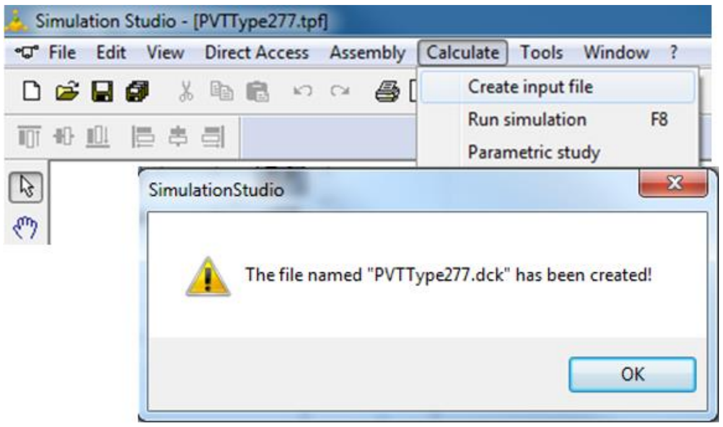
This example is designed to show the use of Types 47, 48, and 50. Electrical output from the Type50 PV/T array is sent to the inverter/charge controller (Type48). Electrical output is compared by the inverter/charge controller with a predefined electrical load (set by a combination of Type14s, Type41 and an equation). The charge controller then decides whether the electical generation should be used to meet the load or charge the battery. Be aware that Types47 and 48 must be used in matching modes. In other words, the battery bank mode parameter and the inverter / charge controller mode parameters must be compatible with each other.
The thermal side of the example is somewhat contrived. The collector pump circulates water from the tank when the differential controller yColl determines that energy can be collected. The tank is emptied by a predefined load and is refilled by a fill pump when it reaches a low volume limit.

Figure 7. The PVT model with Type277

A comparison of Figure 4 and Figure 7 shows that the Equation function for calculating pLoad in Figure 4 is replaced by the proforma for Type277. The input to Type277 is fLoad, and the output is pLoad.

4. In Simulation Studio:

   a. Navigate to the *Calculate* menu item and select *Create input file* from the drop-down menu as shown in Figure 8 and click *OK*; and
   b. Run the PVTType277 example by selecting the *Run* button in Simulation Studio. This step launches the TRNEXE software, showing the simulation progress and values for pLoad calculated in Java. Select the Graph 1 to see the results of this simulation as shown in Figure 9.


Simulation Studio - [PVTType277.tpf]
File Edit View Direct Access Assembly Calculate Tools Window ?
Create input file
Run simulation F8
Parametric study
SimulationStudio
OK

The file named "PVTType277.dck" has been created!

Figure 8. Create a TRNSYS input file for PVTType277 model

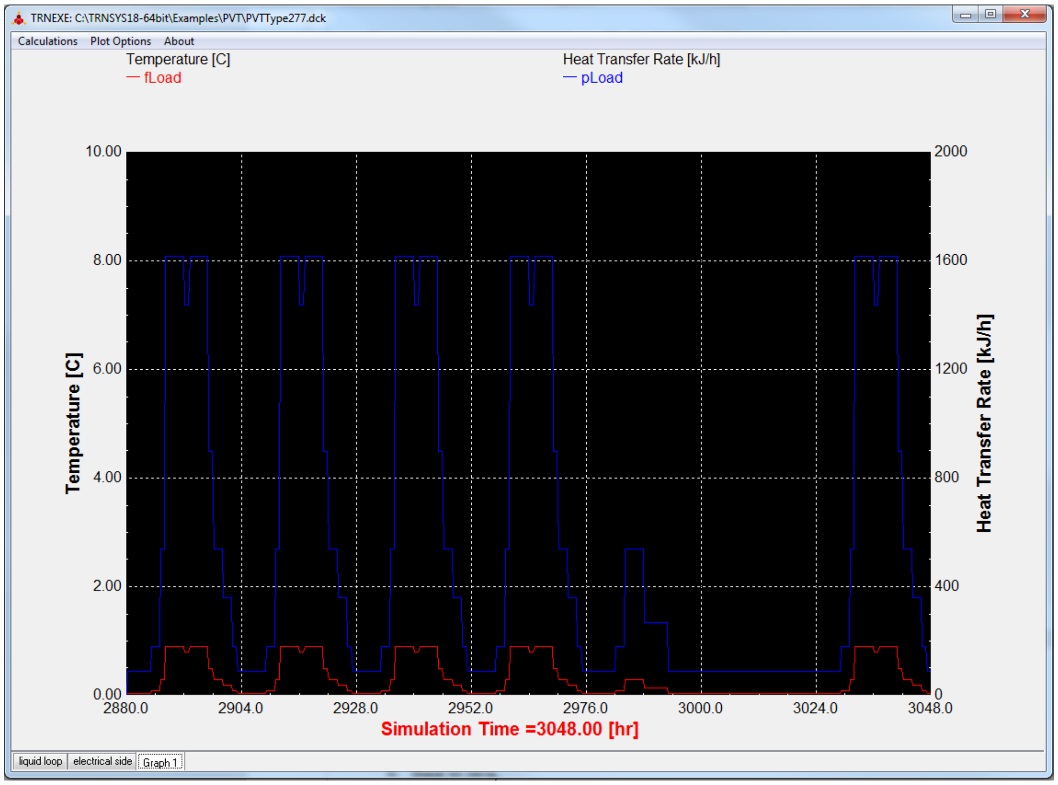


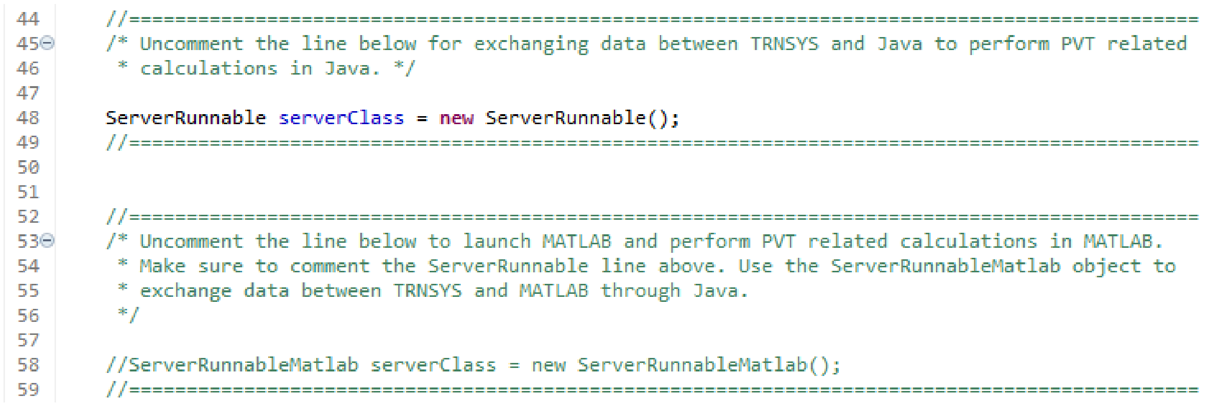
Figure 9. Result of simulating the PVTType277 example with the local server in Java

4. Accessing Data and Modifying the Code

The objective of this section is to describe the process for accessing data, modifying the code, and re-building the Java project. There are two main classes `ServerRunnable` and `ServerRunnableMatlab`, which provide access to the data and the ability to modify the code. In the `ServerRunnable` class, a user can access the data from TRNSYS and modify the code to meet their needs in the Java environment. Note that `ServerRunnableMatlab` is a modified version of the `ServerRunnable` class with additional functionality of connecting with a third-party software tool to perform advanced computation. To achieve the objective of this section, the Java project from Section 3.1 was imported into the Eclipse integrated development environment (IDE). Note that using Eclipse is not required to edit or view the content of the Java project. Users can choose a different integrated environment or none. The instructions for importing the Java project and adding third-party libraries into Eclipse are documented in Appendix.

