

SXT (TM) SOFTWARE EXPLORATION TOOLS

CXT (TM) C EXPLORATION TOOLS

CFT (TM) C FUNCTION TREE GENERATOR
CST (TM) C STRUCTURE TREE GENERATOR

DXT (TM) DBASE EXPLORATION TOOLS

DFT (TM) DBASE FUNCTION TREE GENERATOR

FXT (TM) FORTRAN EXPLORATION TOOLS

FFT (TM) FORTRAN FUNCTION TREE GENERATOR

JXT (TM) JAVA EXPLORATION TOOLS

JCT (TM) JAVA CLASS TREE GENERATOR

LXT (TM) LISP EXPLORATION TOOLS

LFT (TM) LISP FUNCTION TREE GENERATOR

SXT command line versions

SXTWIN Windows versions

December 1998

Copyright (C) Juergen Mueller (J.M.) 1988-1998.

All Rights Reserved World-Wide.

DISCLAIMER OF WARRANTY

THIS SOFTWARE AND ACCOMPANYING WRITTEN MATERIAL (INCLUDING INSTRUCTIONS FOR USE) IS PROVIDED "AS IS" WITH NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS WITH YOU.

THE AUTHOR AND COPYRIGHT HOLDER SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY THIS SOFTWARE OR ANY PART THEREOF.

IN NO EVENT WILL THE AUTHOR AND COPYRIGHT HOLDER BE LIABLE FOR DAMAGES, INCLUDING ANY LOST PROFITS, LOST MONIES, OR OTHER DIRECT, INDIRECT, GENERAL, SPECIAL, INCIDENTAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OR INABILITY TO USE THIS PROGRAM (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, BUSINESS INTERRUPTION, LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), MODIFICATION, DISTRIBUTION AND ON ANY THEORY OF LIABILITY, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

NO DEALER, AGENT, OR EMPLOYEE OF LICENSOR IS AUTHORIZED TO MAKE ANY MODIFICATIONS, EXTENSIONS, OR ADDITIONS TO THIS LIMITED WARRANTY.

ACKNOWLEDGMENT

BY USING THIS SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LIMITED WARRANTY AND ACCOMPANYING REMARKS, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT THIS IS THE COMPLETE AND EXCLUSIVE STATEMENT OF AGREEMENT BETWEEN THE PARTIES AND SUPERSEDE ALL PROPOSALS OR PRIOR AGREEMENTS, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN THE PARTIES RELATING TO THE SUBJECT MATTER OF THE LIMITED WARRANTY. YOU AGREE THAT THIS NOTICE APPLIES TO ALL FILES IN THIS SOFTWARE DISTRIBUTION.

You are expressly prohibited from selling this software or parts of it in any form, circulate it in any incomplete or modified form, distribute it with another product (except as Shareware) or removing this notice. No one may modify or patch any of

the executable files in any way, including, but not limited to, decompiling, disassembling or otherwise reverse engineering this software in whole or part.

The documentation may be distributed verbatim, but changing is not allowed. The information and specifications in this document are subject to change without notice.

THIS VERSION OF THE DOCUMENTATION, SOFTWARE AND COPYRIGHT SUPERSEDES ALL PREVIOUS VERSIONS. THIS SOFTWARE AND ITS RELATED DOCUMENTATION MAY CHANGE WITHOUT NOTICE. THIS DOCUMENTATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN, THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE DOCUMENTATION. THE AUTHOR MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENTATION AT ANY TIME.

This software and documentation is Copyright (C) 1988-1998 by

**Juergen Mueller
Eastleighstrasse 14
70806 Kornwestheim
GERMANY**

Email: jmsxt@compuserve.com

Homepage: <http://ourworld.compuserve.com/homepages/jmsxt>

THERE ARE NO AND HAVE NEVER BEEN RELATIONS BETWEEN THE AUTHORS PROFESSIONAL WORK AND THE SXT DEVELOPMENT. THE SXT PROJECT IS AND HAS EVER BEEN AN INDEPENDENT PRIVATE PROJECT OF THE AUTHOR AND IS WRITTEN AND MAINTAINED IN HIS FREE TIME.

LICENSE

This software is not public domain or free software, but is being distributed as shareware. It is protected by copyright and distributed under this license restricting its use.

Non-registered users of this software are granted a limited license for a 30-day evaluation period starting from the day of the first use to make an evaluation copy for trial use for the express purpose of determining whether this software is suitable for their needs. At the end of this trial period you should either register your copy or discontinue using this software. The use of unregistered copies of this software, outside of the initial 30-day trial, by any person, business, corporation, government agency or any other entity is strictly prohibited.

This means that if you use this software, then you should pay for your copy. This software is not free, but you have the opportunity to try it before you buy it. Either pay for it, or quit using it. A registration entitles you to use your copy of this software in its actual version on any and all computers available to you. If other people have access to this software or may use it, then additional copies or a site license should be purchased. If you want to use a newer version of this software you have to license it.

All users are granted a limited license to copy this software only for the trial use of others and subject to the above limitations. This license does not include distribution, selling or copying of this software package in connection with any other product or service or for distribution in any incomplete or modified form. Operators of electronic bulletin board systems and software servers (like Internet World Wide Web (WWW) Sites or FTP-Servers) are encouraged to post this software for downloading by their users, as long as the above conditions are met.

This package is expected to be distributed as shareware for download, on disk or on CD-ROM, but the fees paid for "distribution" costs (disk, CD-ROM) are strictly exchanged between the distributor and the recipient, and the author makes no express or implied warranties about the quality or integrity of such indirectly acquired copies. Distributors and users may obtain the package directly from the author by following the ordering procedures in the REGISTER files.

REGISTRATION REMINDER

Unregistered copies of this software are 100% fully functional. I make them this way so that you can have a real look at them, and then decide whether they fit your needs or not. This work depends on your honesty. If you use it, I expect you to pay for it. When you pay for the shareware you like, you are voting with your pocketbook, and will encourage me and other shareware authors to develop more of these kinds of products.

THANK YOU FOR SUPPORTING THE SHAREWARE CONCEPT

TABLE OF CONTENTS

1 THE SXT SOFTWARE EXPLORATION TOOLS	6
2 INTRODUCTION	8
3 PROGRAM DESCRIPTION	11
4 LANGUAGE IMPLEMENTATIONS	14
4.1 C-SOURCE CODE AND PREPROCESSING	14
4.2 C++ SOURCE CODE	15
4.3 DBASE / CLIPPER SOURCE CODE	16
4.4 FORTRAN SOURCE CODE	16
4.5 JAVA	17
4.5.1 JAVA SOURCE CODE	17
4.5.2 JAVA CLASS FILES	18
4.6 LISP SOURCE CODE	18
4.7 ASSEMBLER SOURCE CODE	19
5 DATABASE GENERATION	20
6 PROGRAM LIMITATIONS	22
7 SXT OPTIONS	26
8 MISCELLANEOUS	62
8.1 SOURCE CODE METRICS	62
8.1.1 INTRODUCTION	62
8.1.2 BASIC SOURCE CODE METRICS	62
8.1.3 OBJECT-ORIENTED SOURCE CODE METRICS	64
8.2 OUTPUT DESCRIPTION	67
8.2.1 CFT OUTPUT	68
8.2.2 CST OUTPUT	69
8.2.3 OUTPUT INTERPRETATION	70
8.3 TOOLS FOR DATABASE PROCESSING	70
8.4 INTEGRATION INTO PROGRAM DEVELOPMENT ENVIRONMENTS	82
8.5 IMPROVING EXECUTION SPEED	82
8.6 PROBLEMS	83
8.6.1 FREQUENTLY ASKED QUESTIONS (FAQ)	84
8.6.2 TROUBLE SHOOTING	84
8.7 REFERENCES	87
8.7.1 BOOKS AND ARTICLES	87
8.7.2 COMPILER REFERENCE MANUALS	88
8.8 TRADEMARKS	90
9 APPENDIX	92
9.1 APPENDIX 1: C/C++ PRECOMPILER DEFINES	92
9.2 APPENDIX 2: RESERVED C/C++ KEYWORDS	95
9.3 APPENDIX 3: RESERVED FORTRAN KEYWORDS	96
9.4 APPENDIX 4: EFFICIENCY	97
9.5 APPENDIX 5: SYSTEM REQUIREMENTS	99
9.6 APPENDIX 6: INSTALLATION	99
9.7 APPENDIX 7: SXT REVIEWS	99
9.8 APPENDIX 8: AVAILABILITY	100

1 THE SXT SOFTWARE EXPLORATION TOOLS

The SXT Software Exploration Tools are a collection of software analysis tools providing a similar functionality for different programming languages.

The following packages are available:

CXT - C Exploration Tools:

CFT - C Function Tree Generator

Tool to analyze and display the function call relationships within the source code of C programs.

CST - C Structure Tree Generator

Tool to analyze and display the structure/class relationships within the source code of C programs.

DXT - DBASE Exploration Tools:

DFT - DBASE Function Tree Generator

Tool to analyze and display the function call relationships within the source code of DBASE, CLIPPER '87 and other XBASE-like programs.

FXT - FORTRAN Exploration Tools:

FFT - FORTRAN Function Tree Generator

Tool to analyze and display the function call relationships within the source code of FORTRAN programs.

JXT - JAVA Exploration Tools:

JCT - JAVA Class Tree Generator

Tool to analyze and display the class relationships within the source code and class files of JAVA programs.

LXT - LISP Exploration Tools:

LFT - LISP Function Tree Generator

Tool to analyze and display the function call relationships within the source code of LISP and SCHEME programs.

Each of these packages consists of the analysis program ("Analyzer") and a recall program ("Navigator") to retrieve the analysis results which can be stored in a database, plus documentation and additional macros to integrate these tools into popular editors like BRIEF, QEDIT or MicroEMACS.

Each of these packages is available for the following systems:

SXT command line text mode versions:

- * DOS real mode
- * DOS 386 protected mode
- * Windows 32 bit (command line text mode)
- * OS/2 (command line text mode)
- * LINUX (command line text mode)

SXTWIN Windows versions:

- * Windows 16 bit (Windows 3.1/3.11)
- * Windows 32 bit (Windows 95, Windows NT)

There are no differences in the functionality between the versions for the different systems, except for the Windows versions (see SXTWIN.DOC for further information).

The author has no plans to port the SXT programs to other platforms or operating systems like Apple Macintosh, UNIX other than LINUX (SCO, Solaris, AIX, HP-UX, ...), Atari or Amiga. The source code of the SXT programs is not available.

IMPORTANT NOTICE ABOUT THIS DOCUMENT

Although this document is mainly based on the description for the CXT programs CFT and CST (which were up to version 2.13 the only public available SXT programs) and therefore very C / C++ related, the description applies in the same way to all other SXT packages. The names CXT / CXTWIN resp. CFT / CST, CFTN / CSTN and CFTWIN / CSTWIN can be simply exchanged by the similar other product names like DXT / DFT, FXT / FFT, JXT / JCT or LXT / LFT. Where necessary, the specific differences of the SXT packages are described, especially in the chapter about the command line options. I have done it this way to ensure an overall consistency, to keep all related things together and to reduce the effort for writing and maintaining this document.

