farique1 / **MSX-Basic-Tokenizer**

Code | Issues 1 | Pull requests | Actions | Projects | Wiki

master | Go to file | Add file | Code

Fred Rique and **Fred Rique** v1.3 Pyt… ... on 5 Mar 27

Images	v1.2	8 months ago
Tools	new versions	12 months ago
.gitignore	v1.1	14 months ago
Changelog.md	v1.3 Python 3 and +	7 months ago
LICENSE	Create LICENSE	8 months ago
MSXBatok...	v1.2	8 months ago
README.md	v1.3 Python 3 and +	7 months ago
msxbatoke...	v1.3 Python 3 and +	7 months ago
openMSXB...	v1.2	8 months ago
openMSXb...	v1.3 Python 3 and +	7 months ago

About

Convert MSX Basic ASCII programs to tokenized format

#msx #msx-basic #basic #conversion #tokenization

Readme

MIT License

Releases

No releases published

Packages

No packages published

Languages

● **Python** 100.0%

README.md

MSX Basic Tokenizer

v1.3 - Python 3.8!

Tokenize ASCII MSX Basic programs.

> **MSX Basic Tokenizer** is now fully integrated with the **Badig** ecosystem.
> It can be automatically called by the build system on **MSX Sublime Tools** and from **MSX Basic Dignified**.

How to use

`msxbatoken.py <source> [destination] [-do] [-el [0-32]] [-vb <0-5>] [-frb]`

Arguments can be passed on the code itself, on `MSXBatoken.ini` or through the command line with each method having a priority higher than the one before.

Arguments

- *Source File*
  The ASCII file to tokenize.
  arg: `<source>` ini: `file_load =`

- *Destination FIle*
  The tokenized file.
  arg: `[destination]` ini: `file_save =`

  If `destination` is not given, the file will be saved as `source` with a `.bas` extension.
  An error will be given if `destination` is the same as `source`.

- *Delete Original:*
  Delete the `source` file after the tokenized version is successfully saved.
  arg: `-do` ini: `delete_original = [true|false]`

- *Export List:*
  Saves an `.mlt` list file similar to the ones exported by assemblers with the tokens alongside the ASCII code and some statistics.
  arg: `-el [0-32]` ini: `export_list = [0-32]`

The `[0-32]` argument refer to the number of bytes shown on the list after the line number.
The max value is `32`. If no number is given the default of `16` will be used. If `0` is given the list will not be exported.

The format for each line is, eg:

```
 80da: ee80 7800 44 49 ef 50 49 f2 1f 41 31
41 59 26 53 60 00 00        120 DI=PI-
3.1415926536
```

- *Bytes 1-2*: The MSX Basic memory address of the line.
- *Bytes 3-6*: The first four bytes with the next line address and the line number.
- *Bytes 7-...*: The tokenized line.
- The line in ASCII.

> The `.mlt` file uses the same MSX Sublime Tools syntax highlight as the MSX Basic.

- *Verbose level:*
The amount o information given.
arg: `-vb <0-5>` ini: `verbose_level = [0-5]`

`0` - Silent
`1` - Errors
`2` - Errors + Warnings
`3` - Errors + Warnings + Steps
`4` - Errors + Warnings + Steps + Details
`5` - Errors + Warnings + Steps + Details + Tokenization

The 'Tokenization' verbose level shows the process byte by byte on each line.

```
10 PRINT "WH"
20 GOTO 10
```

Will output:

```
|10 PRINT "WH"
```

```
    0a00|PRINT "WH"
    0a0091| "WH"
    0a009120|"WH"
    0a00912022|WH"
    0a0091202257|H"
    0a009120225748|"
    0a00912022574822|
    |20 GOTO 10
    1400|GOTO 10
    140089| 10
    140089200e0a00|
```

- *From Build:*
  Tells **MSX Basic Tokenizer** it is running from a build system (or an external program) and adjust some behaviours accordingly.
  arg: `-frb`

Notes

MSX Basic Tokenizer was tested with over 100 random basic programs from magazines and other sources and some programs crated to stress the conversions, however there should be still some (several?) fringe cases not covered.
Be careful.

There are some known discrepancies between **MSX Basic Tokenizer** and the MSX tokenization, most of them regarding errors on the code. They are:

- MSX &b (binary notation) tokenizes anything after it as characters except when a tokenized command is reached. The implementation here only looks for 0 and 1, reverting back to the normal parsing when other characters are found.
- Spaces at the end of a line are removed. The MSX does not remove them if loading from an ASCII file, only if typed on the machine.
- The MSX seems to split overflowed numbers on branching instructions (preceded by 0e), here it throw an error.
- Syntax errors generate wildly different results from the ones generated by the MSX.

