BASCOM Compiler unter Linux t Geany als IDE nutzen

Die Idee

Die BASCOM IDE läuft zwar prinzipiell unter Wine, aber nicht sehr gut.
Ausserdem möchte ich Geany auch für BASCOM als Editor benutzen, da er mir besser gefällt.

Kann man den Compiler von Geany aus aufrufen? Die Antwort ist: ja, mit einem kleinen Hilfs-Script in Python.
Voraussetzung ist eine funktionierende BASCOM-Installation in einem Verzeichnis irgendwo im Home-Folder (gebraucht werden nur Compiler, DAT-Files und Library-Verzeichnis).

Vorgehensweise

In Geany wird die .BAS Datei erstellt.

Dann wird im **Build-Menü** ein neues Menü-Item „BAS Compile“ erstellt und als Kommando (Achtung: dies muss mit einer geladenen .BAS-Datei geschehen, da die Menüs vom Dateityp abhängen!)

```
python <Path to bascompile script> %f
```

eingegeben, z.B. python /home/jcf/bascom/bascompile.py %f

Geany ersetzt dann %f durch den Dateinamen.

Die Aufgaben des Scripts **bascompile.py** sind folgende:

- Herausfischen des übergebenen Dateinamens mithilfe von sys.argv[1]
- Lesen der BAS-Datei in einen Textstring
- Zerlegen des Textstrings in ein Array von Zeilen
- Suche in allen Zeilen nach
    - $regfile = ….
      Dies wird für das Argument CHIP = … des Compilers benötigt.
      Die Auswertung erfolgt anhand eines Dictionaries, der momentan nur wenige Chips enthält. Er kann aber leicht erweitert werden. Die nötige Information ergibt sich, indem man den BASCOM Compiler ohne Parameter aufruft.
    - $hwstack, $swstack, $framesize
      Auch diese werden in Compilerdirektiven übersetzt.
- Aufruf des Compilers

Geany zeigt dann automatisch den Output des Compilers an, und man sieht ob Fehler aufgetreten sind.

Bemerkungen:

- Dieses einfache Script macht (fast) keine Fehlerüberprüfung, man sieht sofort im Compiler-Fenster von Geany was los ist.
- **Die Stack-Direktiven müssen im Quelltext vorhanden sein, sonst gibt es Fehlermeldungen.**