2 INTRODUCTION

The SXT programs are powerful program development, maintenance and documentation tools. They are primarily intended for analyzing large programs, where it is difficult, if not impossible, for the programmer to find the structure of the whole program. They allow the analysis of the source code of applications, no matter how big or complex they are. The SXT programs are also very useful to explore unknown source code and to get complete overview about its internal structure. The re-engineering of old and/or undocumented source code becomes much easier with these programs. The tools help the programmer to analyze, identify, locate and access all parts of a large software system. They are designed to support software reuse, maintenance and reliability.

By preprocessing, scanning and analyzing the entire program source code as a single unit, these programs build an internal representation of the function call hierarchy (CFT, DFT, FFT, LFT) and of the data structure / type relations (CST, JCT). The Java version JCT can also analyze the contents of binary Java Class Files. The resulting output shows from a global view the interdependencies and hierarchical structure between the functions or data types of the whole, multiple file, software project. Several features and options allow the user to customize the generated hierarchy call tree output and to get a large set of useful information about the source code. The hierarchy structure is always up-to-date because it relies on the original source code as the primary source of information. Written software documentation often differs from that what really has been coded, so the source code itself is the ultimate documentation. The resulting output files can be used for various purposes like development or documentation. For registered users there are no restriction limits in using them for their own work.

The output with the call tree and the other information is written in ASCII format. Information about the functions, data types, files and directories can also be written as formatted text files and used as input for other programs like word processors or spreadsheet calculators. The programs can additionally generate the output in HTML (HyperText Markup Language) format which can be viewed by WWW (World Wide Web) browsers like Netscape Navigator / Communicator or Microsoft Internet Explorer (see option -HTML) and in RTF (Rich Text Format) format which can be imported by word processors. The RTF output can be used to compile Windows help files with the Windows Help Compiler which can be viewed with WinHelp (see option -RTF). The HTML output can also be used to compile Windows HTML Help files (see option -HTMLHELP).

Displaying and printing a graphical representation of the analysis results as a call graph is not directly supported by all SXT programs, only the SXT Windows versions have an integrated tree browser. The owners of RATIONAL ROSE, a powerful software development case tool, can (mis-) use this tool for such purposes because the SXT programs can generate compatible output which can be imported by Rational Rose. (see option -RATIONAL for a detailed description). For the same purpose, the displaying of call graphs, all SXT programs can also generate output for the freely available tools VCG ("Visualizing Compiler Graphs", see option -VCG),

Graphlet (see option -GML), DOT (see option -DOT) and daVinci (see -option daVinci) which perform the automatic layout and displaying of graphs.

An important feature is the database generation. It allows the recalling of information without reprocessing the source code. The database can again be read by CFT and CST to produce different outputs or to add new files. Special recall programs (CFTN and CSTN) allow fast searching for items in the database. These programs can be used within any environment, for example on the DOS command line or from inside editors like BRIEF, QEDIT or MicroEMACS (DOS and WINDOWS), to provide access to all functions and data types with just a keystroke. These features make a comfortable hypertext source code browser and locator out of your editor. A project consisting of several files appears to the developer as if it were a 'whole-part' of software. The developer can walk through programs and trace the logic without having to memorize the directories and files where functions or data types are defined and called. The SXT windows versions provide similar functionality with DLL's for database access.

A useful option of CST is the possibility to generate a source file with which size and byte offset calculations for structures/ unions and their members can be performed. This option is useful especially to support any kind of error searching or hardware debugging, for example with an ICE (Integrated Circuit Emulator), or if data structures have to be exchanged between different hardware platforms with different data alignment.

CFT can also be used to analyze "C"-like languages as they are used by several commercial programs. The macro programming languages of the BRIEF, EPSILON and ME editors are such languages and can be handled by CFT.

CFT and CST have been used since 1989 in several projects with applications ranging from single source files over medium sized applications (like the SXT programs themselves) up to large software projects with hundreds of source and include files, more than 40 MB of source code, more than 1 Million lines, 3500 functions and 1500 data types.

Many public available C/C++ sources (e.g. GNU-C compiler, GNU-EMACS, MicroEMACS, NCSA TCP/IP communication software package, SUIT - The Simple User Interface Toolkit, NIHCL - The National Institute of Health C++ class library, F2C Fortran-to-C translator, Wolfenstein 3D game and many others) were processed with sometimes surprising results during the development and have been used to test and improve the features, reliability, correctness, robustness and execution speed of CFT, CST and their related utilities.

Although the other SXT packages are much newer than CFT and CST, they all are closely related. The CXT programs are used as the common base for all other packages.

IMPORTANT NOTICE ABOUT SXT USE AND SXT RESULTS

The SXT programs are very powerful program development tools. However, due to their complexity, they are sometimes not easy to use and will need some care to produce useful analysis results. Therefore the SXT programs are mainly intended to advanced users. Absolute beginners will need some time to use SXT programs because they probably may be confused by the large number of features and user definable options. Please take time to read the documentation to become familiar with the SXT programs.

Due to the complexity of the SXT programs themselves and their task to analyze unknown software with source code of any kind, any origin and any size, there is absolutely no guarantee that the analysis results are in any way complete and correct for every possible case. Although any attempt has been made to make the SXT programs as reliable as possible these restrictions should always kept in mind when working with SXT programs.

For restrictions and limitations see also the "DISCLAIMER OF WARRANTY" and "ACKNOWLEDGMENT" sections at the beginning of this document.

3 PROGRAM DESCRIPTION

CFT builds a hierarchy call tree of every function with the called functions in it's own function block. These functions are again used as a starting point for subsequent function blocks. Starting the call tree with the "main"-function it will display the complete function flow chart and the function hierarchy dependency of the whole application with all user defined functions and the called library functions. Prototyped but never defined or called functions are also detected. Recursive calls of functions are recognized and displayed, even over several call levels. Repeated calls of previously displayed functions in the output call tree are detected and a message will be given with a reference to their first appearance. This prevents the output of complete subtrees displayed earlier. Overloaded C++ functions and operators are recognized and displayed with the number of overloadings.

CST acts similar to CFT but it works on data types like basic types, structures, unions, enumerations and C++ classes. CST builds a hierarchy call tree of every structure and union data type with their internal elements and their related data types. If these data types are again structures, unions or classes, the substructures will again be displayed. CST recognizes data types defined by 'typedef' and derived from other data types. The type names corresponding to the same basic type are displayed in the output file as 'alias' names for their common basic data type name. Every feature of CFT like the detection of recursive declared structures and unions, references to previously displayed data types and others are available and act similar.

Every function (CFT) and data type (CST) can be displayed with the name of the source file and the line number where it is defined. The output can be customized to display the tree chart as a call-tree ("CALLER-CALLEE"-relation: "WHO CALLS WHOM") or as a caller-tree ("CALLEE-CALLER"-relation: "WHO IS CALLED BY WHOM"). This feature allows the user to determine which functions are called from a specific function or which functions are callers of a specific function.

The function and data type extraction from the source code is done by scanning and parsing the source. There is absolutely no need for the programmer to mark functions or data types of interest, for example with special keywords, starting the definitions at the beginning of a line or to use comments containing special marks, as it is necessary for other source code analyzers and browsers. CFT, CST and the other SXT programs do not need these work-arounds, any source code can be processed without previous work. These tools are also compiler independent because they can be customized to support any kind of compiler.

Since the SXT programs always make a static analysis of the program source code, they are not able to detect references due to the expansion of a macro definition during runtime. This has to be considered for DBASE and LISP programs which allow such dynamic features. The same restrictions apply also to the use of function pointers for C and function names as formal parameters in FORTRAN which cannot be resolved for call graph generation.

Several useful information and software metrics about the processed source code and the included files can be generated like

- file size and comment size in bytes for every file,
- number of source code lines for every file,
- number of included files for every source file,
- total effective number of scanned bytes and lines for every source file and its included files, if files are included multiple times, this will influence the calculations,
- for every defined function the number of lines, the code and comment size in bytes, the number of bytes per line, the number of functions called, the number of flow control statements (*if, else, for, while, case, default, goto, return*), the maximum brace nesting level and if the function is used only inside the file,
- for every defined structure/union the total number of elements and the number of elements which are themselves structures/unions,
- file function or data type reference list for every file,
- total number of displayed, defined, undefined or multiple defined functions and data types,
- location of multiple defined functions and data types,
- location of overloaded C++ functions,
- source file - include file dependencies for every source file (MAKE-dependencies),
- source file - include file hierarchy tree (include file relations),
- final statistical summary for all files,
- test for user defined metric limits like acceptable number of bytes per file, acceptable number of lines per file, minimum comment portion per file, ...,
- cross reference of every occurrence for every function or data type,
- parent/children relationship for every function and data type,
- critical function call path/structure nesting with deepest non-recursive nesting level (unlimited tree depth),
- C++ class inheritance tree,
- JAVA class / interface inheritance tree,
- support for C structure/union byte offset calculation,
- FORTRAN subroutine CALLs,
- FORTRAN COMMON blocks,
- generation of description files for call/inheritance graph visualization with the RATIONAL ROSE CASE tool,
- generation of HTML-output for call tree visualization with WWW Browsers like Netscape Navigator or Microsoft Internet Explorer,
- generation of RTF-output for import to word processors and for compiling Windows Help files,
- generation of graph description files for graph layout programs like VCG, Graphlet, DOT and daVinci,
- integrated function / data type tree / list browser (Windows 32 bit versions only).

The resulting hierarchy structure chart is another representation for a directed call graph. A directed call graph consists of nodes (functions or data types) and connections (call relations) between these nodes. The number of nodes and

connections which are necessary to transform the hierarchy structure chart into a directed call graph will also be calculated as an additional information about the system complexity.

A large number of options to control the program execution and the output generation are available and can be defined on the command line, with interactive dialog windows (applies only to Windows versions), by command files or by defining them in an environment variable used by the program.

CFT, CST and the other SXT programs can be directly invoked from inside editors or integrated development environments like the Borland C++ IDE. Detailed examples for the integration together with necessary macro or batch files are given. The SXT windows versions provide DLL's (Dynamic Link Libraries) to access the generated databases and to retrieve information. These DLL's can be used with user developed programs e.g. written in C, C++, Visual Basic, or from other Windows applications like MS-Word for Windows, MS-Excel and others.

4 LANGUAGE IMPLEMENTATIONS

4.1 C-SOURCE CODE AND PREPROCESSING

The ISO/ANSI C language standard ISO/IEC 9899:1990 (E) resp. X3.159-1989-ANSI C as described in several books about the C-language (see references) was used as a development base. The reserved keywords being recognized are not only the original ISO/ANSI C keywords but were also taken from several compiler implementations like Microsoft, Borland or GNU and their own special language extensions. The books "The C++ Programming Language" and "The Annotated C++ Reference Manual" (ARM) together with information about the work of the ANSI C++ committee X3J16 resp. the ISO/IEC working group SC22 WG21 were used for the C++ keywords. Another major source was the AT&T C++ release 2.1. Compiler specific extensions especially from GNU are also recognized.

