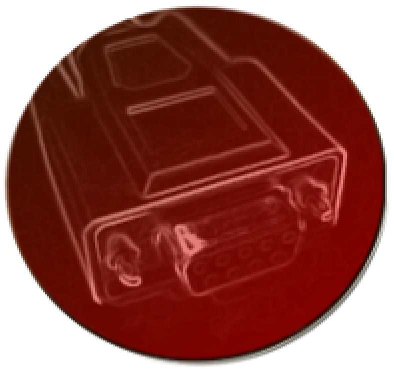

Serial Port Plug-in

Developer's Manual

Last Updated: 6Jun2012

Table of Contents

Serial Port Plug-in

General

Developer System Requirements

- **Windows, Linux**, or **Mac OS X**
- **Servoy (3.x or higher)**
- **Physical or Virtual Serial Port Device**
- **Java**

Installation

To install the plug-in, run the plug-in installer jar file. Follow the instructions carefully, as you'll need to choose your version of Servoy during the installation. Choose to install into the directory you've installed Servoy into.

Once you have the plug-in installed into your Servoy directory, start Servoy Developer.

If a license dialog does not come up, you may need to upgrade Java.

Activation

After you have installed the plug-in and started Servoy Developer, a dialog box will appear prompting you for your license details. If you do not know this information, refer to the e-mail you received when you purchased a license or downloaded a trial.

If you receive an "Authentication Successful" message, you are ready to start using the plug-in. If you are behind a proxy or cannot otherwise connect to the Internet, choose "Manual Activation" instead. This option is not available for trial users.

This process only activates the plug-in for developer usage, you will still need to use run-time activation when deploying (see *Deployment* under *Usage* below).

Removal

To remove the plug-in, first ensure Servoy is not running. Navigate to your Servoy installation folder. For Servoy 4+ users, open the application_server folder. Inside the *plugins* folder, delete the following files and folders:

- SerialPort.jar
- SerialPort.jnlp

- SerialPort *(folder)*

To completely remove all activations and settings, navigate to your home directory (under Windows this is *C:\Documents and Settings\[Your Username]\ or C:\Users\[Your Username]*). Open the *.servoy* folder. This may be hidden on non-Windows systems. Delete the *SerialPort* folder, if it exists.

Usage

Deployment

In order to deploy, your code needs to call the *unlockRuntime(...)* method before any other plug-in methods. To retrieve your runtime code, log into your account at www.prolificaxis.com. Deployment is not available to trial users.

```
plugins.SerialPort.unlockRuntime("YOUR RUNTIME CODE HERE");
```

Serial Port Communication

Connecting to Serial Port

Before attempting to connect to a serial port device, set up an error handler callback (see Callbacks below). This is the preferred way of handling errors when using the plug-in.
To connect to a serial port device, call the openPort() method with the parameters for your device. Keep in mind the device name can change depending on the platform you are using.

```
plugins.SerialPort.openPort("COM1", "9600", "8", "1", "None");
```

If you call *openPort()* without any parameters, the port settings specified by the user inside the Servoy Client's preference panel will be used instead.

Closing Serial Ports

To close a specific port, call the below method with your port name (or blank, to use the preference panel setting):

```
plugins.SerialPort.closePort("COM1")
```

To close all ports being used by the application, use:

```
plugins.SerialPort.closeAllPorts()
```

Reading from a Serial Port

It is recommended that you use a read event callback (setCallbackOnDataAvailable()) when retrieving data from the serial port.

Inside your callback, make a call to the readString() method on the respective port name, like this:

```
plugins.SerialPort.readString("COM1");
```

...and retrieve the result into a string. Since a standard string is limited to what type of data can be stored and read, you can use the alternative readHexString() which will return a human-readable Hex representation of the data retrieved from the port.

You can convert between ASCII and Hex-encoded strings by calling the included encoder methods. However, you may lose data if you encode into Hex after reading with readString(), because of the character limitation mentioned above.

If you find that your data is being received in multiple separate transmissions, it may be a good idea to increase the reader delay using setReaderDelay([port], [time in milleseconds]). The default value is 50.

If your device sends data in a continuous stream instead of in chunks, use setReaderDelay([port], 0) to avoid stalling, and append the data to a string as it comes in, piecing together the data as you need it.

Writing to a Serial Port

You can write a string to the Serial Port by calling writeString([port]) with the provided port name, like so:

```
plugins.SerialPort.writeString("COM1", "Hello World!\n");
```

It is usually a good idea to append a newline at the end of the given string unless your device requires otherwise.

Callbacks

Error Callback

You can have a method handle errors as they occur using the error callback. At the beginning of the program (somewhere after *unlockRuntime()),* pass a name of your handler method by name into the method *setCallbackOnError()*. The method does not need to accept parameters and will only trigger if an error has occurred. The most basic error handler will display a message to the user or print the string result of *errorString()* and then call *clearErrors()*. Here is an example:

```
function error() {
      application.output(plugins.SerialPort.errorString());
      plugins.SerialPort.clearErrors();
}

// somewhere in your initialization code...
plugins.SerialPort.setCallbackOnError(error);
```

Port Event Callbacks

Much like the error callback, port events also have their own callbacks. Each of the callbacks are specific to a port. Pass the port name as the first parameter, and the method to be called as the second. Register these right after you have opened the port.

The port callbacks methods are as follows:

- setCallback**DataAvailable**
- setCallback**CTS** (Clear to send)
- setCallback**DSR** (Data set ready)
- setCallback**BI** (Break interrupt)
- setCallback**CD** (Carrier detect)
- setCallback**FE** (Framing error)
- setCallback**OE** (Overrun error)
- setCallback**PE** (Parity error)
- setCallback**RI** (Ring Indicator)
- setCallback**OBE** (Output Buffer Empty)

Methods

Error Handling

The Serial Port plug-in provides some basic methods for determining if an error has occurred and to retrieve a human readable description of the error. You are encouraged to use the error callbacks during testing since checking for an error after every method execution would otherwise be necessary. Checking after every plug-in call is a good approach if your program changes its behavior as different types of errors occur.

Standard error handling:

- *boolean hasError()*
  Returns whether or not the plug-in had an error.
  Even though errors may trigger a callback, It is recommended to check this after every plug-in method call since the callback will not terminate from the calling function.

- *String errorString()*
  Returns a String detailing the specific error, otherwise returns empty string.
  This is intended for logging purposes.

- *void clearErrors()*
  Clears the error messages. Does not clear Quickbooks errors.

Other Methods

(coming soon)

Technical Details

To increase stability, the plug-in’s serial port functionality is powered by various open-source libraries whose versions are hand-selected and tested for each platform.

Used on Windows and Linux:
RXTX (2.1-7r2)
URL: http://rxtx.qbang.org/

Used on Mac:
PureJavaComm (custom)
URL: http://github.com/flipcoder/purejavacomm

Contact Us

Prolific Axis, LLC
Website: http://www.prolificaxis.com
Support: http://www.prolificaxis.com/support
E-mail: prolificaxis@prolificaxis.com
Alternative E-mail: prolificaxis@gmail.com