Automatic documentation with protex

From Geos-chem

Protex is a very useful Perl script that can strip information from a standard Fortran document header and save that to a LaTeX file. Protex was developed at the Goddard Space Flight Center by Arlindo Da Silva, Will Sawyer, and others.

We shall use protex to auto-document all ESMF-compatible code created at Harvard. Also, GEOS-Chem v8-01-03 and higher will utilize protex documentation (at least for newly-added code). We therefore recommend GEOS-Chem users to become familiar with protex.

NOTE: In order to use protex, you must have the various LaTeX utilities (e.g. latex, pdflatex, dvipdf, dvips, etc.) installed on your system. These should come standard with most builds of Unix or Linux.

Contents

Downloading protex

Click here to download a tarball file (http://acmg.seas.harvard.edu/geos/wiki_docs/protex/protex.tar.gz) with the `protex` script, template headers, and sample output files.

To install protex, do the following:

```
gunzip protex.tar.gz
tar xvf protex.tar
```

This will install the following files into the protex subdirectory:

README
: A file that describes the contents of the tar file.

protex
: Perl script (by Arlindo da Silva et al) from GSFC that converts special F90 header comment tags into LaTeX format.

f90pdf
: Perl script (by Bob Yantosca) which is a wrapper for protex. Calls `protex` and `pdflatex` to convert the output of `protex` to PDF format.

f90ps
: Perl script (by Bob Yantsoca) which is a wrapper for `protex`. Calls `protex`, `latex`, and `dvips` to convert the output of `protex` to PostScript format.

template_introduction.txt
: Front page template file for use with the protex script. Use this template to specify the title of the document, authors, affiliation, and date. This should always be the first file passed to `protex`.

template_includefile.h
: Template header file with sample F90 declarations

template_module.F90
: Template F90 module with module header and internal routines.

template_routine.F90
: Template F90 file for a standalone program or routine (i.e. not contained in a module).

sample.pdf
: Sample PDF output file, created from the `template_includefile.h`, `template_routine.F90` and `template_module.F90` files.

sample.ps
: Sample PostScript output file, created from the `template_includefile.h`, `template_routine.F90` and `template_module.F90` files.

If you have root privilege on your system, you can then copy the `protex` file to `/usr/local/bin` or whichever directory contains system-wide executables. Otherwise, you can install it to a directory in your own space.

Generating PDF documentation

To generate a PDF file, type:

```
 f90pdf [list of files]
```

This will create the files:

output.tex
: Documentation file in LaTeX format file

output.pdf
: Documentation file in PDF format

You may choose to rename these as you see fit. `f90pdf` is a convenience wrapper script. It executes the following commands:

```
protex -s -f [list of files] > output.tex
pdflatex output.tex
pdflatex output.tex
pdflatex output.tex
rm *.dvi *.aux *.log *.toc
```

The `pdflatex` command is a useful shorthand. Using `pdflatex` replaces separate calls to `latex` and `dvipdf`. For example, the above commands are equivalent to:

```
protex -s -f [list of files] > output.tex
latex output.tex
latex output.tex
latex output.tex
dvipdf output.tex output.pdf
rm *.dvi *.aux *.log *.toc
```

NOTES:

1. The -s option of `protex` will cause only the module, subroutine, and function headers to be included in the output.
2. The -f option of `protex` suppresses printing source code file information next to each module or routine name.
3. Calling `pdflatex` 3 times is necessary to ensure that the table of contents information will be compiled correctly into the final output file.
4. If the `f90pdf` script hangs for a while then it has probably encountered an error. To continue you may type "r" or exit with "x".

Gnerating PostScript documentation

To generate a PostScript file, type:

```
f90ps [list of files]
```

This will create the files:

output.tex
: Documentation file in LaTeX format file

output.ps
: Documentation file in PDF format

You may choose to rename these as you see fit. Like `f90pdf`, `f90ps` is a convenience

wrapper script. It executes the following commands:

```
protex -s -f [list of files] > output.tex
latex output.tex
latex output.tex
latex output.tex
dvips output.dvi -o output.ps
rm *.dvi *.aux *.log *.toc
```

NOTES:

1. The -s option of `protex` will cause only the module, subroutine, and function headers to be included in the output.
2. The -f option of `protex` suppresses printing source code file information next to each module or routine name.
3. Note: calling latex 3 times is necessary to ensure that the table of contents information will be compiled correctly into the final output file.
4. Also note: since there is no equivalent "pslatex" command, we must do the intermediate step of creating the dvi file and then calling `dvips` to create the PostScript file.
5. If the `f90ps` script hangs for a while then it has probably encountered an error. To continue you may type "r".