4.1. ServerRunnable Class

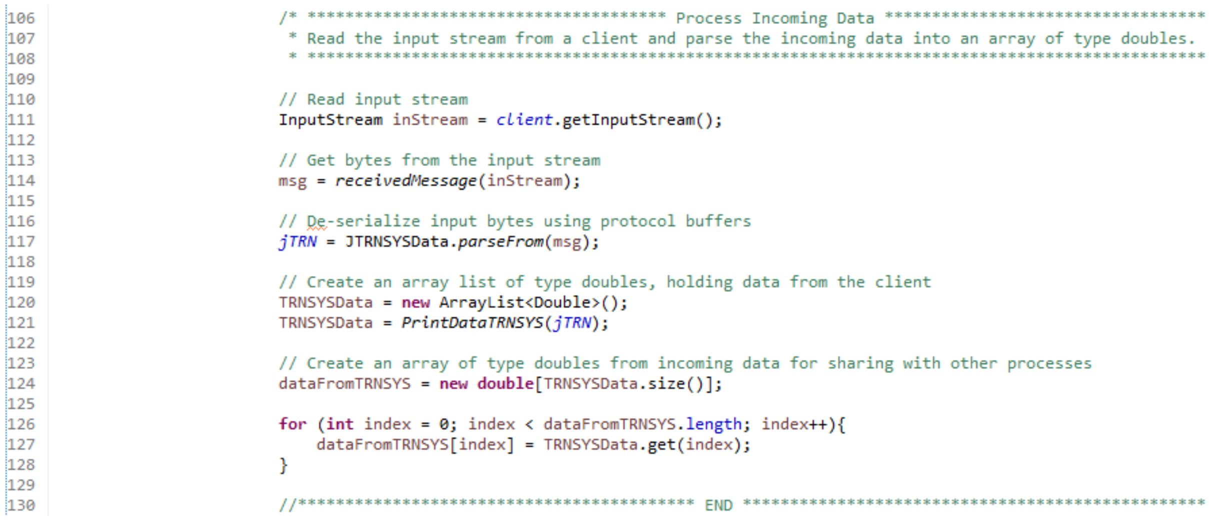
As previously mentioned, the `ServerUI` class is responsible for launching the GUI, which creates an object of the `ServerController` class. The `ServerController` class creates an object of the `ServerRunnable` class as shown in the code snippet of Figure 10. These JAVA classes are in the JTRNSY project repository (JTRNSY\src) downloaded in Section 3.1.



```java
//==========================================================================================
/* Uncomment the line below for exchanging data between TRNSYS and Java to perform PVT related
 * calculations in Java. */

ServerRunnable serverClass = new ServerRunnable();
//==========================================================================================


//==========================================================================================
/* Uncomment the line below to launch MATLAB and perform PVT related calculations in MATLAB.
 * Make sure to comment the ServerRunnable line above. Use the ServerRunnableMatlab object to
 * exchange data between TRNSYS and MATLAB through Java.
 */

//ServerRunnableMatlab serverClass = new ServerRunnableMatlab();
//==========================================================================================
```

Figure 10. Create an object of ServerRunnable class for launching the server (see `ServerController`)

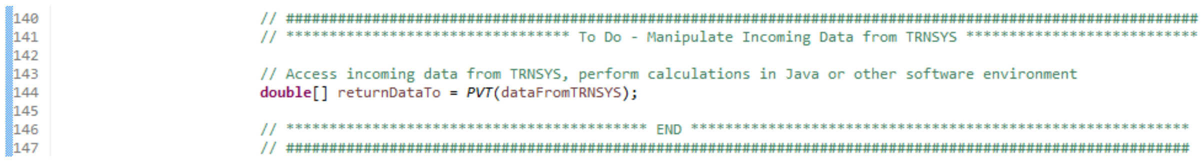
`ServerRunnable` is the main class for sending and receiving data to-and-from TRNSYS. It has three key functionalities: processing incoming data from TRNSYS, performing manipulation of incoming data, and sending the results back to TRNSYS. In the code snippet shown in Figure 11, the incoming data from TRNSYS (client) is read, deserialized, and parsed into a list and an array of the primitive type Double. This code is used to process incoming data and should function properly without modification.



```
/* ************************************** Process Incoming Data ***********************************
 * Read the input stream from a client and parse the incoming data into an array of type doubles.
 * ************************************************************************************************

// Read input stream
InputStream inStream = client.getInputStream();

// Get bytes from the input stream
msg = receivedMessage(inStream);

// De-serialize input bytes using protocol buffers
jTRN = JTRNSYSData.parseFrom(msg);

// Create an array list of type doubles, holding data from the client
TRNSYSData = new ArrayList<Double>();
TRNSYSData = PrintDataTRNSYS(jTRN);

// Create an array of type doubles from incoming data for sharing with other processes
dataFromTRNSYS = new double[TRNSYSData.size()];

for (int index = 0; index < dataFromTRNSYS.length; index++){
    dataFromTRNSYS[index] = TRNSYSData.get(index);
}

//****************************************** END *************************************************
```

Figure 11. Code snippet for processing incoming data from TRNSYS (see `ServerRunnable.java`)

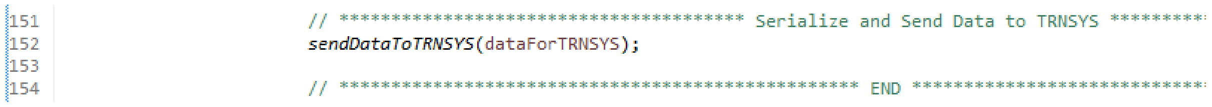
The array of the primitive type Double, `dataFromTRNSYS`, contains the data from TRNSYS. The number of entries in this array depends on the number of inputs connected to Type277as shown in Figure 3. In the modified PVT example fLoad is the only input to Type277; therefore, `dataFromTRNSYS` contains a single entry. Recall that fLoad is used to calculate pLoad. The `dataFromTRNSYS` is passed in as an input argument to the `PVT()` method (Figure 12). The `PVT()` method (see `ServerRunnable.java`) returns an array of the primitive type Double, `returnDataTo`, which contains a single entry which is the calculated value of pLoad for a given value of fLoad. Use the "To Do" space shown in Figure 12 to access and perform manipulation of the data from TRNSYS.



```
// ##############################################################################################
// ****************************** To Do - Manipulate Incoming Data from TRNSYS ******************

// Access incoming data from TRNSYS, perform calculations in Java or other software environment
double[] returnDataTo = PVT(dataFromTRNSYS);

// ***************************************** END ************************************************
// ##############################################################################################
```

Figure 12. Access data from TRNSYS (see `ServerRunnable.java`)

The output of the `PVT()` method, `returnDataTo,` is passed as an input argument to the `sendDataToTRNSYS()` method (Figure 13). The `sendDataToTRNSYS()` is responsible for serializing and sending the data back to TRNSYS over the socket communication.



```
// *************************************** Serialize and Send Data to TRNSYS *********
sendDataToTRNSYS(dataForTRNSYS);

// *************************************************** END ***************************
```

Figure 13. Sending serialized data back to TRNSYS (see `ServerRunnable.java`)

An important feature of this client-server implementation is its ability to halt the simulation for debugging purposes. The TRNSYS simulation will not advance to the next timestep until it receives data from the server. Before using this functionality, the connection between the server and client must be established. In other words, before simulating the TRNSYS model, the server must be running. If a model is simulated without an active server, TRNSYS will not throw an error or display a warning. Instead, the simulation will progress slowly. To suspend the simulation, place a breakpoint in the `ServerRunnable` class and run the `ServerUI` class "JTRNSYS\src\application\" in debug mode.

4.2. ServerRunnableMatlab Class

In the previous section, the process for calculating pLoad in the Java environment was described. In this section, pLoad is calculated using MATLAB. Although TRNSYS has a MATLAB type, this section is used to demonstrate the functionality of the server communicating with a third-party software. The UML class diagram in Figure 14 shows the `ServerUI`, `ServerController`, and `ServerRunnableMatlab` that are included in JTRNSYS\src repository downloaded in Section 3.1. It describes the relationship of various classes used to support the loosely-coupled integration of TRNSYS and MATLAB via the server implemented in Java.

