**Idaho
National
Engineering
Laboratory**

*Managed
by the U.S.
Department
of Energy*

EGG-CATT-9198

INFORMAL REPORT

**CROSS REFERENCE ANALYSIS OF FORTRAN
(CRAFT)**

**Neldon H. Marshall
Earl S. Marwil**

*Work performed under
DOE Contract
No. DE-AC07-76ID01570*

Received by OSTI
NOV 28 1991

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

This document contains new concepts or the author(s) interpretation of new calculations and/or measurements; accordingly, EG&G Idaho, Inc. is required by the United States Government to include the following disclaimer:

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

EGG-CATT--9198

DE92 003312

CROSS REFERENCE ANALYSIS OF FORTRAN
(CRAFT)

Neldon H. Marshall
Earl S. Marwil

July 1991

Scientific Computing Unit
Computer Applications and Technology Transfer Group
Information Resources Department
EG&G Idaho, Inc.
Idaho Falls, Idaho 83415

Prepared for the
U. S. Department of Energy
Idaho Operations Office
Under DOE Contract No. DE-AC07-76ID01570

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

ABSTRACT

Cross Reference Analysis of Fortran (*Craft*) is a set of programs which extract cross reference data from listing files generated by the Cray Research *cft77* Fortran compiler. The data is separated into a number of files by a preprocessor written in C. These data files are sorted and then post processed by a Fortran program to generate various summary reports as requested by the user.

ACKNOWLEDGEMENTS

The authors wish to thank Brent Stacey, Scott Matthews, and Mike Olsen for their comments and suggestions on earlier drafts of this document.

CROSS REFERENCE ANALYSIS OF FORTRAN (CRAFT)

1. INTRODUCTION

Cross reference analysis of Fortran programs can be a tedious process. Fortran compilers generally process a single module at a time and, when requested, provide a cross reference listing for each module. The cross reference listing gives detailed information on how and where the various entity names are used within a given module.

Individually, the module listings are valuable for local entity usage within that module. Collectively, the cross reference listings from all the modules of a program contain information on global entity usage which can be critical to the software engineer developing or maintaining the program.

The Cray Research *cft77* compiler lists entity Name, Location, Type, Usage, Storage and Source Program References. Along with the Unicos operating system, Cray provides a utility, *ftref*, which reads the listing file and creates a report. The report generated by *ftref* optionally contains lists of COMMON blocks and global entities (in COMMON blocks) used in the modules of a program, lists of entry names, parent and child routines and common blocks for each routine, and a call tree. The *ftref* utility, however, provides so much information that for large programs it is still a time-consuming process to analyze.

One of the authors, Neldon Marshall, prototyped a software tool, *scan*, to summarize the report generated by the *ftref* utility. The *scan* program identifies:

1. Entities in COMMON not named consistently,
2. COMMON blocks used by a single module,
3. Entities in COMMON which are never accessed,
4. Entities in COMMON which are used but never defined,
5. Entities in COMMON which are used as actual arguments but never explicitly defined, and
6. Modules which never access a particular COMMON block.

The capability to identify these conditions influenced the design of *Craft*. Reports are available to assist the software engineer in analyzing the functional relationships of the modules, the global data used in the program, and local entities.

In section 2, we present the notation used in subsequent sections. In section 3, we discuss the function cross reference report. In section 4, we describe the global entity (COMMON blocks and entities in COMMON) cross reference report. Then, in section 5 we discuss similar reports for local entities, including constants, variables and arrays. Section 6 contains a discussion of the implementation.

Appendix A contains a brief user's guide for *Craft*. Appendix B contains the examples of the *Craft* reports cited in the text.

2. NOTATION

Abbreviations in the table format reports are used to indicate the type, usage, and references for local and global entities. This section provides a quick reference for the meaning of those abbreviations.

2.1 Type Codes

These are the entity types allowed in *cft77*.

Code	<i>cft77</i> Type	Description
nt		no type
64	Int64	integer, 64 bit precision
46	Int46	integer, 46 bit precision
rp	Real	single precision real
dp	Double	double precision real
cx	Complex	complex
ch	Char	character
lg	Logical	logical
pt	Pointer	pointer

2.2 Usage Codes

These are the usage codes of interest in the *Craft* reports. The first set applies to variables and arrays. The second set applies to functions.