Generating HTML documentation

To generate HTML documentation from F90 files, type:

```
protex -s -f [list of files] > output.tex
latex2html output.tex
```

This will call the latex2html utility (which ships standard with Linux) to parse the LaTeX file "output.tex" and create navigatable HTML pages.

NOTES:

1. The -s option of `protex` will cause only the module, subroutine, and function headers to be included in the output.
2. The -f option of `protex` suppresses printing source code file information next to each module name.

Sample output files

We have created sample output files for you (`sample.pdf`, `sample.ps`) from the F90 template files located in this directory. These were created with the following commands:

```
f90pdf template_introduction.txt *.h *.F90
mv output.pdf sample.pdf

f90ps template_introduction.txt *.h *.F90
```

```
 mv output.ps sample.ps
```

Note that by placing the `template_introduction.txt` file first before any Fortran files (*.h, *.F90), this will cause a title page to be added to the output files. You may omit this if you wish.

Protex header files in more depth

template_introduction.txt

You can use the template_introduction.txt to define a title page for your document, such as:

```
!------------------------------------------------------------------------------
!          Harvard University Atmospheric Chemistry Modeling Group            !
!------------------------------------------------------------------------------
!BOI
! !TITLE: Front page template
! !AUTHORS: Bob Yantosca and Philippe Le Sager
! !AFFILIATION: School of Engineering and Applied Sciences, Harvard University
! !DATE: May 23, 2008
!EOI
!------------------------------------------------------------------------------
```

NOTES:

1. The `BOI` and `EOI` are used to define extent of the header. Protex will only look at the text between `BOI` and `EOI`.
2. Text following `!TITLE`, `!AUTHORS`, `!AFFILIATION`, and `!DATE` tags must be placed on the same line.
3. You may also specify an optional `!INTRODUCTION` tag following `!DATE`, which will become topic #1 in the table of contents. This allows you to go back later and manually insert LaTeX markup later into the output.tex file.

template_includefile.h

You can use the `template_includefile.h` to document an include file with either common blocks or defined parameters, such as:

```
!------------------------------------------------------------------------------
!          Harvard University Atmospheric Chemistry Modeling Group            !
!------------------------------------------------------------------------------
!BOP
!
! !INCLUDE: GC_SomethingIncludeFile.h
!
! !DESCRIPTION: This include file contains the various parameters that will
!   allow the module and routine to do stuff to various things in various
!   routines in various places.
!\\
!\\
! !PUBLIC TYPES:
!
  TYPE t_GeosChemSomething
```

```
    !%%% declare stuff here %%%
  END TYPE t_GeosChemSomething
!
! !DEFINED PARAMETERS:
!
  INTEGER(ESMF_KIND_I8), PUBLIC, PARAMETER :: myIntParam   ! INTEGER value
  REAL(ESMF_KIND_I8),    PUBLIC, PARAMETER :: myRealParam  ! REAL*8 value
!
! !REVISION HISTORY:
!  21 May 2008 - R. Yantosca - Initial Version
!
! !REMARKS:
!EOP
!------------------------------------------------------------------------
```

NOTES:

1. The `BOP` and `EOP` tags denote the beginning and end of the protex prologue. Protex will search for text between these two tags.
2. You must place the name of the include file on the same line as the `!INCLUDE:` tag.
3. The text following the `!DESCRIPTION` tag must also start on the same line. This text will be rendered as formatted text instead of in fixed-space courier font. This means that:
   - Special LaTeX characters (e.g. _ and #) must be prefaced with a backslash (e.g. `\_` and `\#`) or else this will cause LaTeX to choke when parsing the file.
   - Two manual line breaks `!\\` must be placed at the end of the text to prevent the next tag from starting on the same line.
4. Except for `!MODULE` and `!DESCRIPTION`, for the rest of the protex tags it is OK to start text on a new line below the tag. You can place extra Fortran comments `!` after protex tags to make the code more readable.
5. You should list the actual program unit definition (in this case, the `MODULE GC_SomethingMod` statement) on a new line following the `!INTERFACE` tag.
6. References to journal articles or other notes should be placed after the `!REMARKS tag`.

template_module.F90

This file contains headers for declaring a Fortran 90 module, as well for the internal module routines.