Some errors on the code stop the conversion. They are:

- Line number too high, line number out of order, lines not starting with numbers, branching lines too high.
- Numbers bigger than their explicit type (in some cases they are converted up as per on the MSX.)

openMSX Basic (de)Tokenizer

v1.3 - Python 3.8!

Uses **openMSX** to convert a basic program from ASCII to tokenized or vice-versa.

It calls **openMSX** headless (without screen) and with throttle, mount a path (current = default) as a disk, load a basic file from this path, saves it with the chosen format and closes **openMSX**.

The path to an installation of **openMSX** is needed and a machine can be chosen to overwrite the default one. A disk drive extension can also be chosen for machines without one, it will be plugged on the slot A by default but it can force to slot B by putting `:SlotB` after its name.

> Be careful with the folder used as a disk. **openMSX** respects the MSX disk limitations of size (size of all the files must not be greater than the emulated disk size), file name sizes and characters.
> Always work on copies.

> **MSX Basic Tokenizer** is now fully integrated with the **Badig** ecosystem.
> It can be automatically called by the build system on **MSX Sublime Tools** and from **MSX Basic Dignified**.

How to use

```
openmsxbatoken.py <source> [destination] [-of
<t|a>] [-do] [-vb <0-5>] [-frb]
```

Arguments can be passed on the Python code itself, on `openMSXBatoken.ini` or through the command line with each method having a priority higher than the one before.

openMSX setup

On `openMSXBatoken.ini` specify:

- An alternate machine to overwrite the default one.
  `machine_name =` (opitional)
- A disk extension for machines without one.
  Plugged on the slot A by default, can force to slot B by putting `:SlotB` after its name.
  `disk_ext_name =` (optional)
- The path to `openMSX.app` on an installation of **openMSX**.
  `openmsx_filepath =` (required)

> There are no command line arguments for these.

Arguments

- *Source File*
  The file to convert.
  arg: `<source>` ini: `file_load =`

  If a path (absolute) is given, this path will be used to mount the disk on the MSX.
  If only a name is given the current path will be mounted as a disk on the MSX.
  Spaces on files will be replaced with an `_` for compatibility with the MSX filesystem.

- *Destination File*
  The file to be saved.
  arg: `[destination]` ini: `file_save =`

If a path is given it will be ignored with a warning.
The file will always be saved on the previously mounted disk.
If no name is given the `source` name will be used with a `.bas` or `.asc` extension accordingly.
If the file already exists an error will be given.
Spaces on files will be replaced with an `_` for compatibility with the MSX filesystem.

- *Output Format*
  The format to save.
  arg: `-of <t|a>` ini: `output_format = [t|a]`

  `t` is the default and save the file in the tokenized format. `a` saves the file in ASCII.
  Tokenized files will receive a `.bas` extension and ASCII files an `.asc` one.

- *Delete Original:*
  Delete the `source` file after the converted version is successfully saved.
  arg: `-do` ini: `delete_original = [true|false]`

- *Verbose level:*
  The amount o information given.
  arg: `-vb <0-4>` ini: `verbose_level = [0-4]`

  `0` - Silent
  `1` - Errors
  `2` - Errors + Warnings
  `3` - Errors + Warnings + Steps
  `4` - Errors + Warnings + Steps + Details

- *From Build:*
  Tells **openMSX Basic (de)Tokenizer** it is running from a build system (or an external program) and adjust some behaviours accordingly.
  arg: `-frb`

Known bugs

An `autoexec.bas` on the mounted disk will run automatically, taking control of **openMSX** and possibly preventing the conversion.
There is a problem passing file names with some special characters, “&” for instance.

Helper Tools

Some tools made to help develop and test the Tokenizer.

> Their settings can be changed on the code itself.
> The variables are easily named and they are commented when needed.

Token Compare

Convert ASCII listings with **MSXBatoken** and compares them with a conversion from **openMSXBatoken** (using a "real" MSX).

Token Format Viz

Format a tokenized MSX Basic program to display a line of basic per line of tokens.
Also can interleave another tokenized file to compare them line by line.

> Take command line arguments. See code.

Random Numbers

Generate MSX Basic lines with random numbers of several types.
Generate integer, single, double and scientific notation numbers.

Main tokenizers Python 3.8.
Helper tools Python 2.7.
Use with care.