Variables and Arrays		
Code	<i>cft77</i> Usage	Description
a	Array	array
ae	Array,Eqv	equivalenced array
v	Var	variable
ve	Var,Eqv	equivalence variable
Functions		
Code	<i>cft77</i> Usage	Description
ex	External	external program
in	Intrinsic	intrinsic function
sf	Stmt_Func	statement function

2.3 Reference Codes

Variables, Arrays, and Functions can be referenced in four different ways. These are identified in the following table.

Code	Description
d	declared
m	modified
a	actual argument
u	used

3. CROSS REFERENCE ANALYSIS OF FUNCTIONS

3.1 Alternate Entry Point Summary

The Fortran analysis tools which we have used in the past, including *ftref*, do not identify the alternate entry points of Fortran subroutines or functions. One of the features of *Craft* is to identify modules with multiple entry points and to identify all the entry points in each such module. The primary entry point is the defining reference for the module, either in the subroutine statement or the function statement.

Craft reports the alternate entry points in a tabular representation, as a list, or as exceptions. A message is printed if there are no alternate entry points within the set of modules analyzed.

The table format shows alternate entry point names as columns across the top of the table and principal entry points as rows at the left of the table. The first column indicates the defined type of the principal entry point, while the type of the alternate entry point is listed in the row corresponding to the module in which it occurs. Inconsistent typing of principal and alternate entry points is noted as a warning.

In the list format, the primary entry point is the module name and the alternate entry points are listed on subsequent lines. Any type conflicts are noted on the exception report. Example 1 illustrates and discusses the alternate entry point reports.

3.2 Analysis of Functions

Craft can report on function use in a table, as a list, or as exceptions.

Two tables are presented for function use within a program. Both use a two-row format. The first indicates type and use of functions and the second shows the program references.

Functions include externals, intrinsics and statement functions. In the type and use table the defined type of external functions which are part of the program being analyzed is listed in the first column. For externals which are not part of the program under analysis the type is unknown and this is indicated by a '?' in the first column. Type and use conflicts are noted separately.

Craft reports functions that are declared but otherwise unused, so that the declarations may be removed from such a module.

In the list format, each module name is listed and parent and child functions are listed on subsequent lines. Example 2 illustrates the function use report.

A list of entry points not used within the particular program is printed at the end of the exception report. This includes the main program(s), block data module(s), and any other entry points not called. This can reveal artifacts from changes during development or modifications during maintenance.

3.3 Call Tree

Although other tools generate call trees, none that we know will show the intrinsic functions used in a program. Example 3 shows the call tree report from *Craft*. A feature is available to request the call tree for any selected module in the tree.

4. CROSS REFERENCE ANALYSIS OF GLOBAL ENTITIES

4.1 COMMON Block Summary

In analyzing COMMON blocks, a software engineer needs to know what COMMON blocks are used in a particular module and in which modules a particular COMMON block occurs.

In the *Craft* table, both views of the data are available at a glance. The COMMON Block Summary lists all modules (primary entry point name) as columns across the top of the table and COMMON block names as rows at the left of the table. In the 1977 Fortran standard, numeric (Integer, Real, Logical, Complex, and Double Precision) data and character data cannot be mixed within the same COMMON block. The type of each COMMON block is indicated for each module in which it is referenced. If all entities in COMMON are character, the COMMON block is typed character. If all entities in COMMON are numeric, the COMMON block is typed numeric.

If a COMMON block is not used with consistent type it is noted as having a type conflict. If a COMMON block is used in a single module, it is also noted.

In the list format, each module name is listed with the COMMON blocks that are used within the module. In addition, each COMMON block is listed with all modules in which it is referenced. Example 4 illustrates the Common Block Summary report.

4.2 COMMON Entity Analysis

For each COMMON Block two tables are presented. Modules in which the COMMON block is referenced are listed as columns of the table. Along the rows of the table entities are listed by relative location within the block and name. Example 5 shows the Common Entity Summary report.

