Building and Installing GNU `units` on Microsoft Windows with Microsoft Visual Studio

Edition 1 for `units` Version 2.13

Jeff Conrad

This manual is for building GNU `units` (version 2.13) with Microsoft Visual Studio on Microsoft Windows.

Table of Contents

Preface 1

Building from the Windows Command Prompt . . 1

Icons and File Association 2

Currency Definitions Updater 2

Installing Python 2
Installing `units_cur.py` 2
Setting `PATHEXT` 2
Python Version 3
Example 3

Running the Updater 4

Updating from a Command Prompt 4
Automatic Updates 4

Preface

This manual covers building and installing GNU units on Windows, using Microsoft Visual Studio from the Windows command prompt. You may be able to import Makefile.Win into the Visual Studio IDE, but that is beyond the scope of this document.

If you have Unix-like utilities, you may be able to build and install in much the same manner as on most Unix-like systems, perhaps with a few minor adjustments. Versions 2.12 and earlier were built using Microsoft Visual C/C++ 6.0, Visual Studio Express 9.0 and 10.0, and the MKS Toolkit version 9.6 under Windows XP, SP3. Version 2.13 was built using Microsoft Visual Studio 2015 and the MKS Toolkit version 10.0 on Windows 10—see *UnitsMKS* for the details.

A Windows binary distribution is available on the project website; the resulting installation is essentially the same as that using Makefile.Win, and usually can be achieved with less effort.

The most recent build was for units version 2.13, using Microsoft Visual Studio 2015 on Microsoft Windows Professional 10 on 6 June 2016.

— Jeff Conrad (jeff_conrad@msn.com) 6 June 2016

Building from the Windows Command Prompt

If you have Microsoft Visual Studio but don't have Unix-like utilities, you should be able to build and install units from the Windows command prompt using Makefile.Win:

```
nmake /f Makefile.Win
nmake /f Makefile.Win install
```

The build requires that many environment variables be properly set; the easiest way to do this is to select Developer Command Prompt in the Visual Studio folder on the Start menu, and then change to the units source directory.

If you install in the default location, you'll probably require elevated privileges; the easiest way to do this is to right-click on Developer Command Prompt in the Visual Studio folder on the Start menu, and select Run as administrator.

By default, the units executable and data files are placed in the directory given by %ProgramFiles(x86)%\GNU\units; in most cases, this is C:\Program Files (x86)\GNU\units. On a 32-bit Windows system, the directory should be changed to %ProgramFiles%\GNU\units.

You can preview the installation directories with

```
nmake /f Makefile.Win showdest
```

If the destination directories don't exist, they will be created during installation. You can change these locations by editing Makefile.Win.

If you want to run units from a command prompt or from the Start Menu Run box, you can add the installation directory to the PATH environment variable. Alternatively, you can create a shortcut to the program and place it in a convenient location.

Icons and File Association

The installation process associates units data files with the notepad editor; double-clicking on the file icon opens the file for editing. The installation process makes unitsfile.ico the default icon for these files. An additional icon file, unitsprog.ico, is embedded in the executable file as part of the build process; this icon also may be useful if you wish to create a shortcut to the units program. Both icons are copied to the units installation directory.

Currency Definitions Updater

The script units_cur.py can be used to update currency definitions (if your system hides file extensions, this script will display as units_cur). The script requires Python (available from http://www.python.org/) and the unidecode package (available at http://pypi.python.org/).

Installing Python

If you want to use the currency updater, install Python and then install the unidecode package. Unless you have (or anticipate having) applications that depend on Python 2, the best choice is probably to install Python 3.

To install the unidecode package, follow the instructions in the PKG-INFO file included with the package. You then should be able to run units_cur.py using the shortcut on the Start Menu, or if you have added the units installation directory to your PATH, from a command-prompt window.

Installing units_cur.py

To create the appropriate script for the version of Python that you will be using, use either

```
nmake /f Makefile.Win currency2 (for Python 2)
```

or

```
nmake /f Makefile.Win currency3 (for Python 3)
```

The script will then be installed when using the install target.