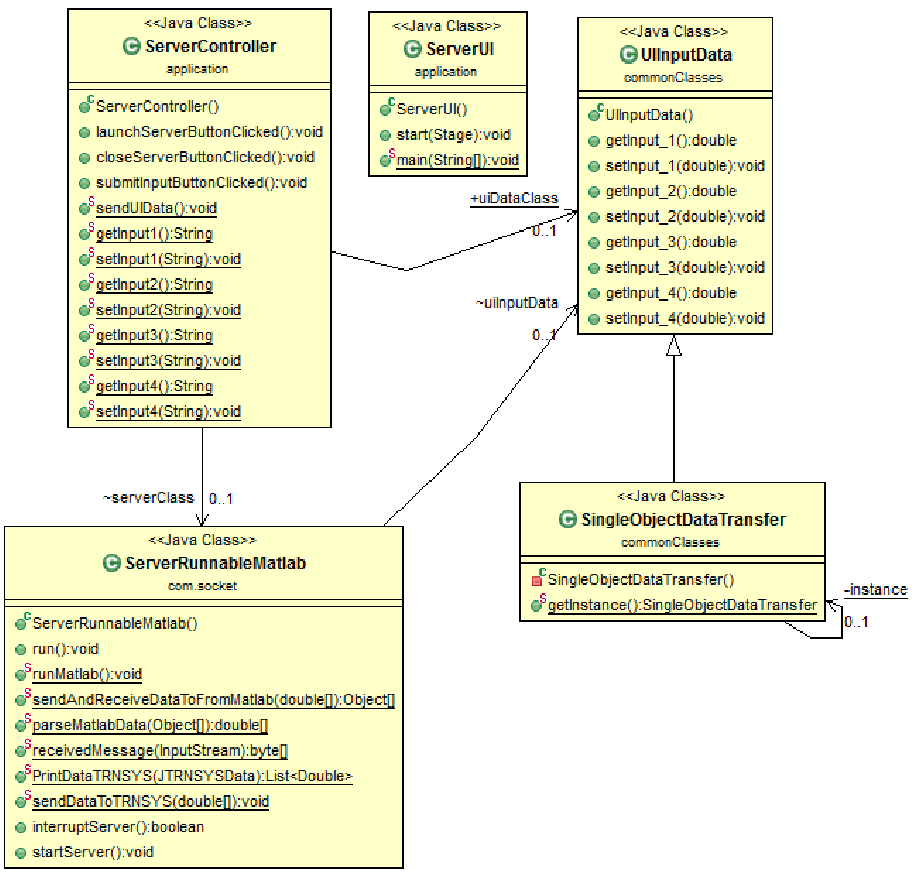
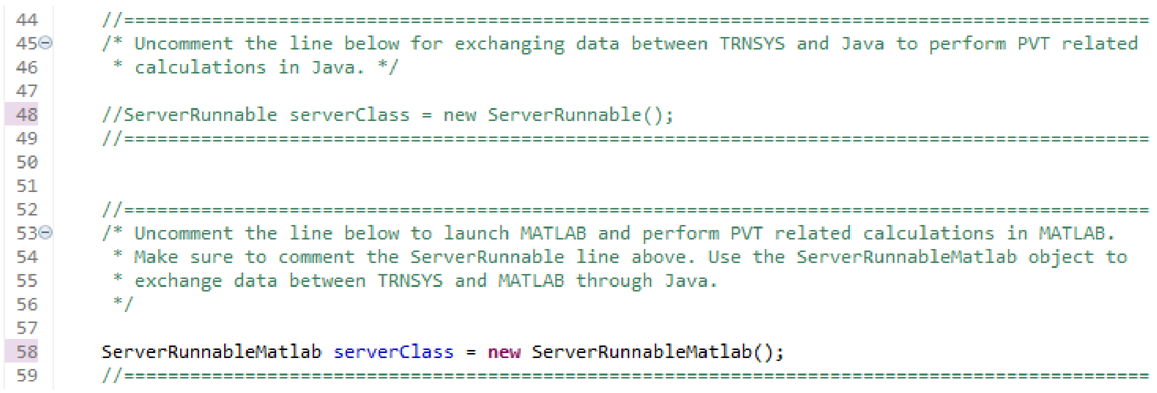


Figure 14. UML class diagram for loosely-coupling TRNSYS and MATLAB via the Java environment

Like before, `ServerUI` is the main class for rendering the GUI and creating an instance of the `ServerController`. The `ServerController` class controls the functionality of the GUI and instantiates an object of the `ServerRunnableMatlab` class as shown in the code snippet of Figure 15. This change is accomplished by commenting out the `ServerRunnable` object on line 48 of Figure 15 and uncommenting the `ServerRunnableMatlab` object on line 58. Once the GUI is rendered (Figure 5), double clicking the Start Server button will launch the server and MATLAB.



```
44      //==========================================================================================
45      /* Uncomment the line below for exchanging data between TRNSYS and Java to perform PVT related
46       * calculations in Java. */
47
48      //ServerRunnable serverClass = new ServerRunnable();
49      //==========================================================================================
50
51
52      //==========================================================================================
53      /* Uncomment the line below to launch MATLAB and perform PVT related calculations in MATLAB.
54       * Make sure to comment the ServerRunnable line above. Use the ServerRunnableMatlab object to
55       * exchange data between TRNSYS and MATLAB through Java.
56       */
57
58      ServerRunnableMatlab serverClass = new ServerRunnableMatlab();
59      //==========================================================================================
```

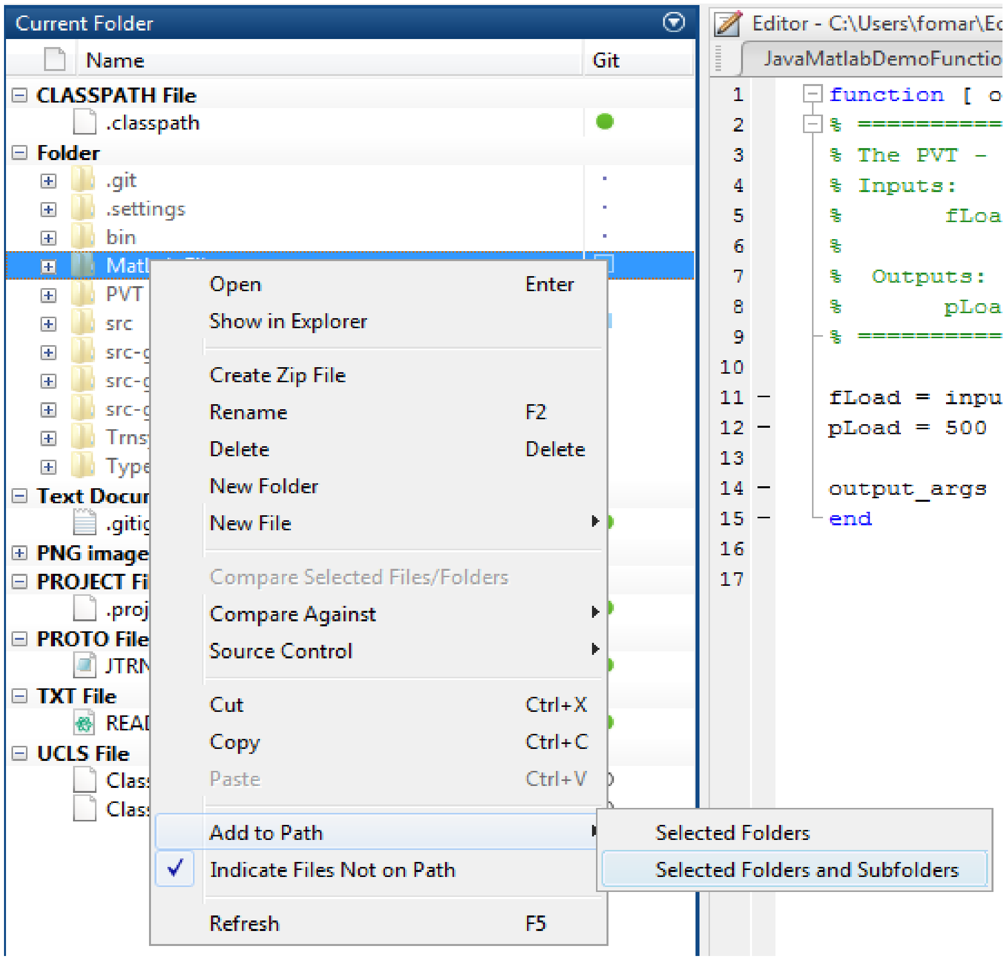
Figure 15. Create an object ServerRunnableMATLAB for exchanging data between TRNSYS and MATLAB

`ServerRunnableMatlab` is the main class for launching the server and facilitating the exchange of data between TRNSYS and MATLAB. It has three key functionalities: processing incoming data from TRNSYS, sending and receiving data to-and-from MATLAB to perform calculation, and finally sending the results back to TRNSYS. The `ServerController` class also instantiates an object of `UIInputData` class to capture the user inputs from the GUI and transfer them to other classes using a Singleton object.

In `ServerRunnableMaltab` class, processing the incoming data from TRNSYS is like the procedure described in Section 4.1 and shown in Figure 11. In the next section, the procedure for calculating pLoad in MATLAB is described.