A complete list of all reserved keywords is show in appendix 2. The large set of keywords may lead to some slight problems in situations where a keyword is not used as itself but as an identifier name, for example a C++ keyword used as an identifier in C.

During a normal file scan, precompiler defines are, if possible, handled as if a real precompiler would be present, but this can cause some trouble with '#if', '#ifdef' and other precompiler controls which are not evaluated. Also the block nesting level, which will be monitored by the source code scanner, may not be at level 0 at the end of the file because of such precompiler controls. To avoid such things, a built-in C-preprocessor allows the complete preprocessing of the source code and include files for several compiler types as an additional option (-P).

Preprocessing or not is a little bit controversial because it can either result in a loss of information if macros are used to change the program behavior and hide function calls, it can lead to errors during file scanning or it can change the function and data type information obtained from the code which may not exactly correspond to the visible source code. Preprocessing can be an advantage or not, so the user has to decide whether he does it or not (see options -P, -I, -E for more information).

There are two C preprocessors integrated (this applies not to the 16 bit versions): The original GNU C preprocessor (since CXT 2.55) and an older preprocessor (since CXT 1.78). The GNU C preprocessor is the default preprocessor in the 32 bit versions (except OS/2) and very fast but does not give very much diagnostics information. If such information is needed in some cases, the old preprocessor should be used at the cost of a quite slow preprocessing. For more information about the two preprocessors see also option -NOGNUPP.

The preprocessor handles the defines for Microsoft C 5.1, Microsoft C/C++ 7.0, Microsoft Visual C++ 1.5 / 2.2 / 4.0 / 4.1 / 4.2 / 5.0, Turbo C++ 1.0, Borland C++ 2.0 / 3.1, Borland C++ 1.0 for OS/2, Watcom C/C++ 10.0, GNU-C 2.2.2 / 2.6.3 / 2.7.2 / 2.7.2.1 / 2.8.1 and Intel 80960 C compiler iC960 3.0 and their memory models (if necessary) or CPU architectures for the Intel 80960 32 bit RISC processor (KA, KB,

SA, SB, MC, CA). Other compiler types can be specified with the -B, -D and -U options. The default ISO/ANSI C predefined macros `'__FILE__'`, `'__LINE__'`, `'__DATE__'`, `'__TIME__'` are generated for preprocessing. The macro `'__STDC__'` is NOT defined (some compilers test with `'#ifndef __STDC__'`), so that non standard ISO/ANSI C extensions in the processed code are allowed. Defining `'-D__STDC__=1'` forces ISO/ANSI C conforming output (if used by the scanned source code, of course!). Additional supported precompiler defines are `'__TIMESTAMP__'`, `'__BASE_FILE__'` and `'__INCLUDE_LEVEL__'`. A list of the predefined preprocessor defines for the supported compiler types is shown in appendix 1 together with tips for the adaptation of other compilers. Features like the replacing of trigraphs, digraphs and the recognition of C++ comments `'//...'` are also treated by the preprocessor.

The old preprocessor recognizes several errors or possible sources for problems like

- the use of undefined variables in precompiler controls,
- misbalanced `'#if...'` control block(s) including the exact location (file, line) where the failing block started,
- recursive called include files,
- nested include files,
- wrong number of macro arguments

and displays diagnostic messages with an exact description of the error or warning reason and its location in the source file.

4.2 C++ SOURCE CODE

For the description of the related C++ language standards and other literature see the chapter about the C language implementation.

Although CFT and CST were initially not developed to process C++ code it is possible to do so. However, some restrictions and limitations should be considered. The recognition of C++ classes by CST is limited because the handling of classes is very complex. The C++ class inheritance relationships are recognized and shown in a class hierarchy graph listing (option -b). Function bodies inside a class or structure definition may cause errors. Templates are not supported and will not be recognized.

Calling member functions will not be recognized correctly due to missing class name and name scope resolving, this leads also to an incomplete CFT call tree and warnings during analysis. The use of overloaded functions with equal names but different parameters in C++ programs may lead to incorrect calling relationships. A variable initialization with parameters will be misinterpreted as a function call. A correct handling of these and other C++ features requires a complete C++ source code analyzer to keep track of the class functions belong to and the different calling parameters.

If precise information about C++ code structure is needed, utilities like 'class hierarchy browsers' or 'class viewers', which are usually part of C++ compiler environments, should be used instead.

4.3 DBASE / CLIPPER SOURCE CODE

DFT can process source code which is based on the DBASE III/IV programming language. This means that also source code written in DBASE derivatives like CLIPPER (Summer '87) or probably FOXBASE can be analyzed. The source code analyzer tries to be as correct as possible to build a reliable hierarchy tree. A function/ procedure declaration is recognized by the FUNCTION resp. PROCEDURE keyword. A function/procedure call is recognized by the following statements:

```
function()  
CALL function  
CALL function WITH parameters  
DO function  
DO function WITH parameters
```

If a file contains no function/ procedure declaration, the filename itself is taken as procedure name. The recognition of built-in functions/ procedures can be ignored (see option -E). All tokens are assumed case-insensitive and are internally converted to upper-case characters. System built-in functions/macros can be specified by option -E. Include files (e.g. with CLIPPER) are not processed. The Clipper 5 C/C++ style comments /*...*/ resp. // are recognized.

4.4 FORTRAN SOURCE CODE

FFT can process source which is based on the FORTRAN 77 standard. Each FORTRAN line is divided into fields for the required information, each column represents a single character.

COLUMN	FIELD
1	comment indicator (C,c,*,!)
1-5	label
6	indicator for line continuation
7-72	statement field (standard is 72, extended to 132)

FFT can process lines up to 132 columns, however, if characters are found beyond column 72 a warning will be given (this can be disabled with -NOWARN72). Note that the default TAB-size of 8 characters may lead sometimes to such unwanted messages which might not occur with TAB-size 6. Continuation lines are merged before they are analyzed. The number of continuation lines is 19 by default and can vary between 0 and 99 (option -qn). Inline comments in the statement field start with '!', text until end of line is ignored. The standard intrinsic functions and additional VAX-FORTRAN intrinsic functions are recognized. Statement functions (comparable

to C function-like macros) are not recognized and may lead to 'undefined functions'. Hollerith constants are not recognized and handled. All tokens are assumed case-insensitive and are converted to upper-case characters. Blanks are not significant and are removed except inside character strings. If option -I is set, INCLUDE statements are recognized and processed. To handle this implementation dependent feature, several different types of include statements are accepted:

C FORTRAN-LIKE SYNTAX, INCLUDE STATEMENT STARTS IN COLUMN 7
include-statement include-filename

C C-LIKE SYNTAX, INCLUDE STATEMENT STARTS IN COLUMN 1
include-statement include-filename

where include-statement is one of INCLUDE, #INCLUDE, \$INCLUDE or %INCLUDE and include-filename is one of 'filename', "filename", <filename> or filename (without surrounding characters). See option -I and the chapter about PROGRAM LIMITATIONS for additional information about file inclusion.

The resulting function call graph may be incorrect due to the ENTRY capability of FORTRAN which allows direct jumps into a function or subroutine body from the outside. This may result in incorrect relationships for the ENTRY statement and the surrounding function/subroutine. The indirect call of functions which are given as formal parameters to subroutines or functions (comparable to pointers to functions in C) are not correctly referenced. Implicit typing for function definitions without return type will not be recognized, the function will be displayed without a return type.

4.5 JAVA

4.5.1 JAVA SOURCE CODE

JCT can process Java source code which is based on the information in "The Java Language Specification", Version 1.0, published by SUN Microsystems in August 1996. Usually, Java source files have the file extension '.JAVA' or '.JAV' (case insensitivity is assumed).

JCT may be confused by ambiguous type names where a detailed naming scope resolution is necessary to determine the exact type name according to package and inheritance rules. JCT does not contain such a complex naming resolution mechanism and may sometimes produce incorrect results, which means that not the correct name is referenced. Currently there is also no handling of so called 'inner classes' defined within another class, which were introduced with Java SDK 1.1. Such 'inner classes' are simply skipped.

If absolutely precise information about Java code structure is needed, utilities like a 'class hierarchy browser' or a 'class viewer', which are usually part of Java development environments, should be used instead.

4.5.2 JAVA CLASS FILES

Besides the analysis of Java source code, JCT can also analyze binary Java Class Files (JCF). The internal format of Java Class Files is described in "The Java Virtual Machine Specification", published by SUN Microsystems in September 1996. JCT considers files with extension '.CLASS' (case insensitivity is assumed) as Java Class Files, for all others Java source code is assumed.

For the analysis results of Java Class Files, the same restrictions apply as stated before for the analysis of Java Source Code. Furthermore, during the analysis of Java Class Files there are no consistency and plausibility checks performed on the Java Class File contents as is described in the "The Java Virtual Machine Specification" and usually performed by Java class loaders and Java runtime environments. This means that an incorrect Java Class File content may produce a JCT program crash. Concerning the sequence of class attributes and class methods within a class, this may differ between the original source code and the results obtained from a Java Class File since the internal structure of Java Class Files do not preserve the original sequence from the source code.

4.6 LISP SOURCE CODE

LFT can process LISP and SCHEME source code. The development of LFT was mainly based on the GNU-EMACS LISP dialect as it is used in the GNU-EMACS macro extension language and its functionality was tested mainly with these macro files. LISP functions/macros are recognized by the DEFUN and DEFMACRO keywords. SCHEME functions are recognized by the DEFINE keyword, SCHEME processing is enabled by option -XSCHEME. Unnamed functions declared with the LAMBDA keyword can be recognized optionally (option -XLAMBDA). Tokens are assumed case-sensitive. Comments are recognized for ';' until end-of-line and between '#|' and '|#' as multi line comment blocks.

The source code analysis is performed in two passes: The first pass detects function/macro declarations and the second pass analyzes the relationships. Function calls via (funcall <fcn>), (function <fcn>), (apply <fcn>), (mapc <fcn>) and similar constructs may not be correctly evaluated if fcn is a function-symbol (e.g. given as a function parameter) and not a valid function name. System built-in functions/macros can be specified by option -E.

LFT was designed to work with different types of LISP source code (as there are XLISP, CLOS, GNU-EMACS LISP, ...), although the large number of dialects may lead sometimes to problems.

4.7 ASSEMBLER SOURCE CODE

As an additional feature, CFT and FFT can process assembler source code for the Intel 80x86 processors (MASM 5.1, TASM) and for the Intel 80960 RISC processors (or any other "AT&T UNIX-like assembler" like GNU) to get information about assembler procedures and functions being called from the assembler source files. This feature is useful for mixed language programming. The assembler source code scanner also detects and handles calls of include files. The processing of assembler macros, however, is not supported, the preprocessing option (-P) works only on C source code. Assembler source files are recognized by their file extensions '.ASM' and '.S', there is no other way to force a file being processed as an assembler file.

The following naming convention is used: For '.ASM' assembler files (MASM, TASM) identifiers are either treated case-sensitive (CFT) or case-insensitive (FFT) and will be transformed to upper case characters. Identifiers in '.S' (GNU, 1960) assembler files are treated case-sensitive. The first leading underscore of a function name will be removed to get exact naming matches. Type modifiers in C source code like 'cdecl' or 'pascal' will not be considered. Remember these conventions when processing C, FORTRAN and assembler files.