Header for top-of-module

Here is the protex header that goes at the very top of a Fortran 90 module:

```
!------------------------------------------------------------------------
!          Harvard University Atmospheric Chemistry Modeling Group            !
!------------------------------------------------------------------------
!BOP
!
! !MODULE: GC_SomethingMod.F90
!
! !DESCRIPTION: This module contains the data type to declare a Something
!   object and the methods to work with the Something object.
!\\
!\\
! !INTERFACE:
```

```
!
MODULE GC_SomethingMod
!
! !USES:
!
  USE ESMF_Mod
  IMPLICIT NONE
!
! !PUBLIC TYPES:
!
  TYPE t_GeosChemSomething
     !... declare stuff here
  END TYPE t_GeosChemSomething
!
! !PUBLIC MEMBER FUNCTIONS:
!
  PUBLIC :: GC_SomethingRoutine1
  PUBLIC :: GC_SomethingFunction1
!
! !PUBLIC DATA MEMBERS:
!
  INTEGER(ESMF_KIND_I4), PUBLIC :: myPublicVariable  ! public data variable
!
! !REVISION HISTORY:
!  21 May 2008 - R. Yantosca - Initial Version
!
! !REMARKS:
! Protex is great!
!
!EOP
!------------------------------------------------------------------------
```

The tags are mostly self-explanatory. However, you must be aware of a few things:

1. The `!BOP` and `!EOP` tags denote the beginning and end of the protex prologue. Protex will search for text between these two tags.
2. You must place the name of the module on the same line as the `!MODULE` tag.
3. The text following the `!DESCRIPTION` tag must also start on the same line. This text will be rendered as formatted text instead of in fixed-space courier font. This means that:
   - Special LaTeX characters (e.g. _ and #) must be prefaced with a backslash (e.g. `\_` and `\#`) or else this will cause LaTeX to choke when parsing the file.
   - Two manual line breaks `!\\` must be placed at the end of the text to prevent the next tag from starting on the same line.
4. Except for `!MODULE` and `!DESCRIPTION`, for the rest of the protex tags it is OK to start text on a new line below the tag. You can place extra Fortran comments `!` after protex tags to make the code more readable.
5. You should list the actual program unit definition (in this case, the `MODULE GC_SomethingMod` statement) on a new line following the `!INTERFACE` tag.
6. References to journal articles or other notes should be placed after the `!REMARKS tag`.

Header for module procedures

Here is the protex header that should be placed at the top of each subroutine that is contained in the module:

```
!------------------------------------------------------------------------
```

```
!          Harvard University Atmospheric Chemistry Modeling Group            !
!-----------------------------------------------------------------------------
!BOP
!
! !IROUTINE:  GC_SomethingRoutine1
!
! !DESCRIPTION: This routine does something to the input variable and returns
! the result in the output variable.
!\\
!\\
! !INTERFACE:
!
  SUBROUTINE GC_SomethingRoutine1( input, inpout, output, status )
!
! !INPUT PARAMETERS:
!
    INTEGER(ESMF_KIND_I4), INTENT(IN) :: input   ! Input variable
!
! !INPUT/OUTPUT PARAMETERS:
!
    INTEGER(ESMF_KIND_I4), INTENT(IN) :: inpout  ! In/out variable
!
! !OUTPUT PARAMETERS:
!
    INTEGER(ESMF_KIND_I4), INTENT(IN) :: output  ! Output variable
!
! !REVISION HISTORY:
!  21 May 2008 - R. Yantosca - Initial Version
!
! !REMARKS:
! Protex is great!
!
!EOP
!-----------------------------------------------------------------------------
!BOC
   !%%% Your code goes here %%%
 END SUBROUTINE GC_SomethingRoutine1
!EOC
```