4.2.1. Performing Calculations in MATLAB

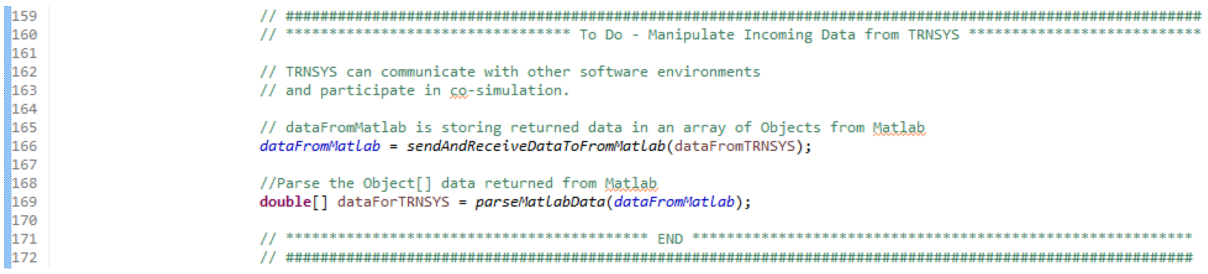
After the server is launched and the MATLAB program is opened, right-click on “Matlab Files” folder and add it to the path as shown in Figure 16. There are two functions in this directory, the `JavaMatlabDemoFunction()`, and the `PVTType277()`. The `JavaMatlabDemoFunction()` is the entry point to the MATLAB environment. The data from Java is communicated to MATLAB through the input argument of this function. The `PVTType277()` function in MATLAB computes pLoad for the given values of fLoad from TRNSYS. Complex calculations using MATLAB’s analytical capabilities are performed in a similar manner. A case study for assessing the performance of six residential energy management control algorithms through co-simulation of TRNSYS and MATLAB facilitated by Type277 is presented in [2], [4].



```
function [ o
% ==========
% The PVT -
% Inputs:
%         fLoa
%
%  Outputs:
%         pLoa
% ==========

fLoad = inpu
pLoad = 500

output_args
end
```

Figure 16. Adding Matlab Files folder to the path of MATLAB software

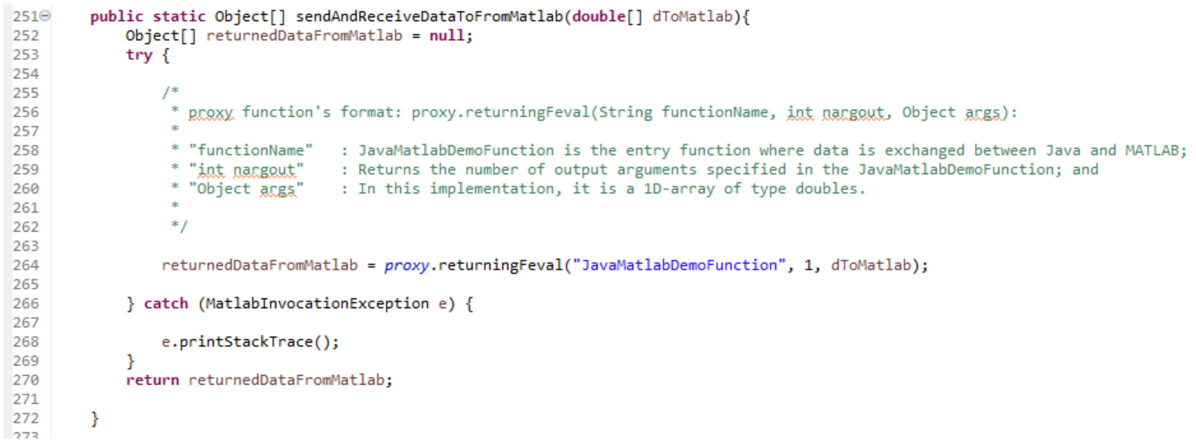
Recall that our intended task is to calculate pLoad in MATLAB; therefore, we need to send fLoad from TRNSYS to MATLAB to perform this calculation. Figure 17 shows the two methods that facilitate the indirect exchange of data between TRNSYS and MATLAB.



```
// #####################################################################################################
// ********************************* To Do - Manipulate Incoming Data from TRNSYS ***************************

// TRNSYS can communicate with other software environments
// and participate in co-simulation.

// dataFromMatlab is storing returned data in an array of Objects from Matlab
dataFromMatlab = sendAndReceiveDataToFromMatlab(dataFromTRNSYS);

//Parse the Object[] data returned from Matlab
double[] dataForTRNSYS = parseMatlabData(dataFromMatlab);

// ****************************************** END **************************************************************
// ####################################################################################################
```

Figure 17. Methods for facilitating the exchange of data between TRNSYS and MATLAB

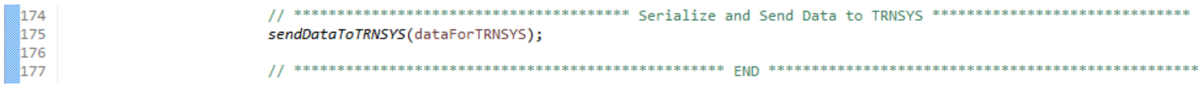
As can be seen from Figure 17, the `datafromTRNSYS` is passed as an argument to the `sendAndReceiveDataToFromMatlab()` method, which facilitates the exchange of data between Java and MATLAB through a proxy established by using the matlabcontrol library [5]. As shown in Figure 18, the proxy calls the `JavaMatlabDemoFunction()` and passes on the incoming data from TRNSYS. The output of this proxy function evaluation contains the returned values from MATLAB.



```java
    public static Object[] sendAndReceiveDataToFromMatlab(double[] dToMatlab){
        Object[] returnedDataFromMatlab = null;
        try {

            /*
             * proxy function's format: proxy.returningFeval(String functionName, int nargout, Object args):
             *
             * "functionName"   : JavaMatlabDemoFunction is the entry function where data is exchanged between Java and MATLAB;
             * "int nargout"    : Returns the number of output arguments specified in the JavaMatlabDemoFunction; and
             * "Object args"    : In this implementation, it is a 1D-array of type doubles.
             *
             */

            returnedDataFromMatlab = proxy.returningFeval("JavaMatlabDemoFunction", 1, dToMatlab);

        } catch (MatlabInvocationException e) {

            e.printStackTrace();
        }
        return returnedDataFromMatlab;

    }
```

Figure 18. Proxy function call to send and receive data to-and-from MATLAB

The output of `sendAndReceiveDataToFromMatlab()` is an array of objects, which is the input argument to the `parseMatlabData()` method (Figure 17). The `parseMatlabData()` method extracts the result of calculations from the array of objects and returns an array of the primitive type Double, `dataForTRNSYS`. The `sendDataToTRNSYS()` method shown in Figure 19 takes `dataForTRNSYS` as an argument. It serializes the data and sends it to TRNSYS using socket communication.

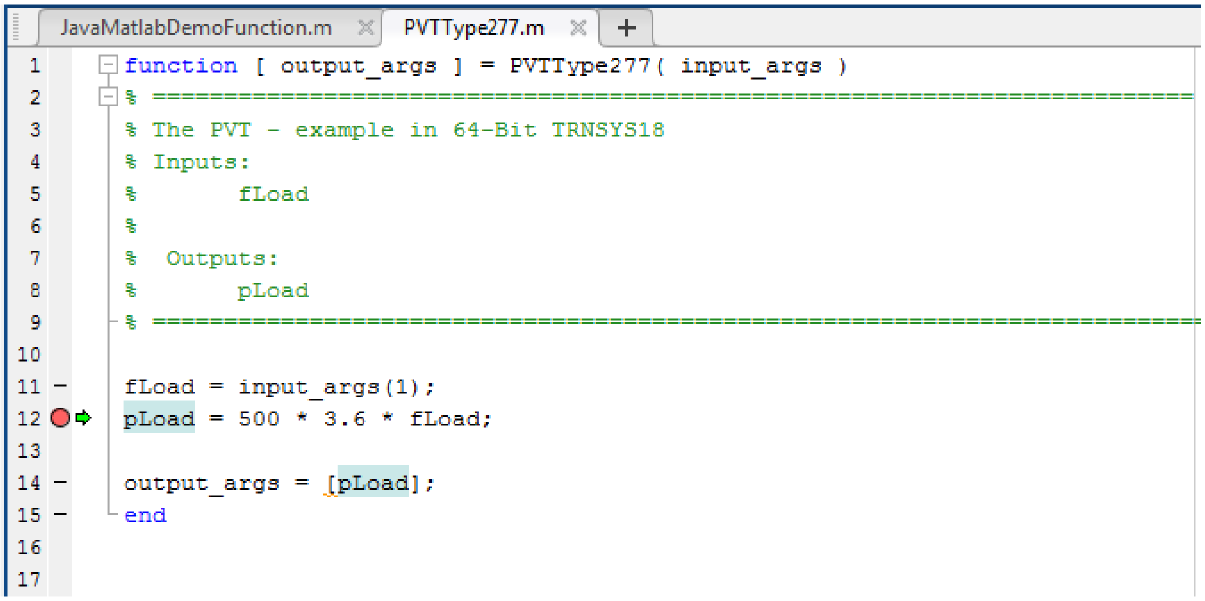


```java
                    // ************************************** Serialize and Send Data to TRNSYS ******************************
                    sendDataToTRNSYS(dataForTRNSYS);

                    // ************************************************** END **************************************************
```

Figure 19. Method to serialize and return data to TRNSYS

As previously mentioned, an important feature of this client-server implementation is its ability to halt the simulation for debugging purposes. Before using this functionality, the connection between the server, client, and MATLAB must be established. In other words, before simulating the TRNSYS model, the server and MATLAB must be running. The path to the MATLAB functions must also be configured as described in Figure 16. To suspend the simulation, place a breakpoint in MATLAB. To pause the simulation in MATLAB, the `ServerUI` class "JTRNSYS\src\application\" does not need to be executed in debug mode. If a model is simulated with both the server and MATLAB running but the path to MATLAB functions are not configured as shown in Figure 16, Java will throw MatlabInvocationException, TRNSYS will display an error message, and MATLAB will display "Undefined function or variable JavaMatlabDemoFunction". The simulation will not run, and the user will need to relaunch the GUI and the server by double clicking the Start Server button.