If the return type of an assembler function cannot be found, the string '<assembler procedure>' will be used instead. This is just to satisfy the need for a return type in the internal output routines.

Assembler code statements (inline code) inside C source code will not be processed and will be skipped, because it is too difficult to handle the several kinds of syntax being used for this like 'asm ...', 'asm "..."' or 'asm(...)' and the different keywords ('asm', '_asm', '__asm', '__asm__', ...) used by various compiler implementations.

5 DATABASE GENERATION

One of the most important features is the database generation (option -G). It is performed after writing the output file to save all information about the processed files in a dBASE compatible database for later use. The database files contain all necessary information like function or data type names, the location where they are defined, their caller / callee relationship, all scanned files with statistic information, include files and so on. I have tried to store the information in the most compact and effective database structure to save disk space. Note that if the contents of the database files is manipulated by external tools like dBASE or something else, the internal consistency will be corrupted and wrong or unexpected results will happen!

The database can be used to recall information, for example to find out, if and in which file and on which line a specific function or data type is defined. A database can be read (option -g) to add new files and/or to produce another output file with new options, for example with a reverse call tree or only with a special selected item being displayed. Such an incremental database generation is also useful if large projects can be divided into a set of common files and project specific files. A good example for this is the GNU C compiler, which consists of language independent files and three language dependent file sets for C, C++ and Objective-C. To analyze this software the language independent part can be stored into a database which is later reused for the language dependent parts to build the complete set of information.

Note that there is currently no possibility to update the analysis results for files which are already in the database and have changed afterwards, the user is only informed about such files. Currently it is only possible to add files but not to update analysis results.

The ability to retrieve information about the sources from the database is useful in many cases. Recalling information from a database is much faster than processing all the sources again to find a specific item of interest. The documentation and maintenance of large software projects is much more effective and easier to do if the developer has a tool to navigate through the source code and that helps him in his comprehension of the program and its internal structure. It is also useful for reverse engineering of source code to get an overview of the internal program structure. Together with user programmable editors it is possible to offer a source code browser with a hypertext like feeling by integrating database recalling functions into the editors.

Additional utility programs to retrieve information from databases, called CFTN and CSTN (for CFT, CST), are available. Supporting macros for their integration into the BRIEF, QEDIT or MicroEMACS editor are described in another section of this manual. The SXT windows versions provide also an interface to the databases via DLL's which are part of the Windows version.

The 32bit Windows 95 and Windows NT versions of CFT and CST support with option -g also the reading of Microsoft Browse Info Files (extension '.BSC') which

can be generated by Microsoft Visual C++. A Browse Info File is a database that contains information from the compilation and linking process with all function and data types used for the project. CFT and CST can read these databases and handle the results in the same way as they do it with their own analysis results. For more information see option -g.

6 PROGRAM LIMITATIONS

First of all, CFT and CST as well as the other SXT programs cannot replace a compiler or a syntax checker like 'LINT' (or similar language specific checkers) to detect errors in the source code. This means that it should be possible to compile the source code without fatal errors before it is possible to analyze it, otherwise the processing results may be incorrect (and may be the system crashes ...).

However, there are some situations where CFT and CST and the other SXT programs can be useful to detect bugs and inconsistencies in the source code like

- multiple definitions of functions or data types,
- different function return types,
- implicit declared functions with no prototype,
- function definitions used as prototype,
- recursive, nested, hidden and frequent calls of include files,
- unterminated strings or character constants,
- nested comments,
- unterminated comments at end of file,
- misbalanced braces,
- unexpected end-of-file characters inside files,
- illegal characters in the source code,
- wrong number of macro arguments,
- missing macro arguments,
- misbalanced '#if...' control blocks.

These code checks are done on multiple files in multiple directories so that inconsistencies between different files can be found and displayed. This is a capability which conventional compilers working only on a single file at a time cannot provide and will miss therefore (maybe the linker will find some of these inconsistencies).

Statistical information about the source code may not be correct if preprocessing is enabled (-P). The size of the 'pure' source code may not be correct due to macro expansion or removing of unnecessary blanks. However, the total file size is always correct because it will be taken from the source file.

Some limitations for the real mode versions are caused by the amount of available memory. The use of memory managers like EMM386, QEMM or 386MAX can help to get more free conventional memory. If memory problems ("out of memory" message) occur during processing, the 32 bit protected mode versions of CFT and CST, called CFT386 and CST386, should be used, which have no memory limitations and are much faster than the real mode versions. There are also SXT versions for Windows 3.1, Windows 95, Windows NT and OS/2 which have no memory limitations.

The number and the sizes of files to be processed is nearly unlimited with 2^{14} files and 2^{31} bytes maximum file length. Each file can have 2^{16} lines. The number of functions and data types being handled is limited to 2^{14} . These values are given for

the real mode versions, the protected mode versions usually exceed them. These limitations should be enough even for the biggest project that could be mentioned.

The ISO/ANSI C minimum requirement for include file nesting is 8 levels and will be fulfilled by CFT and CST. The maximum include file nesting level is limited by the number of files (streams) which can be opened simultaneously. This is a compiler specific limit usually coming from the library startup code. The number of open files (streams) is defined by the macro `FOPEN_MAX` in the include file `'stdio.h'`. During preprocessing the number of nested files is usually less than `FOPEN_MAX` because several streams are used for the default I/O (`stdin`, `stdout`, `stderr`, `stdaux`, `stdprn`). Also the preprocessor needs additional streams besides those for the source and include files: One stream for the temporary output file, one for a log-file (if option `-L` is set) and one for a file list file (if this is declared on the command line with `@`). The maximum number of files being opened simultaneously differs for the various SXT program versions due to the built-in limits of the different compilers used to produce them. All SXT programs which process include files (CFT, CST, FFT) should be able to handle at least 10 include nesting levels. The message for such an include file open error is `'too many open files'`.

The integrated old C-preprocessor limits the size of expanded macros to 6 Kbytes. The number of macros simultaneously defined is unlimited (ISO/ANSI: 1024) and only affected by the available memory. The number of macro parameters is limited to 31 (ISO/ANSI: 31) and there are up to 31 significant characters (ISO/ANSI: 31) recognized. The conditional compilation nesting levels of `'#if ... #endif'` control blocks is limited to 32 (ISO/ANSI: 8). The new GNU-C preprocessor does not have these limits.

The line length is unlimited (ISO/ANSI: logical (?) line length is 509 characters). The number of characters in a string (including `'\0'`) is 2048 (ISO/ANSI: 509). The number of members in one structure/union is unlimited (ISO/ANSI: 127), the number of structure/union nesting levels is unlimited (ISO/ANSI: 15).

The recognition of C/C++ identifiers like function and variable names follows the standard rules: an identifier consists of upper and lower case letters (A-Z, a-z), underscore (`_`) and digits (0-9), additionally the dollar sign (`$`) will be accepted. National character set extensions as they are usual for languages in European countries like Germany, Denmark or Sweden can be defined with option `-J`.

C++ comments `'//...'` are usually only recognized if option `-C++` is set. However, to accept the non-standard extension of some compilers (like GNU, Microsoft and Borland) which allow such comments also in C source code, they are recognized and handled unless option `-NOCPPCMT` is set. Nested C style comments `'/*...*/'` are not allowed and will always produce warnings.

CFT and CST may produce warnings with wrong line numbers if preprocessing is enabled (option `-P`) and if the warning occurs inside a comment. The reason is that line number synchronization with `'#line ...'` is only guaranteed for executable source but not for comments. In such a situation the source code should be processed

without -P to get the correct line number (such warnings are usually related to unexpected characters).

The use of explicit preprocessor #line directives in C/C++ source leads to different results for the 'logical' (synchronized with #line) and physical file line numbers. CFT and CST use the 'logical' line numbers and therefore moving inside a file with an editor to a specific line may fail.

The calculation depth of the critical function call path or structure nesting level is unlimited. The calculation is an extremely recursive function and was successfully tested up to more than 100 nesting levels. It is not known for which nesting level a stack overflow will happen.

CFT cannot recognize and reference a function if it is used with its pure name without parentheses. This happens if a function name is assigned to a function pointer variable or used as a function pointer argument in a function call. Indirect calls to a function via a function pointer cannot be resolved. A similar case with FFT is the use of function names as formal parameters for calls to subroutines or functions.

CFT will be confused in some cases by extensive type-casting operations like 'void __based(void) * __cdecl ... ()' or '__declspec(...)' and will display unexpected messages. A function prototype declaration inside a function block ('function given scope') will not be recognized by CFT as a prototype declaration, instead it will be interpreted as a function call. In assembler source code, some definitions of local variables seem to look like a function or a label definition and are treated by CFT like that although this may be wrong in some cases. It is also not always possible to detect a call of a local label correctly. CFT sometimes displays warning messages about 'return type mismatch' though this may be correct in that special case because the different types are earlier defined by a 'typedef' declaration. The reason is simply that CFT doesn't recognize these 'typedef's (but CST does), it looks only for function names.

An often requested feature for CST is the integration of the calculation of structure/union sizes with byte offset information for every structure/union member. This feature is not implemented in CST. The reason is that this would require CST to treat the various compiler implementations with different basic type sizes (sizeof(int), sizeof(long double)) for different processor types (16 bit, 32 bit, 64 bit, ...) and data type alignment requirements (by default and also controlled with #pragma's like 'align' or 'pack'). It would be possible to do this for just one selected compiler implementation or processor type but not for several different compilers. Especially compilers for advanced architectures like RISC processors have very complicated alignments rules depending on the data types, alignment pragmas, compiler switches, type sizes, available register number and register sizes and resulting structure/union/class sizes to generate highly optimized code. This includes usually the insertion of 'fill' bytes inside a structure/union and sometimes 'padding bytes' at the end of a structure/union to force aligned sizes on specific byte boundaries (For examples see the reference manual of the Intel 80960 C-Compiler iC960, release

3.0). Because of these reasons, an integrated 'byte offset calculation' is not implemented in CST. Instead, a source file for selected data types can be generated with option -O, which performs these calculations, if you compile the generated file with your C compiler. For further information see the description for option -O.

Option -z in combination with option -I produces redundant results with CFT, CST and FFT if files with executable statements (or something that can be interpreted as that, e.g. function calls, data types or COMMON blocks) are included directly inside a function or data type block (especially in FORTRAN this seems to be common practice to include COMMON blocks into function and subroutine bodies). For CFT and CST (but not for FFT) also the call tree references will be incomplete and therefore incorrect. With option -P for CFT and CST everything works fine, because the preprocessor works more precise as the simple file inclusion mechanism option -I uses.

SUMMARY

The above described limitations can lead in some situations to misinterpretations or loss of information of the scanned source code. The only way to avoid these lacks would be the inclusion of parts of a 'real compiler' to handle the complete language syntax in any possible situation. But this was not the intention when the development of these programs as 'little' and easy to use general purpose programming supporting tools began. Although I hope that the SXT programs will in most cases be powerful and useful development and documentation tools!