The first table uses two rows per entity to show entity type and use within each module. Type and use conflicts are indicated in the left-hand margin for each entity. For entities equivalenced, either explicitly or implicitly, to a given relative location within a COMMON block, type and use conflicts are further noted for each module and in the SUMMARY column for the location. If any of these conflicts exist they are flagged to the left of the location and/or entity names.

The second table uses two rows per entity to show how each entity is referenced. For each module a two-by-two array indicates whether the entity is declared (d), modified (m), used as an actual argument (a), or simply used (u) within that module.

If the character is present, that kind of reference occurs at least once; otherwise it is blank. A count of the number of times each kind of reference occurs is not given. Reference data is summarized by entity and location.

The following conditions are flagged:

1. Entities undefined and unused (none of m, a or u),
2. Entities defined but unused (m, but no a or u),
3. Entities possibly undefined (a, but no m or u), or
4. Entities used but undefined (u, but no a or m).

Further, if no entities in a block are used by a module, that column is flagged. Also, if an entity in COMMON is modified by a function subprogram, it is flagged and a warning issued, since this programming practice may possibly generate unexpected side effects.

5. CROSS REFERENCE ANALYSIS OF LOCAL ENTITIES

5.1 Local Constants

In Fortran, constants are set in parameter statements. Two tables are generated for parameters. The first shows the type summary across all modules using parameters. A warning is issued if an entity is used with different types in different modules. An informative message is issued when a constant entity is used with different values in different files. (The *.VAL file may be consulted to determine where the inconsistency occurs.)

The second report indicates the parameter references. If a constant is declared but otherwise unused (no a or u references) an informative message is issued and the entity and module are flagged. Example 6 shows the Local Constants report.

5.2 Local Variables and Arrays

Craft reports on the local entity use in one format with three options for content. Either full, partial, or exception reports may be selected. The full content option lists all local entities. The partial content option lists all dummy arguments and all exceptions. The exception content lists only exceptions. Messages are highlighted for entities which are used but not defined, for those defined but not used, and for declaration conflicts for locally equivalenced entities. Actual arguments are identified as being used for input only, output only, passed through, used for input and output, or not used.

Example 7 illustrates the Local Entity report.

6. IMPLEMENTATION

There are five steps to generating the cross reference analysis reports on the Cray under Unicos.

1. The source code must be compiled with *cft77 -ex program.f* to generate the listing file *program.l* with the common and external cross reference information. The source code may be included in the listing with the *-es* option.
2. The stream editor, *sed*, is used to select the pertinent information from the listing files.
3. The extracted information is piped to a C program, *xpre*, which separates the data into the 8 files.

File Suffix	Content
*.ALT	alternate entry points
*.BLK	COMMON blocks
*.COM	entities in COMMON
*.ENT	primary entry points
*.FCN	function references
*.LCL	local entities
*.PAR	local constants (parameters)
*.VAL	parameter values

4. The data on each file is sorted with the system sort utility.
5. A Fortran program, *xref*, reads the sorted files and generates the cross reference analysis reports requested through the option flags. See Appendix A for use details.

All of these operations are managed by a C-shell script named *craft*.

7. REFERENCES

- [1] CFT77 Reference Manual, SR-0018 C, Cray Research, Inc., October 1988.
- [2] UNICOS User Commands Reference Manual, SR-2011 5.0, Cray Research, Inc., March, 1989.

APPENDIX A - USER'S GUIDE

The following summarizes the analysis reports available from *Craft*.

Analysis Type	Option	Format Options		
		Table	List	Exception
Alternate Entry	-a	t	l	x
Common Blocks	-b	t	l	x
Functional	-f	t	l	x
Global Entities	-g	t		
Parameters	-p	t		x

For local entities only one format is available. The volume of information can be selected from the following choices.

Analysis Type	Option	Content Option		
		Full	Partial	Exception
Local Entities	-p	f	p	x

The call tree analysis options are summarized in the following table.

Call Tree Analysis	Option	Qualifier
Functional	-f	r
Root of tree	-r	name
Depth of tree	-d	nn (number)

The user's path on the Cray must include `~esm/bin` to access the *craft* script, the preprocessor, *xpre*, and the report writer, *xref*.