Setting PATHEXT

If you add .py to the PATHEXT environment variable, you can simply type units_cur to run the updater from a command window. You can do this from a command-prompt window by typing

```
set PATHEXT=%PATHEXT%;.py
```

but you'll need to do this with every new instance. You can make a permanent change by adding ;.py to PATHEXT from the Advanced tab of the System dialog: click the 'Environment Variables' button, find PATHEXT in either the list of User variables or the list of System variables; click the 'Edit' button, make the change, and click 'OK'.

Python Version

By default, the currency updater is configured to use Python 3. If you have older programs that depend on Python 2 and do not wish to install Python 3, rename `units_cur` to `units_cur3` and rename `units_cur2` to "`units_cur`". If your system is configured to not hide filename extensions, these files will show a `.py` extension; if that's the case, be sure to retain the `.py` when renaming, because the extension is needed for Windows to know how to process the script.

If you have both Python 2 and Python 3 installed, the Python launcher will use the latest installed version of Python 2 by default, and the default `units_cur` will fail. The easiest approach here is to run nmake using the `currency2` target as shown above before installing to let Python 2 handle the script—the result from either script is the same. If you want to use Python 3, you can do it several ways, including

- Changing the first line of the default units_cur from `#!/usr/bin/python` to `#!python3`. The default directive is for compatibility with Unix-like systems; the Python launcher for Windows simply interprets it to mean "use Python", and doesn't actually expect to find the program in `/usr/bin`. The `#!python3` form tells the Python launcher to find and use Python 3.

  This should work fine for double clicking the script's icon or running it from a command-prompt window, but it may fail if the script is run from a Unix-like shell that interprets the `#!` directive literally.
- Confirming that the Python launcher `py.exe` is in the Windows directory (usually `C:\Windows`) and changing the first line of the default `units_cur` from `#!/usr/bin/python` to `#!C:\Windows\py.exe -3`"; this will cause the Python launcher to use the latest installed version of Python 3. A fully qualified pathname is interpreted literally by the Python launcher, so if the Python launcher is located elsewhere, the first line should give that location.

  This approach should work for a Unix-like shell as well as the Windows command interpreter.
- Setting the environment variable `PY_PYTHON` to `3`; the best way to do this is from the Advanced tab of the System dialog. This also should work for both Windows and Unix-like command interpreters, but it will affect all Python scripts.

Example

If you are using Python 3 and installing units in the default location of `C:/Program Files/GNU/units`, the process would be to

1. Build the executable by running

   ```
   nmake /f Makefile.Win
   ```
2. Create the currency updater script by running

   ```
   nmake /f Makefile.Win currency3
   ```
3. If necessary, modify `units_cur.py` so that the output file is given by

   ```
   outfile = 'C:/Program Files/GNU/units/currency.units'
   ```
4. Confirm the installation location by running

   ```
   nmake /f Makefile.Win showdest
   ```

It is assumed that the program will be installed in a subdirectory of the standard location for executables (typically, C:\Program Files (x86) on a 64-bit system or C:\Program Files on a 32- bit system), and a warning is given if this directory does not exist. Ignore the warning if you are intentionally installing in another location.

5. Install the files by running

```
nmake /f Makefile.Win install
```

6. Ensure that currency.units is writable by ordinary users. The installation should do this automatically, but if for some reason it does not, set permissions manually by adding 'Modify' permission for the appropriate groups (typically 'Power Users' and 'Users')

Running the Updater

Updating from a Command Prompt

Unless you run the currency-update script from the program installation directory, you will need to modify units_cur.py to give the full pathname of the output file currency.units, i.e., change

```
outfile = 'currency.units'
```

to

```
outfile = 'installation_directory/currency.units'
```

For the default installation directory on a 64-bit system, this would be

```
outfile = 'C:/Program Files (x86)/GNU/units/currency.units'
```

Be sure to use forward slashes to avoid confusing Python. The best approach is to modify this file before installation after creating it with the currency*?* target in Makefile.Win.

Automatic Updates

The easiest way to keep currency values up to date is by having the Windows Task Scheduler run units_cur.py on a regular basis. The Task Scheduler is fussy about the format for the action, which must be an executable file; an entry might look something like

```
C:\Windows\py.exe "C:\Program Files (x86)\GNU\units\units_cur.py"
```

if the Python launcher is in C:\Windows and the script is in C:\Program Files (x86)\GNU\units. The program must start in the units installation directory; the starting directory must be specified *without* quotes.