Figure 20 shows the `PVTType277`() function in MATLAB with a breakpoint which halted the TRNSYS simulation from stepping forward.



```
function [ output_args ] = PVTType277( input_args )
% ==========================================================================
% The PVT - example in 64-Bit TRNSYS18
% Inputs:
%       fLoad
%
%  Outputs:
%       pLoad
% ==========================================================================

fLoad = input_args(1);
pLoad = 500 * 3.6 * fLoad;

output_args = [pLoad];
end
```

Figure 20. Debugging in MATLAB

5. Conclusion

An application of Type277 was successfully demonstrated for facilitating the loosely-coupled integration of a TRNSYS model with a Java environment and MATLAB. Type277 provides a simple and efficient mechanism for running a TRNSYS model in a co-simulation environment without changing the core functionality of the server for exchanging data. Type277 is written in C++ and compiled into 32-bit and 64-bit DLLs, supporting both versions of TRNSYS. It supports co-simulation of TRNSYS with multiple programming languages such as Java, C++, and Python.

Appendix – Importing JTRNSYS Project into Eclipse

To import the Java project (jtrnsys-master) from Step 2 of Section 3.1 into an existing workspace or create a new workspace in Eclipse follow these instructions:

1. In Eclipse, choose *File* and select Import… from the drop-down menu. The Import wizard opens as shown in Figure 21;

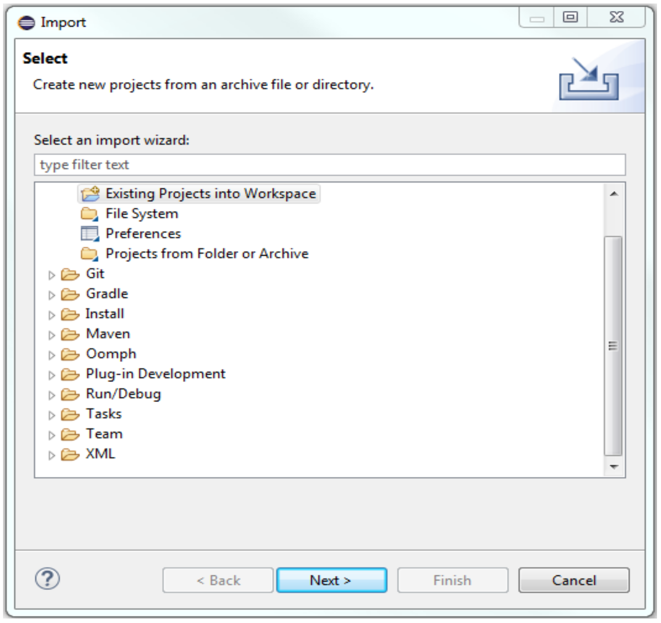
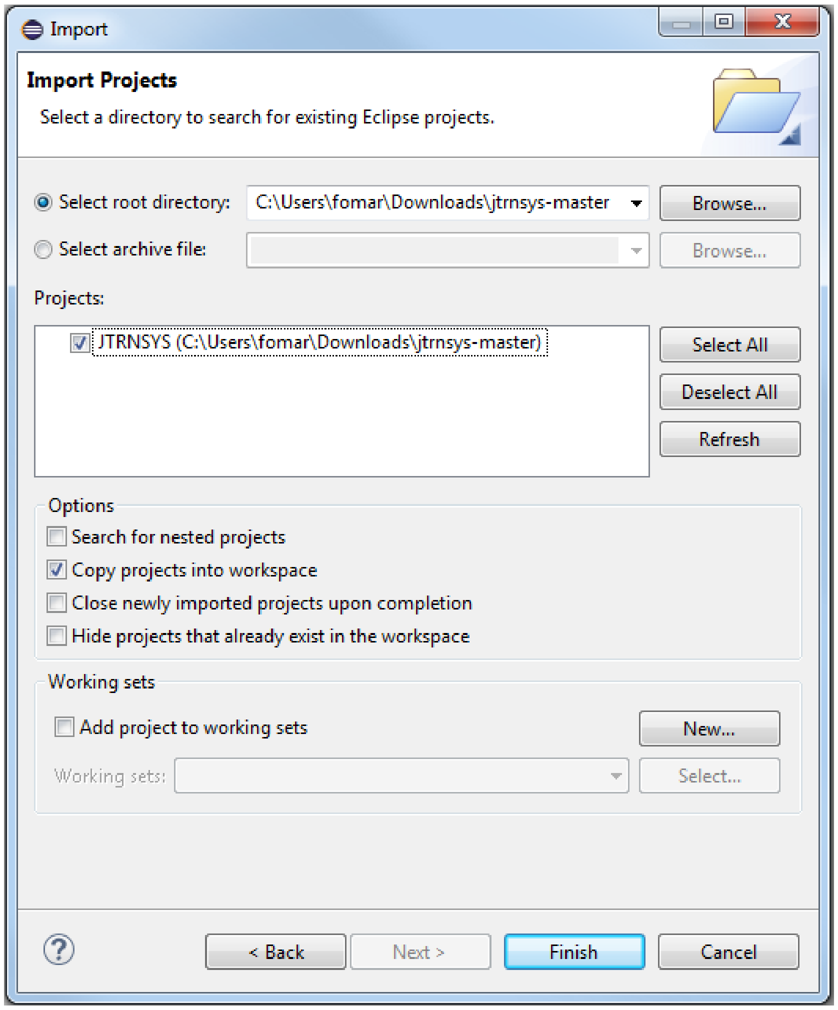


Figure 21. Eclipse import wizard

2. In the Import wizard, choose *Existing Projects into Workspace* and click Next;
3. In the Import Projects window, using the Browse button navigate to the directory where you extracted the zip folder jtrnsys-master;
4. Select the checkbox next to the Copy projects into workspace to make a copy of the project into your workspace as shown Figure 22;



Import Projects

Select a directory to search for existing Eclipse projects.

[✓ Select root directory:] C:\Users\fomar\Downloads\jtrnsys-master

Select archive file:

Projects:

☑ JTRNSYS (C:\Users\fomar\Downloads\jtrnsys-master)

Options

☐ Search for nested projects

☑ Copy projects into workspace

☐ Close newly imported projects upon completion

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

Figure 22. Import projects wizard

5. Click Finish, this process will import JTRNSYS Java project into the workspace as shown in Figure 23.

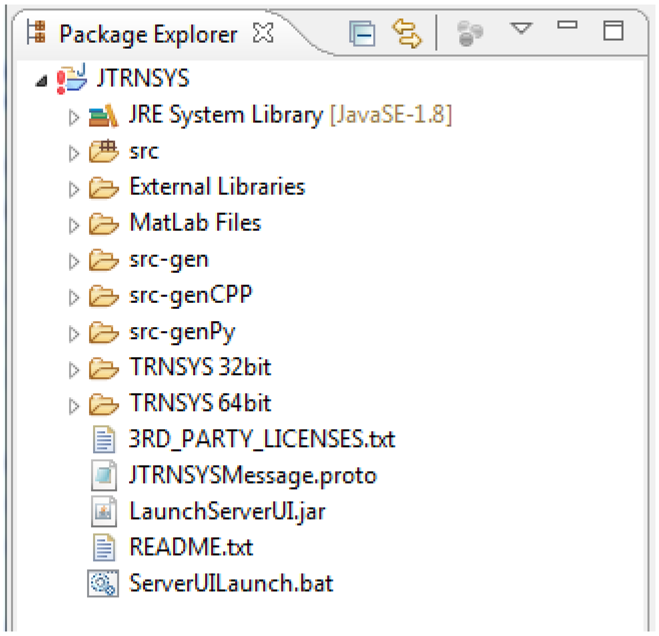


Figure 23. Project explorer in Eclipse

As can be seen from Figure 23, the Package Explorer (red exclamation mark) indicates that there are errors in the project. The errors are because of missing required protocol buffers, matlabcontrol, and JavaFX11 libraries. For convenience, a copy of these libraries is included in the JTRNSYS project under "jtrnsys-master\External Libraries" folder, and a copy of the licenses for these libraries are reproduced in the 3RD_PARTY_LICENSES.txt file.