Suppose the combined listing file is named *xyz*. To run the *craft* tool, requesting all possible reports except the call tree, enter the command:

```
craft -atlx -blx -flx -gt -ptx -lp xyz > report_file
```

Alternatively, the data files can be created by entering the command:

```
craft xyz
```

and then creating reports with the commands:

```
xref -atlx -blx -flx -gt -ptx -lp xyz > report_file xref -fr xyz > call_tree
```

APPENDIX B - EXAMPLES

Example 1

In this example, three modules each have one alternate entry point. These are all subroutine modules in Fortran since the type is listed in the table as *nt* for each primary and alternate entry point.

The list format summary identifies each module and the alternate entry points. The alternate entry points are cross referenced to the primary entry point or module name.

The exception report shows no exceptions for this example

The reports in this example resulted from using the *-atlx* option and format options.

Alternate Entry Summary

		c		r		r		
		D		r		e		o
		E		e		e		x
		F		d		d		y
		I						
		N						
		E						
		D						
crit				nt		nt		
rite				nt		nt		
roxx				nt		nt		

Alternate Entry Summary

Module Name: crit
Entry Points: cred

Module Name: rite
Entry Points: reed

Module Name: roxx
Entry Points: roxy

Alternate Entry Point: cred
Alternate Entry Point: reed
Alternate Entry Point: roxy

occurs in Module: crit
occurs in Module: rite
occurs in Module: roxx

Alternate Entry Summary

No exceptions to report

Example 2

This example illustrates the function report in table and list formats.

The first table shows the type and usage of each function in the program along with the intrinsic functions. Since *clock*, *date*, *exp*, and *float* are not defined within the modules being analyzed, the type definition is unknown. The second table shows the function references.

The list format identifies the parent and child modules for each primary entry point or module name.

The reports in this example resulted from using the *-fil* option and format options.

Function Type and Use Summary

[illegible]

Function Reference Summary

[illegible]

Function Summary

Entry Name: cal	float	psl	recom	tbl
Calls: cal2				
Called by: cogap				
Entry Name: cal2				
Calls: exp				
Called by: cal				
Entry Name: cogap				
Calls: cal				
Called by: --none--				
Entry Name: head				
Calls: timdat				
Called by: read	write			
Entry Name: init				
Calls: psl				
Called by: cogap				
Entry Name: psl				
Calls: exp				
Called by: cal	init			
Entry Name: read				
Calls: head				
Called by: cogap				
Entry Name: recom				
Calls: --none--				
Called by: cal				
Entry Name: tbl				
Calls: --none--				
Called by: cal				
Entry Name: timdat				
Calls: clock				date
Called by: head				
Entry Name: write				
Calls: head				
Called by: cogap				

Example 3

The call tree for the previous example shows the hierarchy of the modules in the program. Note that the intrinsic functions are included in the tree.

The reports in this example resulted from using the *-fr* option and format option.

Static Call Tree Rooted at cogap

```
< 0> cogap
  < 1> cal
    < 2> cal2
      < 3> exp
        < 2> float
          < 2> psl
            < 3> exp
              < 2> recom
                < 2> tbl
                  < 1> init
                    < 2> psl
                      < 3> exp
                        < 1> read
                          < 2> head
                            < 3> timdat
                              < 4> clock
                                < 4> date
                                  < 1> write
                                    < 2> head
                                      < 3> timdat
                                        < 4> clock
                                          < 4> date
```

Example 4

The common block summary table shows at a glance which modules use a particular common block and which common blocks are used in a particular module. This is very effective for small to moderate sized programs. For large programs this table can span multiple pages. In this case the list format provides the same information.