and here is the corresponding header for a function that is contained in the module:

```

!-----------------------------------------------------------------------------
!          Harvard University Atmospheric Chemistry Modeling Group            !
!-----------------------------------------------------------------------------
!BOP
!
! !IROUTINE:  GC_SomethingFunction1
!
! !DESCRIPTION: This function does something to the input variable and returns
! the result in the value variable.
!\\
!\\
! !INTERFACE:
!
  FUNCTION GC_SomethingFunction1( input ) RESULT( value )
!
! !INPUT PARAMETERS:
!
   INTEGER(ESMF_KIND_I4), INTENT(IN) :: input   ! Input variable
!
! !OUTPUT PARAMETERS:
!
   INTEGER(ESMF_KIND_I4)             :: value   ! Function value
!
! !REVISION HISTORY:
!  21 May 2008 - R. Yantosca - Initial Version
!
! !REMARKS:
```

```
! Protex is great!
!
!EOP
!-----------------------------------------------------------------------
!BOC
   !%%% Your code goes here! %%%
 END FUNCTION GC_SomethingFunction1
!EOC
```

NOTES:

1. Here again, the tags `BOP` and `EOP` denote the extent of the protex prologue.
2. The tags `BOC` and `EOC` denote the beginning and end of the source code. If you call protex with the `-s` option (or if you use the `f90pdf` and `f90ps` scripts), then protex will ignore the source code and only render the text from the prologues at the top of each routine.
3. For routines that are contained by a module, we use the `!IROUTINE` tag. The name of the routine must be placed on the same line as `!IROUTINE`.

template_routine.F90

Header for standalone routines

The protex header for standalone routines (i.e. main programs, or subroutines and functions not contained within a module) is slightly different than the header for module procedures:

```
!-----------------------------------------------------------------------
!BOP
!
! !ROUTINE: GC_Routine.F90
!
! !DESCRIPTION: This routine takes in an input variable, does something to it,
!  and then sends out an output variable.
!\\
!\\
! !INTERFACE:
!
SUBROUTINE GC_Routine( input, output )
!
! !USES:
!
  USE GC_SomethingMod
!
! !INPUT PARAMETERS:
!
  REAL(ESMF_KIND_R8), INTENT(IN) :: input    ! input variable
!
! !OUTPUT PARAMETERS:
!
  REAL(ESMF_KIND_R8), INTENT(IN) :: output   ! output variable
!
! !BUGS:
! None known at this time
!
! !SEE ALSO:
! GC_SomethingMod.F90
!
! !SYSTEM ROUTINES:
! None
```

```
!
! !FILES USED:
! GC_SomethingMod.F90
!
! !REVISION HISTORY:
! 21 May 2008 - R. Yantosca - Initial version
!
! !REMARKS:
! Protex is great!
!
!EOP
!------------------------------------------------------------------------
!BOC
  !%%% Your code goes here! %%%
END SUBROUTINE GC_Routine
!EOC
```

NOTES:

1. For standalone routines, we use the `!ROUTINE` tag as opposed to the `!IROUTINE` tag (which is reserved for routines within modules).

--Bob Y. 10:36, 28 May 2008 (EDT)

Useful LaTeX formatting commands

Here we include the LaTeX markup text commands for various useful constructs (lists, sections, etc). NOTE: These are only applicable for text that you place under the `!DESCRIPTION` section in the Protex header. For more information, please see Jeff Clark's online LaTeX tutorial (http://frodo.elon.edu/tutorial/tutorial/) .

Itemized bullet list

```
\begin{itemize}
\item This is Item 1
\item This is Item 2
\item This is Item 3
\end{itemize}
```

Numbered bullet list

```
\begin{enumerate}
\item This is Item 1
\item This is Item 2
\item This is Item 3
\end{enumerate}
```

Description list

```
\begin{description}
\item[Item Name 1] Description 1
\item[Item name 2] Description 2
\item[Item name 3] Description 3
\end{description}
```

Text formatting

```
\textbf{This text will be rendered as Boldface.}

\emph{This text will be rendered as Italics.}

\underline{This text will be rendered as Underline.}

\texttt{This text will be rendered as "teletype" text (i.e. fixed-space Courier font)}
```

Math mode

```
This text contains $i-1$ the term i-1 rendered into math mode.

This text contains $A_{i-1}$ A with the "i-1" subscripted.

This text contains $A^{i-1)$ A with the "i-1" superscripted

$$A_{i-1} = A_{i} - B_{i}$$  (This text describes an equation on its own line, with subscripts)
```

--Bob Y. 11:50, 15 April 2009 (EDT)

Retrieved from "http://wiki.seas.harvard.edu/geos-chem/index.php/Automatic_documentation_with_protex"

- This page was last modified 14:03, 7 July 2009.