



Installation and quickstart



April 2018

Preamble

Each of the four Olsonet software packages mentioned in this note: PICOS, SIDE, VUEE, and PIP comes with documentation. In addition to the documents that you will discover inside the packages, you may also find interesting and relevant various supplementary materials, including technical notes, slide presentations, and miscellaneous papers available from Olsonet's web site:

<http://www.olsonet.com>

This link:

<http://www.olsonet.com/REPO/contents.html>

points to the collection of the requisite software. All the third-party software included there is available (free of charge) from other (official) sites under their respective licenses. Needless to say, all those licenses hold retaining in full the rights of the respective authors.

For some rather obscure historical reasons, SIDE is sometimes called SMURPH (this is how it used to be called in the past). Sometimes we also call it SMURPH/SIDE. Don't let it confuse you: SMURH, SIDE, as well as SMURPH/SIDE refer to the same thing. Here is a brief explanation of what each of the four packages is about:

PICOS contains the source code of the PicOS operating system, libraries, documents, and sample applications (praxes) organized into separate directories (projects). Theoretically, you wouldn't need any of the remaining three packages, if you only wanted to develop software for real-life hardware using manual (command-line tools). But you would still need the toolchains for the respective CPU architectures (e.g., the MSPGCC compiler toolchain for MSP430).

VUEE brings in the set of libraries providing add-ons to SMURPH/SIDE to create a virtual environment for the emulated execution of PicOS praxes. It also contains a GUI to VUEE models (the `udaemon` script). VUEE is completely useless without SMURPH/SIDE and without PICOS.

SIDE contains an independent simulation/emulation package for networks and reactive systems (so it is useful on its own). It provides the low-level vehicle for building and executing VUEE models of (networked) PICOS praxes. Having the three packages mentioned so far, i.e., PICOS, VUEE, and SIDE, will allow you to develop PicOS praxes for real-life devices, as well as execute them virtually, using command-line tools.

PIP is an integrated SDK gluing PICOS, VUEE, and SIDE together. It provides a project view of the PicOS praxes and automates the procedures for their editing, configuring, compiling, uploading (into physical nodes), debugging, and virtual execution (as VUEE models). PIP includes `elvis`, which is a vi-compatible text editor used by the SDK. The editor was originally written by Steve Kirkendall and adapted by us to collaborate with PIP. `Elvis` requires a separate installation, because it requires root access (on UNIX systems).

Hardware

You need:

A PC or laptop capable of running Windows or Ubuntu Linux. The programs have been tested on Windows 10 + Cygwin (32 and 64-bit versions) and Ubuntu 16 (32 and 64-bit). Reasonably older systems should be fine. These days, with the popularity of virtual machines, it is no big deal to have a Ubuntu machine dedicated to the platform. Although we mostly use Windows 10 + Cygwin for our development, Ubuntu is probably friendlier for a beginner because: 1) the installation is a bit simpler (no Cygwin), 2) it offers a more convenient access to the debugger (gdb) for debugging programs running in real devices. By



the way: we practically never use gdb, at least not for debugging programs running in real devices. When you familiarize yourself with VUEE, you will understand why.

The remaining hardware items are needed, if you want to work with real devices. They are not needed, if you only want to compile and run VUEE models of the networks.

A JTAG programmer suitable for the target device. We have had some experience with these programmers:

MSP430/CC430:

MSP-FET430UIF (from Texas Instruments; this is what we mostly use)

EZ430-RF (aka RF2500, also from Texas Instruments)

Tiny USB JTAG (from OLIMEX)

CC1350:

XDS110 (this is the TI standard debugger interface for CC1350 that, e.g., comes with the CC1350 Launchpad)

DevPack Debug interface for CC1350 Sensortag

All the above programmers work under Windows as well as Linux (with assistance of some software). RF2500 is primarily used with EZ430-CHRONOS. Note that the XDS100 that comes “integrated” with the CC1350 Launchpad can be easily decoupled from the board and used as an independent, standalone programmer for other CC1350-based devices.

It is possible that other JTAG programmers will work, too (especially with mspdebug under Linux – see below), but we haven't tried them. PIP can be interfaced to any (reasonable) command-line flash programming software. And, obviously, GUI software can be used independently of PIP (if its integration poses a problem).