The following steps describe the process of adding these libraries to the JTRNSY project:

a. Right click on the JTRNSYS project and select Properties from the list as shown in Figure 24;

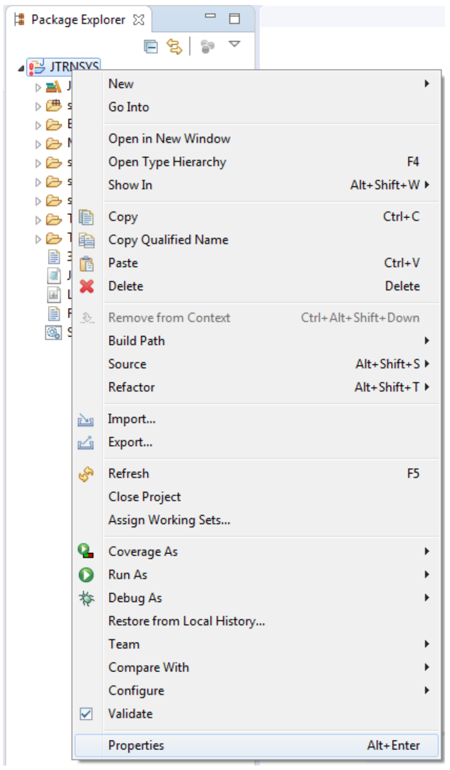
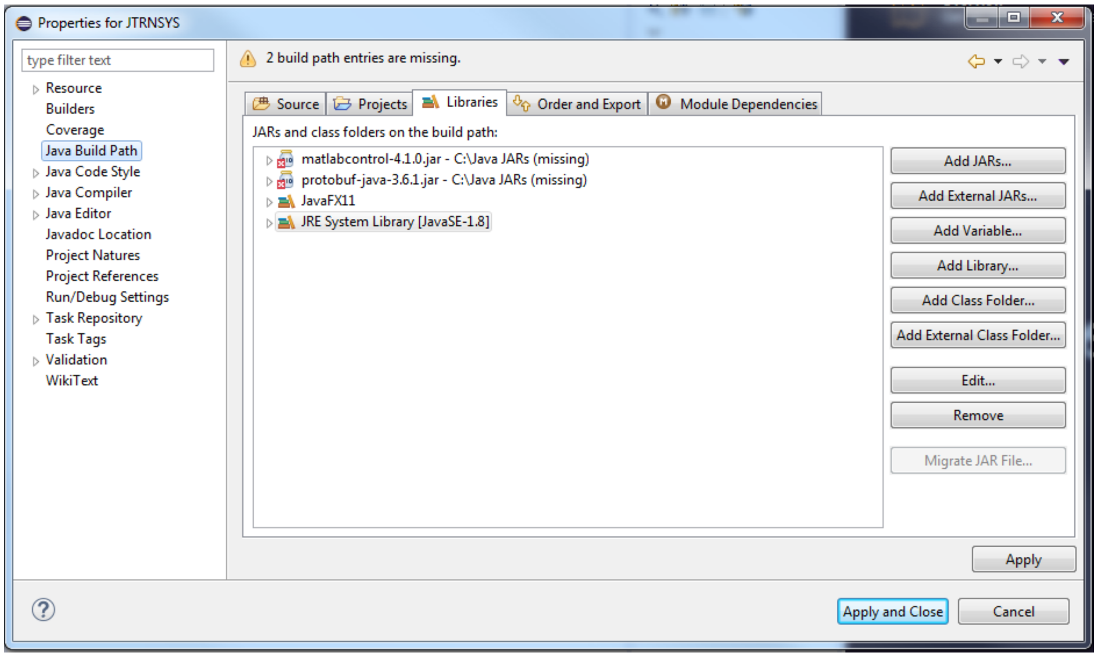


Figure 24. Opening properties window for JTRNSYS

b. In the Properties for JTRNSYS window, select Java Build Path and select the Libraries tab as shown in Figure 25. We first need to remove the references to these missing libraries and then add them from "jtrnsys-master\External Libraries" folder downloaded in Section 3.1;


Properties for JTRNSYS

2 build path entries are missing.

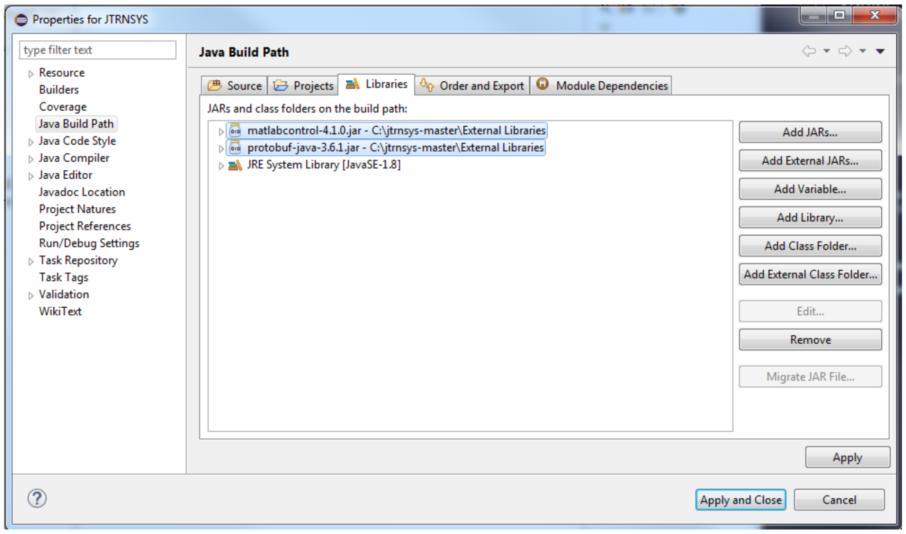
JARs and class folders on the build path:

- matlabcontrol-4.1.0.jar - C:\Java JARs (missing)
- protobuf-java-3.6.1.jar - C:\Java JARs (missing)
- JavaFX11
- JRE System Library [JavaSE-1.8]

Figure 25. Editing Java build path for the missing libraries

c. To remove the missing references to `matlabcontrol-4.1.0.jar`, `protobuf-java-3.6.1.jar`, as well as `JavaFX11` libraries, select each library and click the Remove button as shown in Figure 25;

d. To add the missing libraries, click the Add External JARs button (Figure 25) and navigate to the "jtrnsys-master\External Libraries" folder and select both `matlabcontrol-4.1.0.jar` and `protobuf-java-3.6.1.jar` files and click Open as shown in Figure 26. This step adds these two libraries on the build path for JTRNSYS project;


Properties for JTRNSYS

Java Build Path

JARs and class folders on the build path:

- matlabcontrol-4.1.0.jar - C:\jtrnsys-master\External Libraries
- protobuf-java-3.6.1.jar - C:\jtrnsys-master\External Libraries
- JRE System Library [JavaSE-1.8]

Figure 26. Adding third-party libraries to the build path for JTRNSYS project

To add JavaFX 11[1] library, click the Add Library button as shown in Figure 25.

e. In the Add Library window (Figure 27) select the User Library type and click Next;

