CORONAL DIAGNOSTIC SPECTROMETER
SoHO

CDS SOFTWARE NOTE No. 16

Version 1.0 21 May 1994

C Utility Programs for IDL Programming

S. V. H. Haugan
Institute of Theoretical Astrophysics
University of Oslo
PO Box 1029 Blindern
0315 Oslo
Norway

s.v.h.haugan@astro.uio.no

1 Introduction

This document is a description of four programs that are designed to ease some of the routine tasks encountered when working with programming projects.

1.1 Visual Appearance

During development of programs, the syntactical structure (grouping of statements etc) is changed quite often. Maintaining a coherent visual look that represents the structure of the code is often a tedious task, but it improves the readability and may also be a tool for spotting logical errors in the code. This task is handled by the program **Nice** .

1.2 Naming Conflicts

In large projects involving a large number of routines and files, it is not always easy to avoid using the same name for two different procedures, resulting in a naming conflict that may give spurious results, depending on the organization of the search path, and also depending on the order in which routines are compiled. This is especially difficult to detect for program files that contain more than one procedure declaration, making a check of file name uniqueness insufficient. The program **Defines** detects and reports all naming conflicts in a selected set of programs.

1.3 Documentation of Dependencies

A list of the routines that are called by a program file is included as a part of the documentation for each routine. For large routines the manual construction and maintenance of this list is subject to errors by the programmer overlooking calls. By using **Defines** to make a list of all significant routines (ie., standard IDL library routines may be excluded), the program **Calls** can be used to produce a list of those routines that are also referenced in a specific program.

1.4 Impact of Modifications

Sometimes, modifications of a routine makes it necessary to alter other routines calling it. In order to get a full list of those programs that may be in need of updating, **Callers** goes through all routines and reports those that reference the altered routine.

2 Where to find the programs

The source code is placed in the directory $CDS_C_PROG at cds2 (later at Goddard). This directory contains the C code for the 4 different utilities, contained in eight source files, and a makefile that may be used to compile some or all of the utilities. In order to compile all three just use the command "make" in the directory with all the files.

3 Description of the Programs

3.1 Nice – Block Indentation and Capitalization

Use:

```
unix> Nice idlprog.pro
```

This parses the program and indents the lines according to block level. Current rules:

- PRO ... END and FUNCTION ... END blocks: 2 characters.
- BEGIN ... END and CASE ... ENDCASE blocks: 4 characters.

The indentation is done with tabs where possible. Series of blank spaces (to align comments etc) are also tabified. Future versions may introduce user-defined indentation rules.

Continued ($) lines/statements are always indented precisely as much as the beginning of the statements, so that alignments are preserved, e.g.:

```
begin                  begin
test,x,y,$                 test,x,y,$
     a,b,$      -->             a,b,$
       c,d                        c,d
test2,x,y                  test2,x,y
end                    end
```

In addition, the program can be used to enforce lower/upper case policies, by the use of a .Nicerc file either in the working directory, or in the home directory. For the CDS project, the directory $CDS_C_PROG is also searched. The .Nicerc file has one line for each word that has a special policy. Words in the source program with the same spelling are converted into the form that occurs in the .Nicerc file. A file with the following entries:

```
BEGIN
END
Test
TEst2
```

would produce the following (from the above example):

```
begin                  BEGIN
test,x,y,$                 Test,x,y,$
     a,b,$      -->             a,b,$
       c,d                        c,d
test2,x,y                  TEst2,x,y
end                    END
```

The program does not do any changes to words that have no explicit policy in the .Nicerc file. Also, comments are not parsed word by word, so no changes there either.

3.2 Defines – Defined Names and Naming Conflicts

To find out what procedures and functions an IDL file defines, use:

```
unix> Defines idlfile.pro
```

The output is a list of capitalized names of all routines that are defined in the file, with parentheses appended to signify functions instead of procedures.

The program may also be used on all files in an IDL path, with the same syntax as the IDL_PATH specification:

```
unix> Defines +.
```

This will produce a list of all routines defined in all the *.pro files in the current directory and all subdirectories.

If you have a list of file names in a file (filelist.txt), you can ask Defines to search through these with the command

```
unix> Defines @filelist.txt
```

To get a list of all routines available for a given IDL_PATH, written to the file "list", write:

```
unix> Defines $IDL_PATH > list
```

The specification of where Defines should look may also be mixed, ie.:

```
unix> Defines $IDL_PATH @filelist
```

The program may also be used to report multiple definitions of procedures/functions, by using:

```
unix> Defines - $IDL_PATH
```

The program will produce three or more lines of output for each multiple definition: one line with the name of the routine, and one line for each of the files with a definition of it.

3.3 Calls – Documentation of External Calls

In order to find the functions/procedures that an IDL program file calls externally, it needs a file produced by **Defines** as a definition of what external procedures to care about. Use:

```
unix> Defines $IDL_PATH > definedfiles
unix> Calls definedfiles idlfile.pro
```

The output is a list of the routines defined in $IDL_PATH that are called from idlfile.pro.

The parsing done by this program is very much "dumb". The output is not always correct and needs to be cross-checked manually, as for example the use of a variable "X" in idlfile.pro will result in a report that the file refers to the *routine* X if such a routine exists in the IDL path.

3.4 Callers – Programs Referring to a Routine

Used to find those IDL program files that reference a specified function/procedure. Supply the name of the function, plus a path, as for Defines:

```
unix> Callers detdata $IDL_PATH
```

Will report all routines in the IDL_PATH that contain the symbol detdata. Upper and lower case is not significant.