A USB to serial dongle is generally useful (needed) to provide the UART interface. Note that, e.g., the CC1350 Launchpad is already equipped with such a dongle (which comes with the XDS110 debugger interface). For MSP430/CC430 devices, we have been using TTL232R3V3 (it works both under Windows and Linux out of the box). It can be purchased at a number of places, including the manufacturer: <http://www.ftdichip.com/>. The connector on our workhorse EMSPCC11 device (the so-called Warsaw board) has been especially designed for that dongle. If you want to try other solutions, please keep in mind that PicOS UART drivers don't use the CTS/RTS flow control signals, i.e., only two pins on the connector (besides ground) are needed: pin 4 = RX and pin 5 = TX. Those pins are connected to the microcontroller pins and require 3V logic. Another advantage of the USB dongle is that it provides a handy external power supply for the EMSPCC11 board (5V on pin 3 down-regulated to 3V). The dongle also works with CC1350, including the Launchpad detached from the XDS110 debugger. When connecting the dongle by wires take caution to avoid using the 5V USB power supply wire. It can fry the microcontroller, if accidentally connected to one of its (3V) pin.

On all recent Windows versions the FTDI dongle needs no driver installation (Windows manages to locate the drivers on its own). If needed, the drivers are available from the [FTDI download area](#).

The requisite cables: (for the programmer), i.e., a USB cable, a JTAG cable, and so on.

MSPGCC (for MSP430/CC430)

These software components are required for programming real devices (as opposed to their VUEE models) based on MSP430/CC430 microcontrollers. In addition to our platform, you need the MSPGCC compiler. That compiler (together with the associated development tools) was launched well over a decade ago as a GNU open-source project (based on GCC) and



made available for Linux and Windows systems. In 2013 the project was kindly “adopted” by Texas Instruments which basically means that, while remaining an open-source project, it is no longer supported (or even influenced) by the community. While formally, the most recent official version of the compiler can be used with our platform, we strongly prefer to use instead the last community version (with a few tweaks). This is because the TI compiler generates much longer code (by about 30%). No big surprise: TI has to boast their commercial tools as being superior to the freebies that they so generously give away.

The last (vanilla) community version of MSPGCC (in a binary edition for Windows) is available from <https://sourceforge.net/projects/mspgcc/>. The TI version (Windows + Linux), including the sources, can be obtained here: <http://www.ti.com/tool/msp430-gcc-opensource>. In our **REPO**, we provide ready binaries for Windows (Cygwin) and Ubuntu which have been thoroughly tested with our platform.

ARM toolchain (for CC1350)

These software components are required to program real devices (as opposed to their VUEE models) based on CC1350 microcontrollers. The most recent toolchain can be downloaded from this link:

<https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads>

In our **REPO**, we provide the last version tested with our platform, but (with a very high likelihood) a newer version (if available) will be more than fine.

Ubuntu

If installing Ubuntu under Oracle VirtualBox, select version 3 (xHCI) of the USB controller. This seems to improve the reliability of communication with FET JTag programmers. Otherwise, MSPDebug (see below) may cause problems when more than one USB device is configured with the VM.

Regarding the MSP430/CC430 environment, things work on both 32 and 64-bit versions of Ubuntu, but you must install the right version of MSPGCC for your system. Note that this is in some contrast to Windows + Cygwin (see below). The 64-bit version is generally recommended these days.

The following instructions assume some expertise in Linux (Ubuntu). I believe that all the requisite Ubuntu packages needed by the platform are already included in the standard distribution. The platform needs gcc and g++ with the standard development environment (make, and so on) + Tcl/Tk (packages tcl and tk). A missing package will clearly show up; so just add it when that happens and try again.

Step 1 (not needed for programming and running VUEE models):

Set up the toolchains. Note that

For MSP430/CC430, install MSPGCC. Fetch the prepackaged compiler from our **REPO** and unpack it, e.g.,

```
tar xvfj mspgcc_v4_ubuntu_64.tar.bz2
```

This will unpack into a directory. Move to that directory, become root, and execute:

```
./install.sh
```

The installation script stores the package in `/usr/local/msp430` and creates this file: `/etc/ld.so.conf.d/mspgcc.conf`. This command:

```
rm -rf /usr/local/msp430 /etc/ld.so.conf.d/mspgcc.conf
```

completely uninstalls the compiler.