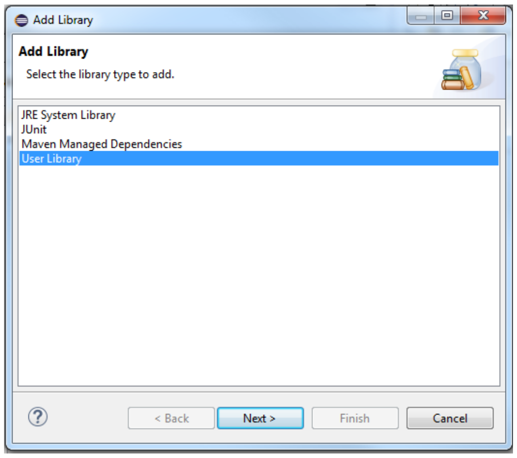
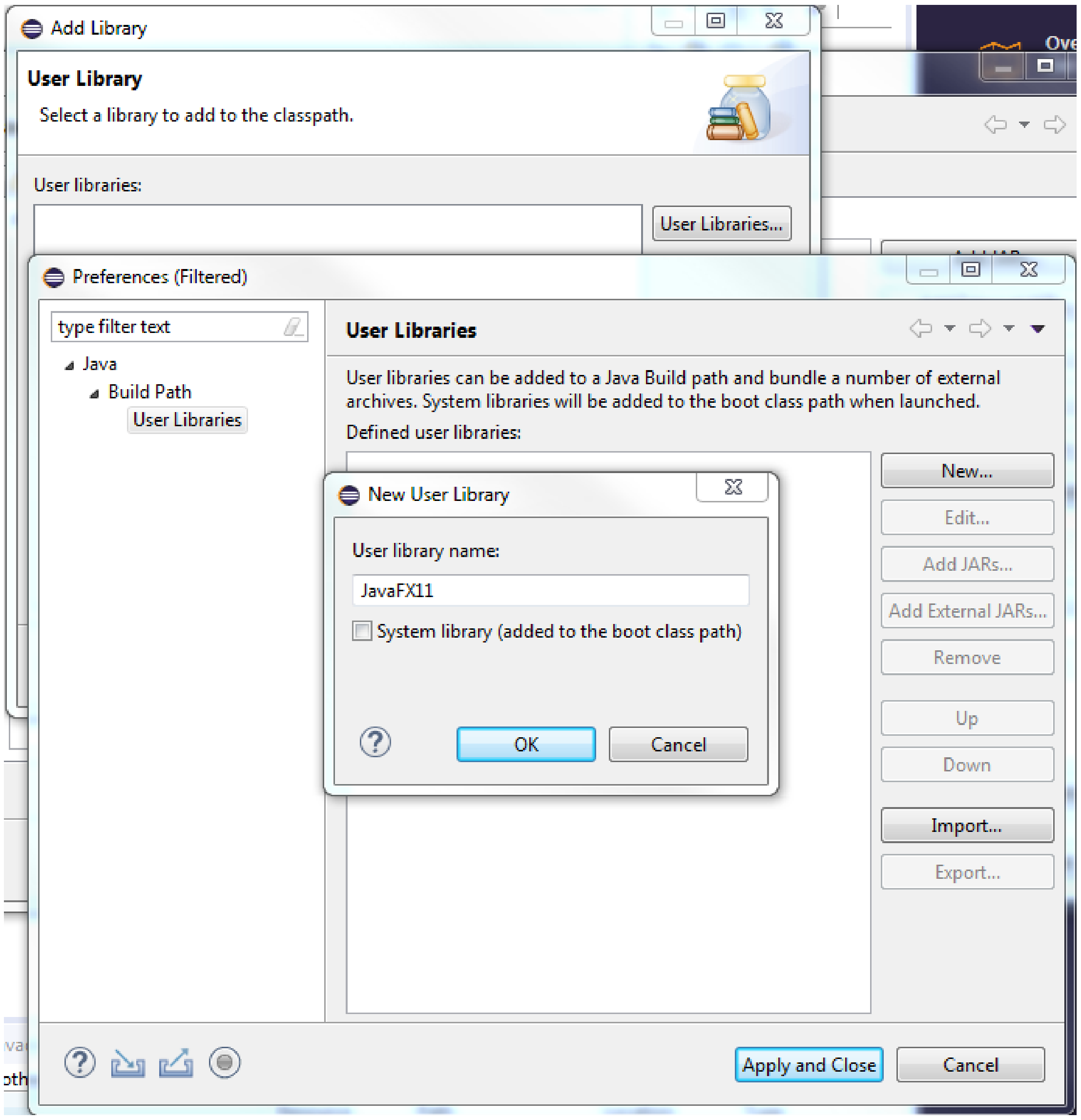


Figure 27. Adding a user library to the JTRNSYS project

f. In the Add Library window shown in Figure 28, click the User Libraries button to open the Preferences (Filtered) window, then click the New button to open the New User Library window. In the New User Library window, create a name for the new library like JavaFX11 or any other desired name, and click OK;

[1] The instructions for adding JavaFX 11 to the Eclipse IDE was obtained from https://openjfx.io/openjfx-docs/


Add Library

User Library

Select a library to add to the classpath.

User libraries:

User Libraries...

Preferences (Filtered)

type filter text

- Java
  - Build Path
    - User Libraries

User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

New... | Edit... | Add JARs... | Add External JARs... | Remove | Up | Down | Import... | Export...

New User Library

User library name: JavaFX11

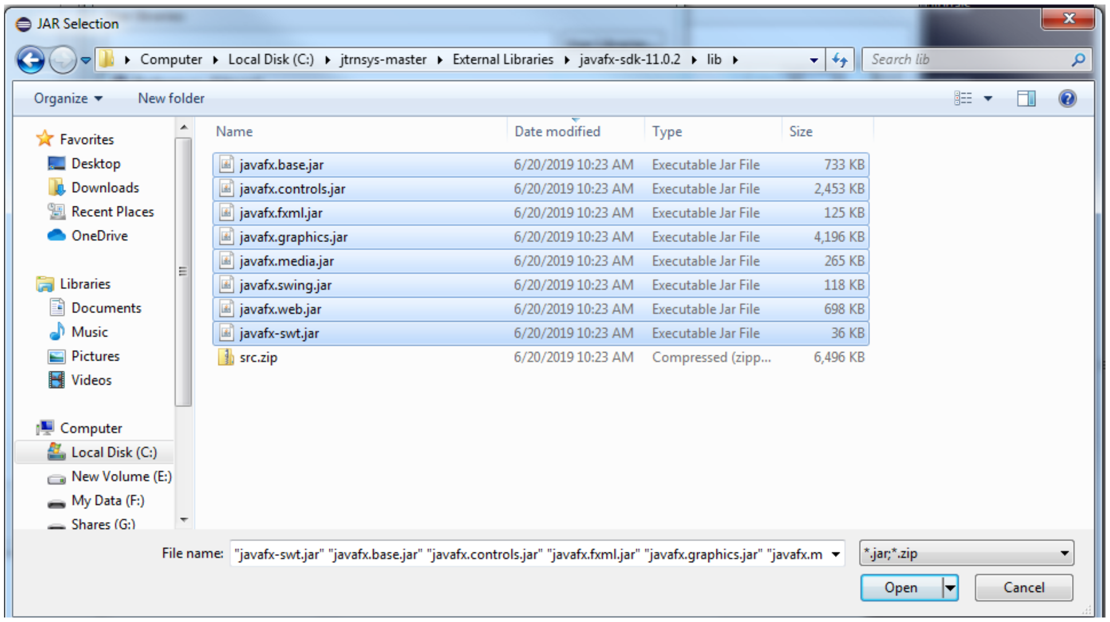
[ ] System library (added to the boot class path)

OK | Cancel

Apply and Close | Cancel

Figure 28. Adding a new JavaFX 11 user library to the JTRNSYS project

g. In Preferences (Filtered) window, click the Add External JARs (Figure 28) button and navigate to “C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib”. Select all jar files as shown in Figure 29 and click Open;