The reports in this example resulted from using the *-btl* option and format option.

c c c h i p r w
a a o e n s e e r
l l g a i l a c i
2 a d t i d o t
p m e

B-12

COMMON Block Summary

Entry Name: cal	call	calx	chnl	cont	mat	mis	misl	recl	vent	vol
Uses: cal1										
Entry Name: cal2	call	calx	chnl	mis	misl					
Uses: cal1										
Entry Name: cogap	call	chnl	cont	mis	misl					
Uses: cal1										
Entry Name: head	chnl									
Uses: chnl										
Entry Name: init	call	calx	cont	mis	misl	vol				
Uses: cal1										
Entry Name: psl	chnl									
Uses: chnl										
Entry Name: read	call	chnl	cont	mat	mis	misl	rec	vent	vol	
Uses: cal1										
Entry Name: recom	call	chnl	mis	misl	rec	recl	vol			
Uses: cal1										
Entry Name: tbl	--none--									
Uses: --none--										
Entry Name: timdat	--none--									
Uses: --none--										
Entry Name: write	call	chnl	cont	mis	vol					
Uses: cal1										

COMMON Block Summary

Block Name: call	cal2	cogap	init	read	recom	write
Used by: cal						
Block Name: calx	cal2	init				
Used by: cal						
Block Name: chnl	cal2	cogap	head	psl	read	recom write
Used by: cal						
Block Name: cont	cogap	init	read	write		
Used by: cal						
Block Name: mat	read					
Used by: cal						
Block Name: mis	cal2	cogap	init	read	recom	write
Used by: cal						
Block Name: mis1	cal2	cogap	init	read	recom	
Used by: cal						
Block Name: rec	recom					
Used by: read						
Block Name: rec1	recom					
Used by: cal						
Block Name: vent	read					
Used by: cal						
Block Name: vol	init	read	recom	write		
Used by: cal						

Example 5

The table report format is the only format available for entities in COMMON blocks. In this example, two different names are implicitly equivalenced at the position 0, relative to the start of the block. The name *agex* occurs at two different relative locations. If the intent were to use the same data for this variable, then this is a program defect. This hypothesis is further substantiated in the reference summary. The second occurrence of *agex* is used but is never defined.

Another point to note is that the array *rtable* is declared integer (46 bit precision) in the module *dartab* but in all the other modules is declared as real. The reference summary shows that the only module modifying or using this array is *dartab*. Therefore this is not a defect. The program could be clarified by removing this array from COMMON.

The reports in this example resulted from using the *-gt* option and format option.

Reference Summary for COMMON Block: comca

[illegible]

- Undefined and unused
- * ERROR: used but undefined

Example 6

This example shows the table and list formats for the parameter report. The tables identify the parameter names and the declaration types. The reference table shows that several occurrences of the parameter are defined, but otherwise unused. This information is reiterated in the exception report.

The reports in this example resulted from using the *-ptx* option and format options.

[illegible]

Parameter Reference Summary

[illegible]

> D - Declared but unused

Parameter Summary

Parameter dmax	declared but not used in module: clcplt
Parameter dmin	declared but not used in module: clcplt
Parameter maxpvt	declared but not used in module: dsubb
Parameter maxpvt	declared but not used in module: rddata
Parameter maxsp	declared but not used in module: clcplt
Parameter maxsp	declared but not used in module: wrplt
Parameter mxcrv	declared but not used in module: dsubb
Parameter rmax	declared but not used in module: clcplt
Parameter rmin	declared but not used in module: clcplt

Example 7

This example is an except from a local variable report using the partial listing option. All dummy arguments are listed. Two of these are unused in the module. There were no local variables identified with potential problems. The full listing option would list all local variables.

The report in this example resulted from using the *-lp* option and qualifier.

Module: dplt

cor	Real	Var	Dum_arg	-----> Output ----->
h	Real	Var	Dum_arg	----->> UNUSED <-----
hh	Real	Var	Dum_arg	<-- Input/Output -->
hlid	Real	Var	Dum_arg	<< Passed through >>
is	Int46	Var	Dum_arg	<-- Input/Output -->
it	Int46	Var	Dum_arg	<< Passed through >>
pi	Real	Var	Dum_arg	----->> UNUSED <-----
rough	Real	Var	Dum_arg	<< Passed through >>
u	Real	Var	Dum_arg	<-- Input/Output -->
vd	Real	Var	Dum_arg	<----- Input <-----
vg	Real	Var	Dum_arg	<< Passed through >>
xxg	Real	Var	Dum_arg	<-- Input/Output -->

END

**DATE
FILMED**

12/27/91