Following the installation, make sure to add `/usr/local/msp430/bin` to your shell PATH, so the compiler can be easily found by the platform.

For CC1350, install the ARM toolchain (see the above link). The most natural way to do that is to unzip and untar the tarball file into `/usr/local` and make a link from `/usr/local/arm` to the directory created this way. Then add `/usr/local/arm/bin` to your shell PATH.



Figure 1: Connecting EMSPC11 to the PC.

Step 2:

Make sure that you have a directory named `bin` or `BIN` in your home directory and that it is mentioned in your PATH. Unpack the software (see **Preamble**), then go to directory `PICOS` and execute:

```
./deploy
```

This will configure SIDE, set up a few symbolic links for VUEE, and copy some scripts to your `bin` (or `BIN`) directory. Note that `deploy` (as well as most other scripts used by the platform) is written in Tcl. This is our favorite scripting language.

Install the modified elvis editor that comes with PIP. On Cygwin, just execute

```
./instelvis
```

in PIP's directory. On Ubuntu, you should first check if you have the (standard) elvis package already installed. Use the package manager to find out if you have a package named "elvis". Most likely you don't (it doesn't get installed by default, and is not extremely popular these days), but if you do, remove it first. Our version of elvis is not installed as a package (perhaps some day the whole thing will become a package), but just "installed" as an executable program in `/usr/bin` with some files going into `/usr/share/elvis/`.

The MSP430/CC430 setup comes with MSPDebug flash loader/debugger (included in our version of the toolchain). This allows you to immediately flash target devices with the programmers listed above.



For CC1350, you have to install the uniflash [flash programmer from TI](#). You may also want to read the relevant fragments from the [swra446 document](#) by TI. I haven't been able to get any of the two versions of the uniflash loader to work under my 64-bit version of Ubuntu run in a virtual machine hosted in Windows 10. Version 4.1 complains about problems with USB communication related to FTDI (and says they are looking into that); version 3.4.1 almost works (like going through some parts of the drill), but doesn't quite. Your mileage may vary. Things will probably tend to improve with time. I wasn't desperate, because (even if I do the development under Linux) I can easily flash the device from Windows.

An exercise (MSP430):

You are basically set. To check whether things work, do this exercise. We assume that:

1. The platform has been installed as described in Steps 1 and 2 above.
2. You have a device (node) known by PicOS as one of the configured "BOARDS". In any case, if you have received the system from us, you probably have at least one such "board". In the remainder of this exercise, we assume that the device is EMSPCC11 (aka the Warsaw board).
3. You have one of the supported flash programmers, e.g., [MSP-FET430UIF](#) (from Texas Instruments).
4. You have a means to connect the node's UART to the PC, such that it appears as a `/dev/TTYUSBx` device. For EMSPCC11, this is probably a [TTL232R3V3](#) dongle from [FTDI](#). Strictly speaking, this item is not absolutely required.

Make sure that you (as a Ubuntu user) can control the USB devices. This boils down to making yourself a member of the dialout group (edit `/etc/group` and add the user `ld` at the end of the dialout entry). Re-login if necessary.

Plug the programmer to the PC and connect its other end to the node. Also connect the node's UART to the PC, as shown in the picture above. When you do that, you should see two new devices: `/dev/ttyUSB0` (the UART) and `/dev/ttyACM0` (the programmer). The numbers may differ, if you happen to have other serial devices connected to the PC.

Execute this command:

```
pip
```

in a terminal window. You may type:

```
pip -D
```

instead (especially if doing it for the first time), to see debug messages produced by `pip`, which may help to detect problems, if any. This command invokes a script, which has been put into your personal bin (or BIN) directory by `deploy`, implementing the platform's GUI SDK.