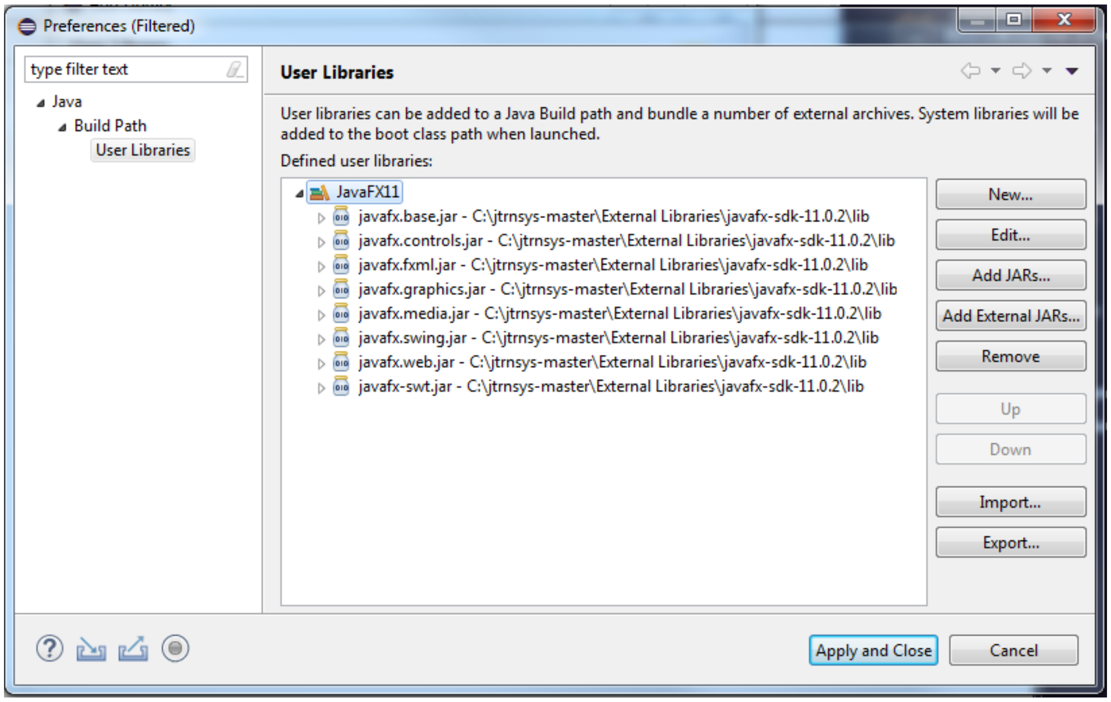
Name	Date modified	Type	Size
javafx.base.jar	6/20/2019 10:23 AM	Executable Jar File	733 KB
javafx.controls.jar	6/20/2019 10:23 AM	Executable Jar File	2,453 KB
javafx.fxml.jar	6/20/2019 10:23 AM	Executable Jar File	125 KB
javafx.graphics.jar	6/20/2019 10:23 AM	Executable Jar File	4,196 KB
javafx.media.jar	6/20/2019 10:23 AM	Executable Jar File	265 KB
javafx.swing.jar	6/20/2019 10:23 AM	Executable Jar File	118 KB
javafx.web.jar	6/20/2019 10:23 AM	Executable Jar File	698 KB
javafx-swt.jar	6/20/2019 10:23 AM	Executable Jar File	36 KB
src.zip	6/20/2019 10:23 AM	Compressed (zipp...	6,496 KB

File name: "javafx-swt.jar" "javafx.base.jar" "javafx.controls.jar" "javafx.fxml.jar" "javafx.graphics.jar" "javafx.m

.jar;.zip

Figure 29. Selecting JavaFX 11 jar files

The procedure in Step 7 creates a JavaFX 11 user library as shown in Figure 30 that can be added to the Java build path for JTRNSYS project.



User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

JavaFX11
- javafx.base.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib
- javafx.controls.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib
- javafx.fxml.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib
- javafx.graphics.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib
- javafx.media.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib
- javafx.swing.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib
- javafx.web.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib
- javafx-swt.jar - C:\jtrnsys-master\External Libraries\javafx-sdk-11.0.2\lib

Figure 30. JavaFX 11 user library

h. Click Apply and Close button to close the Preferences (Filtered) window, and then click the Finish button to close the Add Library window (Figure 31);

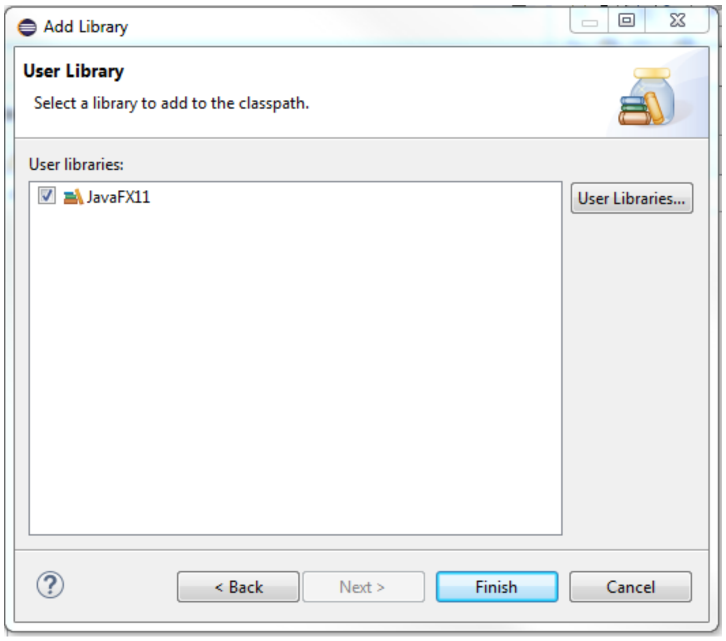


Figure 31. Close the Add Library window

As shown in Figure 32, the JavaFX11 user library has been added to the build path for JTRNSYS project. Click the Apply and Close button to exit the Properties for JTRNSYS window.

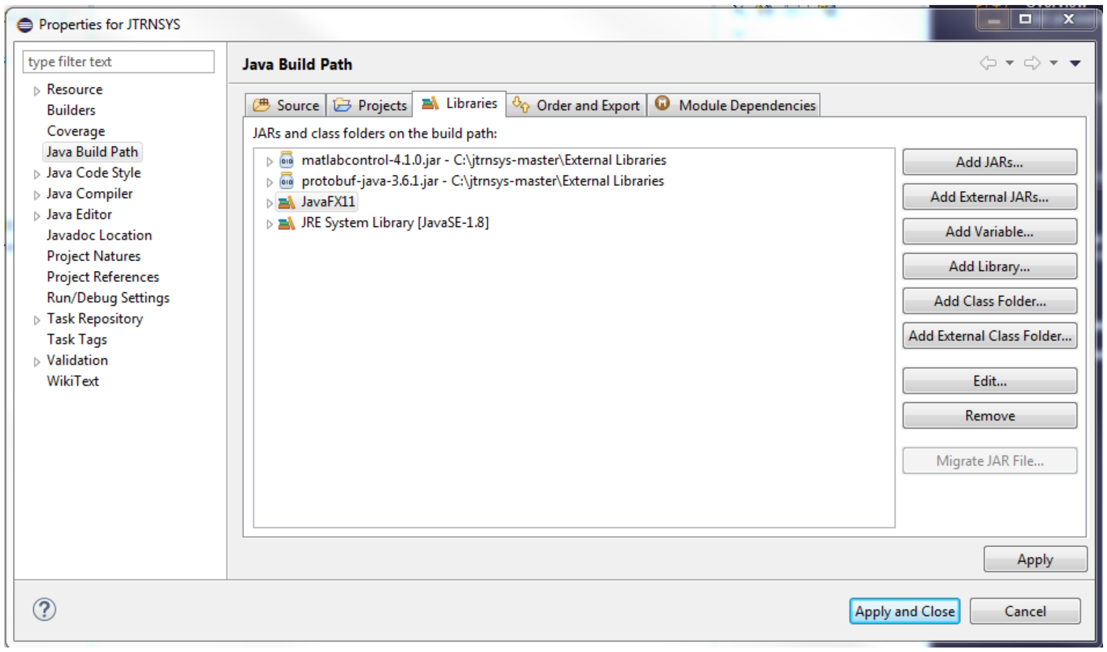


Figure 32. Added external libraries to the build path of the JTRNSYS project

The instructions in Step a through h eliminated the errors caused by missing the required external libraries. To launch the Server GUI from Eclipse (as shown in Figure 5), navigate to the "JTRNSYS/src/application" and run the `ServerUI.java` class as a Java Application. After the Server GUI is activated, follow the instructions in Section 3.2 to launch the server and start communicating with a simulation model in TRNSYS.

To access the data and modify the code, navigate to "JTRNSYS/src/com.socket" and open `ServerRunnable.java` or `ServerRunnableMatlab.java` class and follow the instructions provided in Section 4.

References

[1] S. A. Klein et al, “TRNSYS 17: A Transient System Simulation program, Solar Energy Laboratory, University of Wisconsin, Madison, USA,” *Trnsys*, 2010.

[2] F. Omar, “A Residential Energy Control Algorithm Assessment Tool for Smart Grid : Multi-Criteria Decision Making Using the Analytical Hierarchy Process,” *Ph.D. Dissertation*, 2019. [Online]. Available: https://doi.org/10.18130/v3-d6fh-nj31.

[3] Google, “Protocol Buffers | Google Developers,” 2008. [Online]. Available: https://developers.google.com/protocol-buffers/. [Accessed: 10-Sep-2014].

[4] F. Omar, S. T. Bushby, and R. D. Williams, “Assessing the Performance of Residential Energy Management Control Algorithms: Muti-Criteria Decision Making Using the Analytical Hierarchy Process,” *NIST TN 2017*, 2018.

[5] Google product, “Google Code Archive - Long-term storage for Google Code Project Hosting.,” 2013. [Online]. Available: https://code.google.com/archive/p/matlabcontrol/. [Accessed: 17-Sep-2014].