7 SXT OPTIONS

This section gives a complete overview about all SXT options and their syntax. It gives also remarks for their use and shows several examples with detailed descriptions. The options are case-sensitive! There are no differences between the real mode and the other versions of the SXT programs. In the Windows versions, all options can be set both on the command line and by menu selections and dialog windows. For every option the SXT programs which support it are listed in parentheses. This section of the documentation should be read very careful by all users to get an overview about all the features which are provided.

THE OPTIONS ARE LISTED IN LEXICOGRAPHICAL ORDER. NONE OF THESE
OPTIONS IS SET BY DEFAULT.

SYNTAX:

CFT	[options [\$cmdfile]] <[+]file> <@filelist>
CST	[options [\$cmdfile]] <[+]file> <@filelist>
DFT	[options [\$cmdfile]] <[+]file> <@filelist>
FFT	[options [\$cmdfile]] <[+]file> <@filelist>
JCT	[options [\$cmdfile]] <[+]file> <@filelist>
LFT	[options [\$cmdfile]] <[+]file> <@filelist>

(for LINUX use '=' instead of '\$' to mark cmdfile)

OPTIONS: (valid for SXT program)

-Bsizes (CFT, CST)

Redefine the basic type sizes and pointer type sizes (all values must be declared in bytes) for conditional preprocessor controls with the 'sizeof()' keyword like '#if sizeof(int) == 4'. This option is only valid with the -P option.

The required format for this option is

-Bv,c,s,i,l,f,d,ld*data,code
(delimiter between data and pointer sizes is '*')

with the following types and their respective default data size values in bytes (the pointer type sizes are model dependent):

```
v : void (sizeof(void) is usually 0, but for GNU-C it is 1)
c : char (1 byte)
s : short (by definition 2 bytes, hardware independent)
i : integer (hardware dependent, 2 or 4 bytes)
l : long (4 bytes)
f : float (4 bytes, IEEE format)
d : double (8 bytes, IEEE format)
```

ld : long double (10 bytes, IEEE format, some compilers assume long double == double (= 8 bytes), some CPU's and their compilers have special alignment requirements like the Intel 80960, where sizeof(long double) is 16 bytes due to register and memory access requirements and structure alignment)
data : data pointer (type pointers, 2 or 4 bytes, memory model dependent)
code : code pointer (function pointers, 2 or 4 bytes, memory model dependent)

The sizes of signed and unsigned types of the same basic types are considered equal, this means that, for example, the following expression is true:

`sizeof(unsigned int) == sizeof(signed int) == sizeof(int)`

The sizes of type pointers to data and function pointers to code are also considered equal, this means that, for example, the following expressions are true:

`sizeof(int *) == sizeof(float *)`
`sizeof(int (*)()) == sizeof(float (*)())`

A 64 bit (8 bytes) integer type like 'long long int' or 'bigint' (or something else) is currently not supported although some (co-) processors and their assemblers are able to handle it (see Intel 80960 assembler manual for examples). Also the DEC Alpha processor with its 64 bit architecture should support this.

If the -B option is not set, the default values for the various memory models and compiler types (as they are known to me) are used, the assumed target hardware has an Intel 80x86 microprocessor. Note that during preprocessing type modifiers like "near" or "far" are not recognized.

If the -B and the -T options are not set, the sizes of data pointers and code pointers are always considered equal:

`sizeof(int *) == sizeof(int (*)()) (= 4, large model)`

For example, -B0,1,2,2,4,4,8,10*4,4 would be the correct declaration for MS-C 7.0, large/huge memory model, with the values for data types (void = 0, char = 1, short = 2, int = 2, long = 4, float = 4, double = 8 and long double = 10 bytes) and pointers to data types and function pointers (all values 4 bytes). These values are set automatically by defining -TMSC70,L (or -TMSC70,H) as compiler type and memory model description for preprocessing.

-BATCH[inifile] (SXT WINDOWS VERSIONS ONLY)

This command line option can only be used with the SXTWIN programs. It allows to start a SXTWIN program in batch mode via the 'File' 'Run' menu or any other program. The SXTWIN program executes the commands (from the command line and INI-file) and is closed automatically after having finished. The optional parameter 'inifile' specifies the complete name (drive:/dir/filename) of the INI-file that

should be used for analysis. The SXTWIN program window is initially minimized and all messages and progress info is disabled to speed up processing.

-C++ (CFT, CST)

Enable C++ source code processing. This includes the definition of the macro name `'__cplusplus'` for preprocessing, the recognition of C++ keywords and the handling of C++ comments `'//...'`. Option -C++ is strictly recommended to process C++ code.

-C[s] (CFT, CST, DFT, FFT, JCT, LFT)

List the function / data type contents for every processed file, 's' sorts by line numbers (DEFAULT ORDER: lexicographical). There are additional information possible with the option -s. A remark is given if none of the functions defined in a file is called from functions defined in other files (internal versus external linkage). Functions for which no external caller outside the file is found will be marked [INTERNAL], such functions are candidates for defining them as 'static' (Calling a function by a function pointer won't be noticed!). This information is useful to find out whether the contents of a file is unnecessary for the project so that the file must not be linked. This option gives useful information about source code metrics for every defined function. Most of the values are given in the format "average [minimum ... maximum]". Almost the same information is provided for data types with CST. For CFT, the values may not be correct if `'#if line...'` constructs are used with not strictly ascending line numbers as they may be generated by tools like LEX or YACC!

-CALL (FFT)

Recognize and display only subroutine `'CALL ...'` statements in a function or subroutine body. In some cases this option can be useful if the source code scanner gets confused by indexed array accesses which might be misinterpreted as function calls. With this option only the `'CALL ...'` statements are detected.

-CLIPPER (DFT)

Handle CLIPPER specific extensions: comments `'/'` and `'/* */'` and extended character set `':{}'`.

-COMMON (FFT)

Recognize and display COMMON block names. The block name is surrounded by `'/'` like `/name/`, a blank COMMON block is named `//`. In the output file, COMMON block names are handled like function names, e.g. the call statistics says `'# calls'`. Note that the BRIEF and MicroEMACS macros cannot handle block names correctly since they do not accept the `'/'` character.

-CPPWARN (CFT, CST)

Check for C++ keywords being used in C source code. This is definitely not an error for C source code, however, if this source code will be later incorporated into a C++ project serious portability problems will arise. This check will not be done if option -C++ is set or if warning level is 0.

-CTAGS[x] (CFT)

This option generates a CTAGS file with information for VI-like tagging of defined identifiers. The CTAGS file is named "TAGS" by default unless the optional extension 'x' is used to specify another filename. Each entry in the TAGS file has the format

```
identifier<tab>file<tab>vi-search-pattern
```

where 'vi-search-pattern' is a regular expression matching the line where 'identifier' is defined. This is the default implementation for CTAGS. The generated TAGS files were successful tested with the MSDOS versions of VI (from Mortice Kern Systems Inc.) and the public domain VI-clone ELVIS, but should also work with other VI versions. Since VI originates from UNIX, the TAGS file contains only LF, not CR+LF, as line ending. I have done it this way to avoid possible problems with close-to-UNIX VI ports, but this may also lead to new problems if CR+LF is really needed. To change from DOS to UNIX styles and vice versa, tools like DOS2UNIX or UNIX2DOS can be used. See also option -TAGS for additional information.

This option cannot work together with option -P (preprocessing), this applies also to results generated from databases previously created with option -P. The reason is that the preprocessed source does usually not correspond with the original source, it contains additional #line directives, comments and obsolete blanks are removed, file offsets are not valid. Therefore the option -CTAGS can only be used with the original source code.

-D[.] (CFT, CST, DFT, FFT, JCT, LFT)

Specifies macro name(s) (-Dname or -Dname1=name2) or file with macro names (-D@filename) of functions/data types which should be predefined and linked together, also used as preprocessor define if the integrated preprocessor is called (-P). The defined names are case sensitive and trigraph and digraph translation is performed on them. For preprocessing, the -D option has lower precedence than the -U option. That is, if the same name is used in both a -U option and a -D option, the name will be undefined regardless of the order of the options.

The definition of a string as replacement for a macro name is different on the command line and inside a macro definition file or command file (marked with '\$'). On the command line, the double quotation marks must be 'escaped' and the string must be quoted like '-DXYZ="\123\'' (similar to C strings) to work correctly, the reason is the DOS wildcard expansion of the command line. Inside a macro definition or command file, the double quotation marks need not be 'escaped', so the

definition can be written like '-DXYZ="123"'. This option cannot be used in environment defines if the equal sign '=' is used because this produces a syntax error for DOS when trying to store a 'SET=...' command with a second equal sign in one line. If a define item consists of two words see the notes at option -S for a description. Keep these differences and exceptions in mind to avoid unexpected results using the -D option.

-DOT[x] (CFT, CST, DFT, FFT, JCT, LFT)

This option generates a DOT output file with function resp. data type call graph information. The DOT output file is named "prog.DOT" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT) by default unless the optional extension 'x' is used to specify another filename. There are two special cases for 'x' which can be defined additionally. '-DOT+' prints for each item the filename and line number where it is defined. For CST and JCT, '-DOT*' prints for each structure, union and class its data type members.

DOT is part of the AT&T "Graph Visualization" package which is a freely available tool for the automatic layout, displaying and printing of complex graphs. The package consists of several layout programs (DOT, TCLDOT, DOTTY, NEATO) which can read a DOT specification file and visualize the graph resp. generate PostScript output. There are versions for Unix and for DOS. Graphlet was developed by AT&T. More information about the Graph Visualization package and DOT (including executables and source code) can be found at <http://www.research.att.com/sw/tools/reuse>.

-Ename (CFT, CST, FFT)

Almost the same as -I, but the path for the include files will be taken from the environment variable 'name'. Typing -EINCLUDE would produce the same results as -I alone.

-E[.] (DFT, LFT)

Specifies name(s) (-Ename) or file with names (-E@filename) of external or built-in functions. For LISP, this option is useful if GNU-Emacs Lisp source code is scanned to reduce the number of undefined functions listed in the output file. A list of GNU-EMACS (version 18.59) built-in functions is given with the file GNULISP.FCT. For DFT this option prevents output of built-in functions/ keywords. An example file with DBASE functions/ procedures is provided with DBASE.FCT.

-F (CFT, CST, DFT, FFT, JCT, LFT)

Use only ASCII characters for the call tree output instead of the DEFAULT semi graphic characters. This option is useful if the generated output file should be printed on a printer which does not support semi graphic characters like they are defined in the IBM character set. It can also be used to prepare the output file for

use in a WINDOWS application like MicroEMACS if there is no font with semi graphics available.

To view the semi graphic characters from the original output in Windows you have three choices:

- select a fixed size font that supports semi graphics (like Terminal)
- open SXT output file as 'MS-DOS text' and select fixed size font (like Courier)
- open SXT output file from Windows Write and convert to Write format, then select fixed size font (like Courier)

-FORTRAN77 (FFT)

Assume that all source files contain Fortran 77 code in fixed format.

-FORTRAN90 (FFT)

Assume that all source files contain Fortran 90 code in free format. By default only files with extension '.f90' are treated as Fortran 90 source code files.