Select "Open project" from the "File" menu, then choose `EXAMPLES → INTRO → LEDS`, by double-clicking on the respective directories in the file browser. When you double-click on the last directory, i.e., `LEDS`, the browser window will show no more directories (you are within the `LEDS` project directory). Then click "OK" in the window. You have entered the `LEDS` project. It amounts to a very simple program that blinks a LED.

Select "Arch+Board" from the "Configuration" menu. In the window that pops up make sure that the "Lib mode" box is unchecked, the "Arch" selection is `MSP430`, and the "Board" selection agrees with your device (it should be `WARSAW` for `EMSPCC11`). Click "Done" when finished.

Click "Build (make)" in the "Build" menu. You should see various compilation messages in the main display pane of the PIP window. If the compilation is successful, which it should be, you will see at the end the list of sizes of the various sections of the flashable image produced by the compiler terminated by the string:



--DONE--

Connect to the node's UART (optional). For that, select "Start piter" from the "Execute" menu. This opens a terminal emulator window. Click the "Connect" button in that window and select the right device from the topmost menu in the popup window that the button has triggered. In the simple case when there is a single ttyUSB device, it should automatically appear at the top. Make sure that "Speed" is 9600 and "Prot" is Direct (these should be the defaults). Then click "Proceed" or "Save & Proceed". The terminal emulator is now connected to the device's UART.

Flash the node. First configure the loader. Select "Loaders" in the "Configuration" menu. In the window that pops up, select the MSPDebug loader (by clicking on the radio button), then make sure that:

FET Device selection is "Auto"

The driver is "tilib"

The "allow firmware update" box is checked

GDB connection port is "none"

Click "Done" to close the window and finalize the loader selection. Note that most likely you will need a firmware update on the programmer when you use it for the first time. In subsequent scenarios, following a successful firmware update, the box can be unchecked.

Power on the device (I mean EMSPC11). If there are no batteries, this means pushing the switch towards the dongle socket.

Note that the firmware update procedure can be capricious, so you may have to try it several times. Generally, once we get pass this stage, communication with the programmer becomes reasonably smooth.

The programmer writes messages to the PIP window. The flashing procedure consists of erasing the precious contents of the node's flash memory and writing the new image over it. At the end of the sequence of messages indicating successful programming, you should see something like:

```
Programming...
Writing 4096 bytes at 4000 [section: .text]...
Writing 4096 bytes at 5000 [section: .text]...
Writing 4096 bytes at 6000 [section: .text]...
Writing 2272 bytes at 7000 [section: .text]...
Writing 870 bytes at 78e0 [section: .rodata]...
Writing 16 bytes at 7c46 [section: .data]...
Writing 32 bytes at ffe0 [section: .vectors]...
Done, 15478 bytes total
MSP430_Run
MSP430_Close
--DONE--
```

At the end, the programmer resets the device, so the program starts. One of the LEDs should start blinking and the terminal window should show lines looking like this:

```
PicOS v5.0/PG170419A-WARSAW, (C) Olsonet Communications, 2002-2017
Leftover RAM: 9802 bytes
Commands:
on
off
```

This concludes the exercise.

Beware that the flash loader is occasionally capricious and may signal a communication error when invoked for the first time after connecting the programmer to a USB port (error



number 35). In such a case, simply try again. Typically, this error occurs only before the first use of the programmer, immediately after plugging it in.

If MSPDebug complains about errors in accessing USB devices (hinting at inadequate permission), the crude but effective solution is to make mspdebug setuid root, i.e., become root, cd to /usr/local/msp430/bin and do:

```
chmod +s mspdebug
```

While such solutions are not recommended for serious production systems, doing this on a virtual machine dedicated to running the platform makes perfect sense.

Windows

You have to start by installing Cygwin from <http://www.cygwin.com/>. Follow the instructions, i.e., download the installer (setup.exe) and so on. Note that you can install either the 64-bit version of Cygwin (recommended if you have a 64-bit machine), or the 32-bit version (works under both 32 and 64-bit Windows), or even both.

If you have the 64-bit version of Cygwin, then you can install either the 32-bit or the 64-bit version of MSPGCC. If you have the 32-bit version of Cygwin, then the 64-bit version of MSPGCC isn't going to work.

