# 1 Default minted “C++” lexer

```
// Saves the current state to a file following the xyz-standard
// (see // http://en.wikipedia.org/wiki/XYZ_file_format)
void IO::saveState(System &system)
{
  if(file.is_open()) {
    file << system.atoms().size() << endl;
    file << endl;
    for(Atom *atom : system.atoms()) {
      vec3 p = atom->position;
      file <<
        "H " << p.x() << " " << p.y() << " " << p.z() << endl;
    }
  }
}
```

# 2 Custom Molecular Dynamics lexer

```
// Saves the current state to a file following the xyz-standard
// (see // http://en.wikipedia.org/wiki/XYZ_file_format)
void IO::saveState(System &system)
{
  if(file.is_open()) {
    file << system.atoms().size() << endl;
    file << endl;
    for(Atom *atom : system.atoms()) {
      vec3 p = atom->position;
      file <<
        "H " << p.x() << " " << p.y() << " " << p.z() << endl;
    }
  }
}
```

## 3 Makefile

```
all: example.pdf

# The Pygments Custom language module uses Environment Variables to
# add keywords for syntax highlighting. All are optional.
# Note that this maps strings to subtypes of Token.Keyword.

export PYGMENTS_CUSTOM_BASE_LEXER=CppLexer

export PYGMENTS_CUSTOM_TYPE        := [ "vec3", "Atom", "System" ]
export PYGMENTS_CUSTOM_CONSTANT    := [  ]
export PYGMENTS_CUSTOM_KEYWORD     := [  ]
export PYGMENTS_CUSTOM_DECLARATION := [  ]
export PYGMENTS_CUSTOM_NAMESPACE   := [  ]
export PYGMENTS_CUSTOM_PSEUDO      := [  ]
export PYGMENTS_CUSTOM_RESERVED    := [  ]
export PYGMENTS_CUSTOM_REMOVED     := [  ]


# Pattern rule for building the pdf from latex and C source files.
# Note 1: £< represents the latex filename.
# Note 2: The date embedded in the pdf is set to the last modification
#         time of the latex file according to the git commit log or
#         the file timestamp.
%.pdf: %.ltx %.cpp FORCE
        SOURCE_DATE_EPOCH=`(git log -1 --format=%at $< 2>/dev/null; \
                            date +%s -r $<) | head -1` \
                pdflatex -shell-escape $<

# Always re-create the pdf at each `make`.
FORCE:

.PHONY: clean
clean:
        rm -rf *.aux *.log *.out.pyg  _minted-* *~
```

# 4 Latex example

```
\documentclass{article}
\usepackage[cache=false]{minted}
\usepackage[strict]{changepage}

\begin{document}
\setminted{linenos=true, frame=single}

\section{Default minted ‘‘C++’’ lexer}
\inputminted{Cpp}{example.cpp}

\section{Custom Molecular Dynamics lexer}
\inputminted{custom}{example.cpp}

\clearpage\section{Makefile}
\setminted{linenos=false, frame=lines}
\inputminted{Makefile}{Makefile}

\clearpage\section{Latex example}
\inputminted[curlyquotes]{tex}{example.ltx}

\clearpage\setminted{linenos=false, frame=lines}
\changepage{4\baselineskip}{4cm}{-2cm}{-2cm}{}{-2cm}{}{}{}
\section{Custom keyword code}
\inputminted{python}{../pygments_custom/__init__.py}

\end{document}
```

## 5 Custom keyword code

```
# pygments_custom
# This module uses environment variables to customize which
# keywords to highlight and which Pygments Lexer to inherit from.

from pygments.token import Name, Keyword
from pygments.lexers import *
from os import getenv
from sys import stderr
import json

# Previously we used "from pygments.lexers import CLexer as mysuper".
# Now, which lexer to inherit from is variable.
base = getenv( "PYGMENTS_CUSTOM_BASE_LEXER" )
if not base:
    base="CLexer"

try:
    mysuper = locals()[ base ]          # Do we have a class named that?
except KeyError:
    mysuper = CLexer
    print( f'\n*** CustomLexer Error: Unknown Lexer: "{base}". ',
           f'Defaulting to CLexer.', file=stderr )
    import pygments.lexers
    print( f'\n*** CustomLexer Error: Please set PYGMENTS_CUSTOM_BASE_LEXER '
           f'to one of {pygments.lexers.__all__}', file=stderr )


class CustomLexer(mysuper):
    """CustomLexer for pygments which extends an existing lexer with
    new keywords. The existing lexer defaults to CLexer but can be
    changed by the environment variable PYGMENTS_CUSTOM_BASE_LEXER.
    For example, one could inherit the C++ Lexer's keywords like so:

            export PYGMENTS_CUSTOM_BASE_LEXER="CppLexer"

    New keywords can be highlighted as Type, Constant, Namespace,
    Declaration, Pseudo, Removed, Reserved, or plain old Keyword. To
    add keywords, set the environment variables PYGMENTS_CUSTOM_TYPE,
    PYGMENTS_CUSTOM_CONSTANT, ..., PYGMENTS_CUSTOM_KEYWORD.

    Each variable is a Python list (square brackets surrounding a
    comma separated list of quoted strings). For example, this
    highlights new types (e.g., classes or typedefs):

            export PYGMENTS_CUSTOM_TYPE="[ 'vec3', 'Atom', 'System' ]"

    """
```

```
    name = 'Custom'
    aliases = ['custom']
    kwtable = [ ('PYGMENTS_CUSTOM_TYPE',        Keyword.Type),
                ('PYGMENTS_CUSTOM_CONSTANT',    Keyword.Constant),
                ('PYGMENTS_CUSTOM_NAMESPACE',   Keyword.Namespace),
                ('PYGMENTS_CUSTOM_DECLARATION', Keyword.Declaration),
                ('PYGMENTS_CUSTOM_PSEUDO',      Keyword.Pseudo),
                ('PYGMENTS_CUSTOM_REMOVED',     Keyword.Removed),
                ('PYGMENTS_CUSTOM_RESERVED',    Keyword.Reserved),
                ('PYGMENTS_CUSTOM_KEYWORD',     Keyword),               ]
    EXTRA = {}

    tr = str.maketrans( "'", '"' )          # transliterate python to JSON strings.
    for v, k in kwtable:
        s = getenv( v )
        if s:
            s = s.translate(tr)
            try:
                EXTRA[k] = json.loads( s )
            except Exception as e:
                print(  f'\n*** Could not parse: {s}',
                         '\n*** Error:', e, file=stderr)

    def get_tokens_unprocessed( self, text, stack=('root',) ):
        for index, token, value in mysuper.get_tokens_unprocessed(self, text, stack):
            if token is Name:
                for key in self.EXTRA:
                    if self.EXTRA[key] and value in self.EXTRA[key]:
                        token=key
                        break

            yield index, token, value


if __name__ == '__main__':
    print( "testing" )
    x = CustomLexer()
    for y in x.get_tokens_unprocessed( "M_PI", "hello_my_name is", "removed" ):
        print(y)
    for y in x.get_tokens_unprocessed( "vec3 x,y,z; System;", "reserved", Type):
        print(y)
```

---