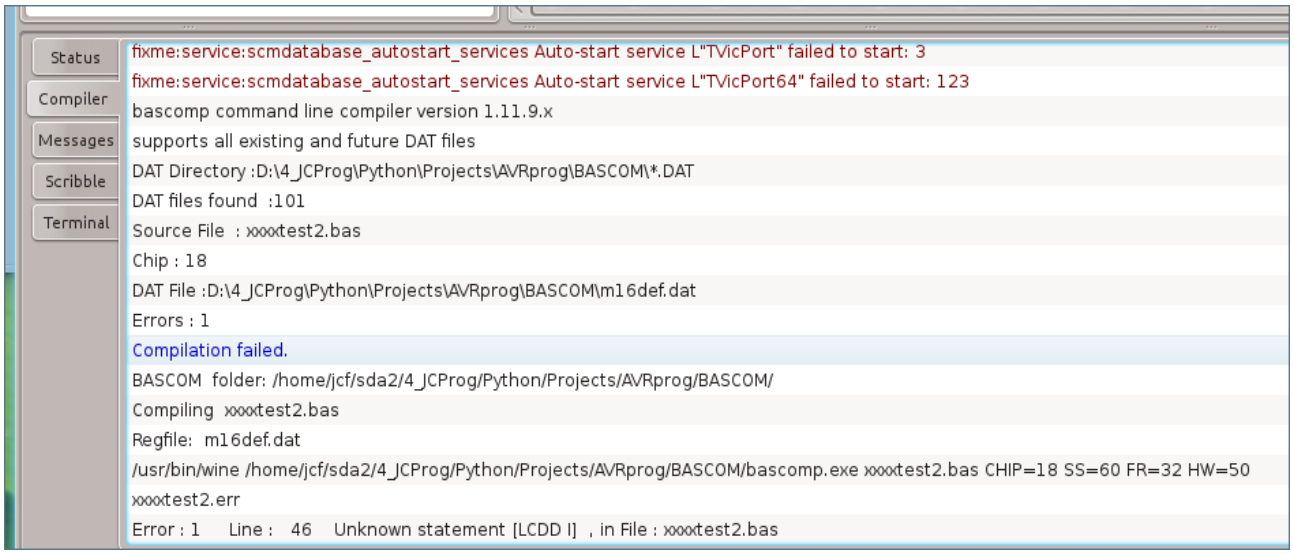
Das Python Script bascompile.py

```
""" Helper program called by Geany on execution of Build command

Bascom compiler and libraries must be found in a subfolder BASCOM
under the folder where this script is located.

Supports only M8, M16 and M32 for the moment!
New chips must be added in the dictionary conversion in the EDIT AREA

In Geany build command from the Build menue must be set to something like
python /home/.../bascompile.py %f
python <Path to this script> %f

THE BAS FILE MUST CONTAIN STACK INFO !

"""

#.................................................................
# EDIT HERE

# This is the dictionary that translates regfile to CHIP number
conversion = { "m8def.dat":"CHIP=17",
    "m16def.dat":"CHIP=18",
    "m32def.dat":"CHIP=23"}

# END EDIT AREA
#.................................................................

import sys
import os.path
import subprocess


def progdir():
    """returns dir in which program lives"""
    return os.path.dirname(os.path.abspath(__file__))


def readBAS (myfile):
    """ read BAS file to text t"""
    f=open (myfile,'r')
    t=f.read()
    f.close()
    return t

def read_err(myfile):
    myerrfile=os.path.splitext(myfile)[0]+".err"
    if os.path.isfile(myerrfile):
        print myerrfile
        f=open (myerrfile,'r')
        t=f.read()
        f.close()
        return t
    else:
```

```
        return ""

def search_for_item(t,item):
    """searches for item (e.g. '$regfile' in test t)
    returns line containing item
    """
    tlines=t.splitlines()
    for line in tlines:
        l=line.strip()    # strip white spaces
        l=l.lower()       # convert to lower letters
        if len(l)>0:
            if l[0]!= "'":    #only lines that are not rem'd out
                if item.lower() in l:

                    # strip eventual comment at the right of line:
                    if "'" in l:
                        l=l.split("'")
                        l=l[0]

                    return l


def identify_chip(t):
    """Searches for a line containing $regfile= ....
    and returns CHIP=... for compiler"""
    myline = search_for_item(t, "$regfile")
    regfile = myline.split("=")
    c=regfile[1]
    c=c.strip()         # strip whites
    c=c.strip('\"')     # strip "

    print "Regfile: ",c
    chip=regfile_to_chip(c)
    return chip


def regfile_to_chip(regfile):
    """converts regfile e.g. 'm8def.dat' to CHIP=17
    For the moment only for M8, M16, M32
    Info for this conversion is obtained by just executing bascomp with
    no arguments"""

    try:
        return conversion[regfile]
    except:
        print "Unknown chip!"
        raise

def get_hwstack(t):
    myline = search_for_item(t, "$hwstack")
    s = myline.split("=")
    c=s[1]

    c=c.strip()         # strip whites
    return c

def get_swstack(t):
    myline = search_for_item(t, "$swstack")
    s = myline.split("=")
    c=s[1]

    c=c.strip()         # strip whites
    return c
```

```
def get_framesize(t):
    myline = search_for_item(t, "$framesize")
    s = myline.split("=")
    c=s[1]

    c=c.strip()         # strip whites
    return c

#-------------------------------------------------------------------

""" Main"""

bascomfolder = os.path.join(progdir(), "BASCOM/")
print "BASCOM  folder:", bascomfolder

myfile=sys.argv[1]
print "Compiling ", myfile

try:
    # read BAS file and extract info from it
    t=readBAS(myfile)
    chip=identify_chip(t)
    hwst = get_hwstack(t)
    swst = get_swstack(t)
    frsize = get_framesize(t)

    # setup compiler command
    comp=os.path.join(bascomfolder, "bascomp.exe")
    compiler_cmd = comp +" " + myfile + " " + chip + " SS="+swst + " FR=" +
frsize + " HW=" + hwst
    ##print compiler_cmd
except:
    raise

# compile
cmd="/usr/bin/wine " + compiler_cmd
print cmd
os.system(cmd)

# read and print error output (if there is)
e=read_err(myfile)
print e

# leave script with return value according to error level
if e == "":
    sys.exit(0)     # no error
else:
    sys.exit(1)     # error: compilation failed
```

Vom Script wird auch der Return-Wert mit sys.exit() gesetzt, so dass Geany weiss ob die Kompilation erfolgreich war oder nicht, und dies auch meldet, mit „Compilation finished successfully“ bzw. „Compilation failed“.
Im Fehlerfall zeigt Geany sogar die Zeilennummer des Fehlers an.