-G[name] (CFT, CST, DFT, FFT, JCT, LFT)

Generate a database with the complete set of information about the processed sources. The additional parameter 'name' (path and filename) is used as an unique base name for the set of database files (up to 6 significant characters), the DEFAULT SXT NAME ('CXT', 'DXT', ...) is used if no name is specified. If 'name' ends with a (back-) slash, it is used as a path name. The generated database files (extension '.DBF') are dBASE compatible. There are two additional files created, one with the command line options (extension '.CMD') and one with a list of the source files (extension '.SRC') being use for database generation. They can be used as command line definition files with '\$' (command list) and '@' (file list).

As a result of the database generation you will find files named 'CXTxy.ext' (or other SXT names) respectively 'namexy.ext' (user defined 'name'), where 'x' will be 'F' ("function") for CFT, DFT, FFT and LFT, 'S' ("structure") for CST or 'C' ("class") for JCT, 'y' is replaced by an internally used character to mark the different database files and their contents.

The database contents is similar for every operating system it was generated except for LINUX: Under LINUX, the complete path name for each file does not have a drive letter. This means, that databases generated under LINUX cannot be used on DOS, Windows 95 / NT or OS/2 and vice versa, of course.

-GML[x] (CFT, CST, DFT, FFT, JCT, LFT)

This option generates a GML output file with function resp. data type call graph information. The GML output file is named "prog.GML" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT) by default unless the optional extension 'x' is used to

specify another filename. There are two special cases for 'x' which can be defined additionally. '-GML+' prints for each item the filename and line number where it is defined. For CST and JCT, '-GML*' prints for each structure, union and class its data type members.

GML stands for "Graph Modeling Language" and is used by Graphlet, a freely available tool for the automatic layout and displaying of complex graphs. Graphlet reads a GML specification file and visualizes the graph. If not all positions of nodes are fixed, the tool performs a graph layout using several heuristics as reducing the number of crossings, minimizing the size of edges and centering of nodes. There are versions for Unix as well as Windows 95 / Windows NT. Graphlet was mainly developed by the University of Passau, Germany. More information about Graphlet and GML (including executables, examples, documentation and source code) can be found at <http://www.fmi.uni-passau.de/Graphlet>.

-H[elp] (CFT, CST, DFT, FFT, JCT, LFT)

See option -?.

-HTML[x] (CFT, CST, DFT, FFT, JCT, LFT)

This option generates HTML (HyperText Markup Language) compatible output files which can be viewed with WWW (World Wide Web) Browsers like Netscape Navigator / Communicator or Microsoft Internet Explorer. The optional parameter 'x' is the name of the destination path, by DEFAULT the output is written to the current directory. This option can be combined with option -HTMLPREx to change the HTML filename prefix from its DEFAULT value (program name CFT, CST, ...).

The HTML output is done in parallel with the '.LST' output file, so all call tree options influence also the HTML output. The most important HTML output files are named "prog.HTM", "progNAMES.HTM", "progFILES.HTM", "progXREF.HTM" and "progSTAT.HTM" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT). Additionally CST and JCT produce the files "progCLASS.HTM" and "progOOM.HTM". All these HTML files can be viewed with every WWW browser.

Besides the call tree, cross reference, item name index, filename index and statistics information, several other HTML files are generated which provide additional functionality based on Netscape Navigator / Communicator extensions (frames, JavaScript, Plug-In's). The main entry to the advanced HTML output features is the file "progMAIN.HTM" (for 'prog' see above) which opens three frames, one frame as a display window, one frame with a lexicographical index ("progINDEX.HTM") and another frame with buttons ("progNAVIG.HTM") to select the information being displayed. The last HTML file ("progCONT.HTM") controls the output behavior.

For CST and JCT there are additional HTML files with the class inheritance tree (file "progCLASS.HTM", generated if option -b is set) and with class metrics (file "progOOM.HTM").

The HTML files are similar for every operating system they were generated except for LINUX: Under LINUX, the complete path name for each file does not have a drive letter. This means, that HTML files generated under LINUX cannot be used on DOS, Windows 95 / NT or OS/2 and vice versa, of course.

With Netscape Navigator / Communicator it is possible to locate items and view the related source code with a special Netscape Navigator Plug-In called NPSCC (file NPSCC32.DLL version 1.03 or higher). NPSCC32.DLL is part of the shareware package SCC Source Code Colorizer. There is also a LINUX plug-in named NPSCC.SO. See the SCC documentation for further information about the SCC Plug-In.

Because the HTML output files refer to each other by hypertext links there is no possibility to rename them. To handle multiple sets of HTML output files copy each set into a separate directory or use the optional 'x' parameter. To avoid unresolved links it is recommended that options -a (and -u, if available) are set to ensure that all items are listed. Note that national character sets are not handled properly. For additional information see HTML_RTf.DOC.

For best viewing results with a WWW client, select a fixed-size font e.g. Courier New or Courier. The SXT-HTML output was tested with the english versions of Netscape Navigator 3.01 (all features), Netscape Communicator 4.01 / 4.05 (all features) and Microsoft Internet Explorer 3.02 / 4.01 (no source code viewing). Under LINUX, Netscape Communicator 4.05 was used for test purposes. Additional tests were also made with Mosaic and Netscape browsers running on a SUN workstation. The latest versions of Netscape Navigator / Communicator can be found at '<http://home.netscape.com>', the Microsoft Internet Explorer can be found at '<http://www.microsoft.com>'.

-HTMLHELP (CFT, CST, DFT, FFT, JCT, LFT)

This option generates additionally to the HTML output also a Microsoft HTML Help project file. This HTML Help project file (file extension .hhp) can be used as input for the Microsoft HTML Help Compiler HHC.EXE to compile all related HTML files into one HTML Help file (file extension .chm). The big advantage of this procedure is that all HTML files are combined into one single file and that the file contents is highly compressed to save memory. However, to use this feature you need to install the Microsoft HTML Help Workshop, that can be downloaded via WWW or FTP from the Microsoft Homepage www.microsoft.com. You also have to install the Microsoft Internet Explorer 4.0 or higher to have all runtime components that are necessary for viewing HTML Help files.

-HTMLPREx (CFT, CST, DFT, FFT, JCT, LFT)

This option sets the HTML filename prefix to 'x' (default: program name (CFT, CST, ...)). It can be combined with option -HTML[x] to set the directory path.

-I[path] (CFT, CST, FFT)

This option enables the scanning of include files declared with `#include "..."` or `#include <...>` or with a similar syntax for FORTRAN. The required path for the include files is taken from the INCLUDE environment variable (DEFAULT BEHAVIOR) or can be user defined by 'path'. Paths defined with -I will be searched before any other paths taken from environment variables specified by -E or -P, so care should be taken with that option. Include paths can be given either absolute or relative. A relative path is always considered relative to the directory of the source file it is used with, not to the directory the analysis is started from or the analysis program is located.

Using the -I or -E option without -P allows the scanning of the source file and the included files without preprocessing. In that case an include file is handled as if it were a complete new file, this can lead to errors if a file inclusion is specified within a function or structure. Also preprocessor controls like `'#if ...'` are not evaluated and can lead to unexpected results. Remember this especially if you place special comment sections enclosed in `'#if 0 ... #endif'` blocks.

-I* (CFT, CST, FFT)

Declaring -I* ignores missing include files or errors with files which could not be opened due to compiler limits (message 'too many open files', see chapter PROGRAM LIMITATIONS). This is a 'quick and dirty' approach, but can sometimes be useful, if include file locations are unknown or inaccessible. However, the results will not be absolutely correct.

-I8.3 (CFT, CST, FFT)

This option enables the handling of include file names in DOS 8.3 format. Include file names which are longer than the 8.3 format (8 characters for name, 3 characters for extension) are truncated to 8.3 if the original include file was not found. With this new include file name a second file search is done. This option may be useful for projects with files which were ported from UNIX to DOS (truncated file names but probably original file names in `#include` statements) and are now used on Windows NT or Windows 95.

-Jcharset (CFT, CST, DFT, FFT, JCT, LFT)

Extend the DEFAULT character for identifier recognition with a user defined character set 'charset'. All characters must be specified within one -J option. The following default identifier character sets are used:

C/C++	a-z A-Z 0-9 _\$
DBASE	a-z A-Z 0-9 _
FORTRAN	a-z A-Z 0-9 _
	a-z A-Z 0-9 _\$ (with option -\$)
JAVA	a-z A-Z 0-9 _
LISP	a-z A-Z 0-9 +-.*/<=>!?:\$%_&~^

This option allows the programmer to use national character sets as they are common in Germany, Denmark, Sweden and other European countries.

-L[L][+] (CFT, CST, DFT, FFT, JCT, LFT)

Redirect the screen output to a file, called 'CFT.LOG' resp. 'CST.LOG'. If '+' is set, the output is both written to screen and redirected to the log file so that the output messages can both be viewed as they appear and later analyzed. Finally, -LL resp. -LL+ appends the output to an existing log-file, this can be useful if CFT and CST are run from a batch-file to catch all output.

-LIB[.] (CFT)

Specifies name(s) (-LIBname) or file with names (-LIB@filename) of library functions. Library functions are not listed as undefined functions in the output file and there are no warnings about missing prototypes or missing return types. Lists of C-library functions are given with the files MSVC15.FCT (MS Visual C++ 1.5) and GCC233.FCT (GNU-C 2.3.3).

-M (CFT, CST, FFT)

This option generates a source file/include file dependency table for every processed file. This table shows the dependent include files of a source file and can be used for a MAKE file. It is also useful to check if the included files are taken from the correct directories. If a file is included more than once, the number of inclusions will be displayed. Note that this option will only work correctly if options -P or -I are used, but not on preprocessed files. There may also be problems with files produced by source code generators like LEX/FLEX or YACC/BISON.

-N (CFT, CST, DFT, FFT, JCT, LFT)

Disable output file writing. This option can be useful if, for example, only a database (option -G) should be generated with CFT or CST and no output file is required. In that case the sometimes very time consuming process of output file writing is skipped. Note that for CST the writing of the byte offset file "CST_OFFS.C" will not be affected by this option.

-NOCPPCMT (CFT, CST)

Do not accept C++ comments '//...' in C source code. By default, C++ comments are recognized in C code since this is now a common feature of many popular compilers like Microsoft, Borland or GNU and will probably be also in the next C standard. However, this option can be used to ensure compatibility with C compilers which do not recognize C++ comments within C source code.

-NOCPPONCE (CFT, CST)

Disables the handling of '#pragma once' during preprocessing for GNU-C and MS Visual C++ 4.1 or higher. By default, for both compiler types '#pragma once' is handled during preprocessing to prevent multiple / repeated inclusions of the same include files for one source file.

-NOGNUCPP (CFT, CST)

Do not use the GNU C preprocessor for source code preprocessing. Since CXT 2.55 the default built-in C preprocessor for the Windows 95 / NT (command line and Windows user interface versions), LINUX and DOS386 versions is the GNU C preprocessor. Option -NOGNUCPP gives the possibility to use an alternative preprocessor which is integrated in CFT / CST since version 1.78. This original CFT / CST preprocessor is quite old and was designed to work on all platforms, even on MS-DOS, with very low memory requirements. It does very much disk access and holds only parts of the source and include files in memory and is therefore quite slow. Today, only the 16 bit versions for MS-DOS and Windows 3.1 use this preprocessor and have not integrated the GNU C preprocessor.