When installing Cygwin, select the recommended setting "For all users". To avoid problems with missing items, you may just install *everything*. If you want to economize (I do it these days, because, e.g., the installation of TeX/LaTeX takes forever), then you will have to keep adding things, if they appear missing. No big deal. Just make sure to install initially X11, gcc, g++, make, Tcl, and Tk.

Then install the MSPGCC compiler. Fetch the proper binary version from our [REPO](#) (it comes as a 7z archive) and unpack it into C:\mspgcc. You can extract the archive into C:\ and subsequently rename the containing folder to mspgcc. You should see there these subfolders: bin, lib, libexec, msp430, and share.

Make sure that C:\mspgcc\bin is appended to the system PATH (in a way that will work both for Windows and for Cygwin). On Windows 10, go to "Control Panel" → "System and Security" → "System" → "Advanced system settings" → "Environment variables". Select "Path" in the lower pane and click "Edit". Then click "New" and add C:\mspgcc\bin\, e.g., at the end of the list.

You may want to reboot (or at least restart Cygwin) to make sure that the change has been noticed. Execute:

```
echo $PATH
```

in a Cygwin terminal to see if the new addition to PATH is there.

The rest, including the exercise is exactly as for Linux, assuming that you have invoked X11 under Cygwin and are entering commands from an xterm. Please read the introduction to PIP (the link can be found in the [REPO](#)).

As an option, you may prefer to use under Windows the native version of Tcl/Tk from ActiveState (native, in the sense that it doesn't rely on Cygwin's X11), which gives a somewhat nicer appearance of Tk windows. For that, install the most recent "community edition" (free) version of Tcl/Tk from here: <http://www.activestate.com/activetcl/downloads>. Then, to make sure that the platform will use the ActiveState Tc/Tk version whenever it can, you have to add links to the respective binaries to you personal BIN directory, like this:

```
cd ~/BIN
ln -sf /cygdrive/c/ActiveTcl/bin/tclsh.exe tclsh86
ln -sf /cygdrive/c/ActiveTcl/bin/wish.exe wish86
```



The source paths may differ in your setup, depending on your installation of the ActiveState package, but the names under which the programs are made visible from BIN must be as shown.

Then, when you provide the `-l` argument to deploy (this is el, not capital aye), i.e.,

```
./deploy -l
```

then PIP and a few other GUI scripts will opt to use the ActiveState Tcl/Tk for their execution environment. Note that this doesn't make the platform Cygwin-independent, or even X11-independent.

For CC1350, you may want to install flash programming software from TI, e.g., Flash Programmer 2 from this link:

<http://www.ti.com/tool/flash-programmer>

The package consists of two flash programmers: GUI + command line. The first one can be used independently of PIP, the latter can be configured with PIP, so it can be conveniently invoked from PIP's Execute menu. For that, within a CC1350 project, click Configuration → Loaders ... and fill in the "Command line" section. First browse to the executable file of the command-line loader (the file is somewhat confusingly named `srprog.exe` and can be found in the bin directory of the installation), then enter this string into the Arguments field:

```
-t lsidx(0) -e pif -p -f %f.hex
```

The sequence `%f` will be replaced by the root file name of the flashable image when the loader is invoked. You can learn about the arguments by running `srprog.exe` from a DOS command prompt.

VUEE

To check whether VUEE/SIDE has been installed correctly, try running the same LEDS program under the emulator. Under PIP, while still within the LEDS project, choose Build → VUEE. After a while, the compiler will create the model. Click Configuration → VUEE and make sure that the "Run with udaemon" box is checked. Then click Execute → VUEE. On Windows, this may trigger a firewall warning, because the model needs to open a network socket to communicate with its GUI. Just let the program do it.

The model will start (you will see some messages in the main pane of the PIP window) and a small `udaemon` (model interface) window will pop up. The trivial network in our example consists of a single node, so there isn't a lot you can do with it. Enter 0 into the "number" field (this is the node number), select LEDS in the menu to the right (initially showing UART) and click "Connect". You will see the three LEDs of the node with the green LED blinking, as in the real node.

Note that you can also connect to the node's UART. For that, select "UART (ascii)" in the menu and click "Connect again". You will see the output produced by the node. You can switch the blinking on and off by entering the respective commands from the UART.