Nachtrag 23.2.2020: Wine meckert da die 32 Bit-Umgebung nicht installiert ist. Dies kann man aber nachholen, siehe Fehlermeldung.

Brennen aus Geany heraus

Wenn das Compilieren so schön funktioniert, geht das auch mit dem Brennen?
Ja, mit AVRDude und einem mkII-kompatiblen Programmiergerät.

Voraussetzung ist natürlich ein installiertes AVRDude.

Hierzu wird ein ähnliches Script geschrieben, welches folgendes tut:

- Herausfischen des übergebenen Dateinamens mithilfe von sys.argv[1]
- Lesen der BAS-Datei in einen Textstring
- Zerlegen des Textstrings in ein Array von Zeilen
- Suche in allen Zeilen nach

    - $regfile = ….
      Dies wird für das Argument -p = …. des Compilers benötigt.
      Die Auswertung erfolgt anhand eines Dictionaries, der momentan nur wenige Chips enthält. Er kann aber leicht erweitert werden. Die nötige Information ergibt sich, indem man im Manual von AVRDude nachschaut.
- Aufruf von AVRDude mit den nötigen Parametern.

Das Script wird wieder unter Build commands in Geany eingetragen.

```
""" flash_mkII.py
Helper script for Geany to flash microcontrollers with an mkII.
For the moment only a few chips are defined, but the dictionary can easily be
adapted to new chips by editing the conversion dictionary.

In Geany a build command from the Build menu must be set to something like
python /home/.../flash_mkII.py %f
python <Path to this script> %f
```

```
"""

import sys
import os.path

#.............................................................
# EDIT HERE

# This is the dictionary that translates regfile to CHIP number
conversion = { "m8def.dat":"m8",
    "m16def.dat":"m16",
    "m32def.dat":"m32"}
#.............................................................



def readBAS (myfile):
    """ read BAS file to text t"""
    f=open(myfile,'r')
    t=f.read()
    f.close()
    return t

def search_for_item(t,item):
    """searches for item (e.g. '$regfile' in test t)
    returns line containing item
    """
    tlines=t.splitlines()
    for line in tlines:
        l=line.strip()    # strip white spaces
        l=l.lower()       # convert to lower letters
        if len(l)>0:
            if l[0]!= "'":    #only lines that are not rem'd out
                if item.lower() in l:

                    # strip eventual comment at the right of line:
                    if "'" in l:
                        l=l.split("'")
                        l=l[0]

                    return l


def identify_chip(t):
    """Searches for a line containing $regfile= ....
    and returns CHIP=... for compiler"""
    myline = search_for_item(t, "$regfile")
    regfile = myline.split("=")
    c=regfile[1]
    c=c.strip()         # strip whites
    c=c.strip('\"')     # strip "

    print "Regfile: ",c
    chip=regfile_to_chip(c)
    return chip

def regfile_to_chip(regfile):
    """converts regfile e.g. 'm8def.dat' to m8 for AVRDude
    as defined in the conversion dictionary
    Info for this conversion is obtained by AVRDude manual"""

    try:
        return conversion[regfile]
```

```
    except:
        print "Unknown chip!"
        raise


def get_hex_name(myfile):
    f,e = os.path.splitext(myfile)
    return f + ".hex"

#--------------------------------------------------------------------

# MAIN

myfile=sys.argv[1]
t=readBAS(myfile)
chip=identify_chip(t)
myhexfile=get_hex_name(myfile)
print "Flashing ", myhexfile

e=os.system("avrdude -c avrispv2 -P usb  -p " + chip + " -U flash:w:" +
myhexfile)

# leave script with return value according to error level
if e == 0:
    sys.exit(0)     # no error
else:
    sys.exit(1)     # error: flashing failed
```

BASCOM Hilfe

Was noch fehlt ist die Hilfe-Funktion, mit der man schnell die Syntax eines Befehles nachschlagen kann.
Die BASCOM-Hilfe ist in der Datei BASCAVR.CHM gespeichert, im Microsoft Compiled Help Format. Mit **archmage** (`sudo apt-get install archmage`) kann dies in HTML konvertiert werden.

Im BASCOM-Folder:

```
archmage BASCAVR.chm
```

Nun hat man einen Folder BASCAVR_html, in dem die BASCOM-Hilfe dekomprimiert ist.

In Geany kann diese Hilfe aufgerufen werden, indem man sich einen neuen Menüpunkt im Build-Menü macht, der die Hilfe-Datei mit dem Browser aufruft, z.B.

```
firefox /home/..../BASCOM/BASCAVR_html/arch_contents.html
```