The advantage of the GNU C preprocessor is its enormous speed (its more than 30 times faster than the other integrated preprocessor!) and its better macro expansion capabilities according to the ANSI / ISO C standard. Therefore this preprocessor was chosen as the default one. The GNU C preprocessor needs very much memory because it reads all source and include files completely into memory. This and its better and newer internal program structure results in a very high execution speed. Due to the high memory requirements the GNU C preprocessor is currently only implemented for Windows 95 / NT, LINUX and DOS386.

However, there are some disadvantages with the GNU C preprocessor that should be considered:

- the output of warning and error messages cannot be customized with warning options, they are either on (warning level ≥ 5) or off (warning level < 5), the other preprocessor supports these warning levels,
- there are other / additional / missing warning and error messages compared with the other preprocessor,
- if unrecoverable errors are recognized, the GNU C preprocessor immediately stops execution with an internal exit or abort, there is no possibility to continue with the next source file, the whole program is finished,
- in the Windows user interface version the preprocessing cannot be stopped or aborted,
- the behavior for preprocessing may differ from the other preprocessor and lead to different results, especially for statistics values like number of included files, number of lines, number of code bytes or comment bytes, lines of code and so on,
- the include file relationships may differ due to the recognition and handling of already included files,
- the GNU C preprocessor produces memory leaks (there are very few calls to free() in the preprocessor source, preprocessing one source file can produce

- several megabytes of memory leaks), since it was designed as a stand-alone program that frees all memory on exit this was no problem, but for the integration into CFT and CST memory leaks are not acceptable for consecutive calls to the preprocessor, to avoid this a simple garbage collector was integrated which needs some additional time, but there is still no guarantee that no memory is wasted,
- the GNU C preprocessor has sometimes a UNIX-like behavior, especially with carriage return line feed.

The GNU C preprocessor as it is used here for CFT / CST is part of the GNU C / C++ Compiler and is developed by the FSF - Free Software Foundation, Inc., Boston, Massachusetts, USA. The complete source code is public available and can be found at <http://www.gnu.ai.mit.edu>.

-NODIGRAPH (CFT, CST)

Disable digraph translation during preprocessing (options -P). This option can be useful for processing some Microsoft specific code, because the Microsoft specific 'based-code' operator ';>' (introduced with MS-C 6.0) is also a digraph which translates to '|'. This may lead in some situations to 'unbalanced brackets'.

-NOINTR (FFT)

Do not recognize and display INTRINSIC functions in output call tree. Only subroutines and functions defined in the source code are displayed.

-NOLIB (CFT)

Do not display library functions defined with option -LIB in output call tree.

-NOPROT (CFT)

Do not display a warning message if a function is defined without previous prototype ("function definition used as prototype").

-NOREPLACE (CFT, CST)

Disable C++ keyword replacing. This option can be useful if older C++ code with identifiers which are now recognized as new C++ keywords is processed (e.g. the keyword 'bitor' is replaced with '|').

-NOUNSAFE (CFT, CST)

This option warns in case of errors with 'unsafe' macro expansions of the integrated preprocessor (option -P) which otherwise would abort execution with fatal errors (see PROBLEMS.DOC). This is only a 'work-around' that can lead to following errors.

-NOWARN72 (FFT)

Do not warn if characters are found beyond column 72. FFT can process lines up to 132 columns length. By default a warning is given if characters are found beyond column 72. This can be disabled with this option.

-O[..] (CST)

Specifies name(s) (-Oname) or file with names (-O@filename) of data types for which the calculation of structure/union sizes with byte offset information for every data type member should be performed. Additionally specifying -O+ sets a flag for the recursive collection of sub-structures during expansion which are displayed without specifying them by -O. This means that if a structure/union consists of members which are also structures or unions (and so on), it is not necessary to specify all these data type names with -O to enable them for byte offset calculation. Instead, you have to specify only the top most data type with -Oname and additionally -O+ to force CST to select all related sub-types for displaying. If -O+ is set but NO names are specified, ALL structures and unions will be used for byte offset calculations (the resulting file can be very big)!

As the result of this option, CST generates a source file called 'CST_OFFS.C' resp. 'CST_OFFS.CPP' (if option -C++ is set). This file needs some additional editing to declare necessary include files, data types, defines or pragmas before it can be compiled with the C compiler for which the file was generated (be sure to use the same includes!). The resulting executable prints for every structure/union member the byte offset relative to the beginning of the structure/union (decimal and hexadecimal) and the size of each member, the resulting structure/union size and also information whether a structure/union member has been aligned (= compiler dependent insertion of fill bytes before that member) or if the structure/union was padded with fill bytes at the end of it to align the size to a specific length.

To get these information and to perform the necessary calculations therefore, the source file can become very large and makes use of the C macro programming capabilities, which may lead in some rare cases to errors during the compilation due to the internal limitations of some C compilers.

The -O option is very useful if you need detailed information about structures or unions in case of error searching and debugging, especially for hardware debugging with an ICE (Integrated Circuit Emulator). It is also useful for finding out the differences in the internal layout of a structure/union in the case of porting C source code between different compilers and/or operating systems or if data structures are exchanged between different hardware platforms, for example via a data communication system. You can verify the expected structure/union layout and size made by the target compiler.

-P[name] (CFT, CST)

Run the integrated C preprocessor before the file scan. In this case the include path is taken from the INCLUDE environment variable (DEFAULT BEHAVIOR), from the

user defined 'name' environment and additional paths from -I and -E option are used. If special paths should be searched before the default paths, they must be specified by the -I path or the -E environment option and they must be placed on the command line before the -P option to be processed first. The -D, -U preprocessor defines and -T type and memory model and -B size information are also used, if defined. The path for the preprocessor output file can be specified by the -v option, otherwise the current working directory will be used (DEFAULT BEHAVIOR). If option -C++ is set, the macro '__cplusplus' will be predefined before preprocessing to enable C++ macros and C++ comment recognition.

If a fatal error occurs during preprocessing, the analysis of the file will be aborted and the next file in the queue will be processed (if 'out of memory' occurs the complete analysis is aborted!). For a description of the error format see option -W.

If you are using a compiler which is not supported by CFT and CST or the build-in preprocessing doesn't satisfy your needs because the results seem to be different from your preprocessor, you can preprocess the files you want to analyze with your own preprocessor and use these files as input for CFT and CST. To ensure that the filename and line number references are valid your preprocessor must insert #line directives into the source.

Preprocessing with another compiler can also be necessary if the integrated preprocessor has problems with your source, e.g. because the nesting level for include files is too high (see section PROGRAM LIMITATIONS) or it aborts during macro expansions (see PROBLEMS.DOC).

-Q[.] (CFT, CST, DFT, FFT, JCT, LFT)

Specify the name (-Qname) or a file with the names (-Q@filename) of those source files for which their functions/data types should be listed in the output call tree file (sorted by line numbers). The files are handled in the given -Q sequence. If additional -S options are specified, the -S items will be listed after all -Q options are done.

-R (CFT, CST, DFT, FFT, JCT, LFT)

By default, CFT and CST generate the hierarchy tree chart of the called function/data type ("CALLER: CALLEE relation", "WHO CALLES WHOM"). The -R option produces an inverted listing showing the callers/users of each function/data type. It generates the output as the function/data type hierarchy member list tree chart in reverse order as a list of calling items of the referenced basic item ("CALLEE: CALLER relation", "WHO IS CALLED BY WHOM"). This option is useful to get the relations between functions/data types and their callers/users.

-RATIONAL (CFT, CST, DFT, JCT, FFT, LFT)

This option generates so called 'Petal' files for Rational Rose (Windows), a CASE-tool supporting the Booch Object-Oriented Analysis and Design (OOAD) method.

The generated output files can be imported by Rational Rose to use the built-in capabilities for visualization, but in the case of the SXT programs (mis-) used to graphically visualize the calling relationships of functions resp. data types. The -RATIONAL option is a work-around for the missing graphical layout capabilities of the SXT programs (which some users have requested in the past) by using an external program for doing the missing features.

Two different types of description files are generated, one for class diagrams and one for finite state machine (FSM) diagrams. The class diagram files are named 'prog.PTL' and the FSM files 'prog_FSM.PTL' with 'prog' as the name of the SXT program being used. CST and JCT generate two additional files named 'CSTCLASS.PTL' / 'CSTCLFSM.PTL' resp. 'JCTCLASS.PTL' / 'JCTCLFSM.PTL' describing the class inheritance relationships in the same way. In the class diagrams, for functions (CFT, DFT, FFT, LFT) the USES-relationship was used to display the calling relationships (the classes are misused as functions), whereas for data types and classes the INHERITANCE-relationship is used.

If you have Rational Rose, you have to perform the following steps to get impressive results: Start Rational Rose, select a new model ('File' - 'New') and import the generated file ('File' - 'Import...'). If you imported a FSM description, a class diagram with one class symbol named 'CallGraph' (FSM) appears. Click on that symbol and choose 'Browse' - 'State Diagram'. Now select 'Tools' - 'Layout' to start the layout optimization function. As the result the graphical call tree of the source code analysis is displayed with each function/data type shown as a circle ('state') and the call relationship shown as an arrow ('transaction') from the calling to the called item, for classes from the superclass to the subclass. You can zoom into the diagram, print the results or incorporate the diagrams into your program documentation via Clipboard, e.g. into Word for Windows.

In case of a class description file, a class category with one category named 'Program' appears after file import. Double click on that symbol to open the associated class diagram and activate the layout function. As the result the graphical call tree of the source code analysis is displayed with each function shown as a class and the call relationship shown as a double line from the calling (circle) to the called function.

NOTE: The SXT programs cannot generate input for Rational Rose comparable to that generated by the C++ analyzer of Rational Rose, SXT just uses Rational Rose's printing capabilities!

The -RATIONAL option was tested with the Beta version of Rational Rose/C++ (Windows) 2.0. If you get warnings reading the files with an earlier (later) version of Rational Rose edit the file and reduce (increase) this value. The SXT programs currently set this value 35 for version 2.5. Note that Rational Rose needs even for small and medium sized projects some time to import the file and process the diagram layout. There is no guarantee that this will always be successful.

Rational Rose is a commercial product, for more information (demo versions, ...) contact Rational directly at

Rational, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951, U.S.A.
(Internet site: www.rational.com)

-RTF[x] (CFT, CST, DFT, FFT, JCT, LFT)

This option generates a RTF (Rich Text Format) compatible call tree and file contents list output file. The RTF output file is named "prog.RTF" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT) by default unless the optional extension 'x' is used to specify another filename. The RTF output is done in parallel with the '.LST' output file, so all call tree options influence also the RTF output. The RTF file can be imported by word processors which can handle RTF (e.g. Word for Windows). Another very useful feature is the possibility to generate a Windows Help file from the RTF output. This can be done by using the Microsoft Windows Help Compiler HC.EXE, HCP.EXE or HC31.EXE (or via the HC.BAT batch file) which compiles the .RTF file into a .HLP file (e.g. 'hc31 cft.rtf' generates cft.hlp). There will also be a Windows Help compiler project file generated, which can be used for RTF file compilation with compression to reduce the resulting help file size. The command is 'hc cft.hpj'. The help file can be viewed with WINHELP.EXE. It contains a searchable sorted list of all items (functions, types), a file contents list, a caller / callee cross reference list, statistical information and (only for CST) a class inheritance tree. Similar to the HTML output, hypertext links are embedded which allow jumps within the help file to item definitions. To avoid unresolved links it is recommended that options -a (and -u, if available) are set to ensure that all items are listed. For additional information see HTML_RTf.DOC.

-S[.] (CFT, CST, DFT, FFT, JCT, LFT)

Specify name (-Sname) or file with names (-S@filename) of functions/data types to search for and to dump if present (names are case sensitive). Only these items are listed in the output call tree file, all others are ignored. The items are listed in the given -S sequence. If also all the other items not covered by -S should be listed, the user can specify -S+ on the command line. The '+' parameter forces all remaining items to be printed after all the -S items have been printed. For DFT and FFT the item names are considered uppercase. By using -S on the command line, it is necessary to surround a data type name that consists of two words with double quotation marks like "struct _iobuf" to connect the two words. This is not necessary inside a list file, but there every search name must be on a separate line.

-TAGS[x] (CFT, CST, DFT, FFT, JCT, LFT)

This option generates a TAGS file with information for VI-like tagging of defined identifiers. The TAGS file is named "TAGS" by default unless the optional extension 'x' is used to specify another filename. Each entry in the TAGS file has the format

identifier<tab>file<tab>vi-search-pattern

where 'vi-search-pattern' is the line number in 'file' where 'identifier' is defined. As using the line number is a usual practice if 'typedef' definitions are included into the TAGS file this should be no problem. The generated TAGS files were successful tested with the MSDOS versions of VI (from Mortice Kern Systems Inc.) and the public domain VI-clone ELVIS, but should also work with other VI versions. Since VI originates from UNIX, the TAGS file contains only LF, not CR+LF, as line ending. I have done it this way to avoid possible problems with close-to-UNIX VI ports, but this may also lead to new problems if CR+LF is really needed. To change from DOS to UNIX styles and vice versa tools like DOS2UNIX or UNIX2DOS can be used. See also option -CTAGS for additional information.

-Tn (FFT)

Set the tabulator expansion size to 'n' (DEFAULT: 8 characters).

-Ttype,m (CFT, CST)

Use this option to set the compiler type for source code preprocessing to one of the following types:

- MSC51 Microsoft C 5.1
- MSC70 Microsoft C/C++ 7.0
- MSVC15 Microsoft Visual C++ 1.5
- MSVC22 Microsoft Visual C++ 2.2
- MSVC40 Microsoft Visual C++ 4.0
- MSVC41 Microsoft Visual C++ 4.1
- MSVC42 Microsoft Visual C++ 4.2
- MSVC50 Microsoft Visual C++ 5.0
- TC10 Borland Turbo C++ 1.0
- BC20 Borland C++ 2.0
- BC31 Borland C++ 3.1
- BC10OS2 Borland C++ 1.0 for OS/2
- WATCOMC100 Watcom C/C++ 10.0a
- GNUC222 GNU-C 2.2.2
- GNUC263 GNU-C 2.6.3
- GNUC272 GNU-C 2.7.2
- GNUC2721 GNU-C 2.7.2.1
- GNUC281 GNU-C 2.8.1
- I960 Intel 80960 iC960 3.0

The supported memory models are T(iny) (valid only for MSC70, MSVC15, TC10, BC20, BC31), S(mall), M(edium), C(ompact), L(arge), H(uge), 'L' is assumed as default if no model is specified. MS Visual C++ 2.2 / 4.0 / 4.1 / 4.2 / 5.0, Borland C++ for OS/2, GNU-C and Intel iC960 do not need a memory model because they compile really 32 bit code. The Intel iC960 compiler requires the definition of the 80960 RISC processor architecture which is one of KA, KB, SA, SB, MC, CA (default is KB). For the Watcom C/C++ compiler there are no memory options predefined due

to the wide range of possible targets, the user has to specify the necessary options by himself with options -B and -D.

This option causes several compiler dependent preprocessor macros (if they were known to me, however) to be defined before preprocessing starts. This option can only be used with the -P option, otherwise it has no effect.

If your compiler is not supported, you can perform the following steps: Find out which preprocessor defines are necessary (manual, help file) and declare them with option -D, then declare, depending on the selected memory model or processor architecture, the type sizes with option -B.

-U[.] (CFT, CST)

Specifies a predefined macro name (-Dname) or file with predefined macro names (-U@filename) to be undefined for preprocessing. Note that the default predefined macro names '___FILE___', '___LINE___', '___DATE___', '___TIME___' cannot be undefined. All other predefined names for the various compiler types can be undefined. Like for -D, the names are considered case-sensitive, but trigraph or digraph translation is not performed because the internal representation cannot contain them.

-USYSCMT (FFT)

Handle UNISYS inline comments '@...' in Fortran source code.

-V (CFT)

List prototyped functions which are neither called nor defined (option -a and -u). This option is useful to find unused function prototypes which could be probably removed from the source code.

-VCG[x] (CFT, CST, DFT, FFT, JCT, LFT)

This option generates a VCG output file with function resp. data type call graph information. The VCG output file is named "prog.VCG" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT) by default unless the optional extension 'x' is used to specify another filename. There are two special cases for 'x' which can be defined additionally. '-VCG+' prints for each item the filename and line number where it is defined. For CST and JCT, '-VCG*' prints for each structure, union and class its data type members.

VCG stands for "Visualization of Compiler Graphs" and is a freely available tool for the automatic layout and displaying of complex graphs. The VCG tool reads a VCG specification file and visualizes the graph. If not all positions of nodes are fixed, the tool performs a graph layout using several heuristics as reducing the number of crossings, minimizing the size of edges and centering of nodes. The VCG tool allows folding of dynamically or statically specified regions of the graph. It runs on X11, on XView and on Windows. The VCG tool was developed by Georg Sander and Iris

Lemke, both from University of Saarland, Germany. More information about VCG (including executables and source code) can be found at <http://www.cs.uni-sb.de/RW/users/sander/html/gsvcg1.html>.

-Wlevel (CFT, CST, DFT, FFT, JCT, LFT)

Set error and warning message level. Higher warning levels include lower ones. The DEFAULT level is always the highest supported warning level, possible levels are:

- 0 : all error and warning messages are suppressed except fatal errors,
- 1 : display serious errors or warnings,
- 2 : includes level 1 plus additional errors and warnings,
- 3 : includes level 2 plus errors/warnings/remarks,
- 4 : includes level 3 plus warnings about implicit declared functions and lacks of type or storage class.

The following levels affect only preprocessing (CFT and CST):

- 5 : includes level 4 plus warnings and errors during preprocessing (non-fatal errors and warnings during preprocessing are otherwise not displayed, preprocessor is running in 'silent mode'),
- 6 : includes level 5 plus remarks/slight warnings during preprocessing.

The output format for messages during file scan is

filename(line): error: description
filename(line): warning: description

and during preprocessing (warning levels 5 and 6)

preprocessor: filename(line): error: description source line
preprocessor: filename(line): warning: description source line

-XLAMBDA (LFT)

Recognize the LISP resp. SCHEME keyword 'lambda' for unnamed function declarations. By DEFAULT, 'lambda' is treated as a simple identifier.

-XSCHEME (LFT)

Assume SCHEME source code instead of LISP source code (DEFAULT). This means that functions are recognized by the 'define' SCHEME keyword instead of the 'defun' resp. 'defmacro' LISP keywords.

-Z[s] (CFT, CST, DFT, FFT, JCT, LFT)

Display every caller and member for each function/data type, 's' sorts by the number of calls (DEFAULT ORDER: lexicographical), this is an extension of the -c option. This option shows the relations in the following form:

List of parent functions/data types:

1. caller (reference #) <# of calls from>

...

n. caller ...

function/data type (reference #) <# of calls from parents, # of calls to children>

List of child functions/data types:

1. called member (reference #) <# of calls to>

...

m. called member ...

This compact form lists all callers and members with the number of their calls, recursions are detected and displayed. Note that this option can be extremely time consuming if the number of source files is very large!

-a (CFT, CST, DFT, FFT, JCT, LFT)

List every function/data type, also previously referenced functions/data types. This generates a complete list of every function/data type in lexicographical order with references to their first location.

-asmb[..] (CFT, FFT)

Specifies name(s) (-asmbname) or file with names (-asmb@filename) of assembler procedure begin keywords. This option can be used to specify additional keywords for the recognition of the beginning of assembler procedures. The specified names are considered case insensitive.

-asmc[..] (CFT, FFT)

Specifies name(s) (-asmcname) or file with names (-asmc@filename) of assembler procedure call keywords. This option can be used to specify additional keywords for the recognition of the calling of assembler procedures. The specified names are considered case insensitive.

-asme[..] (CFT, FFT)

Specifies name(s) (-asmename) or file with names (-asme@filename) of assembler procedure call keywords. This option can be used to specify additional keywords for the recognition of the ending of assembler procedures. The specified names are considered case insensitive.

-asmlower (CFT)

Convert assembler names in .ASM files to lower case.

-asmupper (CFT)

Convert assembler names in .ASM files to upper case.

-b (CST)

Display the C++ class inheritance relationships. This option generates two listings. The first one displays the complete C++ class hierarchy graph(s). The second one shows for each class first the superclasses from which the class inherits and the access restrictions (public, protected, virtual, ...) and second the subclasses which inherit from the given class, also with access restrictions. This option is useful to find out things like the class dependencies or multiple inheritance.

-b (FFT)

Handle backslash in FORTRAN character constant (DEFAULT is NO). This option is useful if non-standard UNIX-like character constants with backslash escape sequences (e.g. "\") are used, which are not handled by default and will therefore lead to errors.

-c[s] (CFT, CST, DFT, FFT, JCT, LFT)

Display the number of calls to each function/data type, 's' sorts by the number of calls (DEFAULT ORDER: lexicographical). Useful to find out which functions/data types are never called/used (maybe unnecessary and can be deleted) and which ones are the most frequently called/used (together with profiler results a subject for further optimization efforts). For FFT see also the description for option -CALL.

-calldf=x (CFT)

Define a function caller / callee relationship. This option can be used to specify additional calls from a calling function to a called function that are not recognized during normal file scan. This can, for example, happen if a function is not called directly but indirectly via a function pointer. These additional calls are inserted after the normal file scan so that they may appear as the last calls of the calling function in the call tree. Each caller / callee relationship can only be defined once, multiple calls are ignored.

The following syntax must be used for the definition:

`-calldf=calling-function,called-function`

An example for this option could be:

`-calldf=main,get_token`

Note that there is no space between the comma and the name of the called function, otherwise double quotation marks are necessary on the command line. There are no plausibility checks performed on this information, if any error is found the option is ignored.

The previous example defines a call from function 'main' to function 'get_token'. It is not necessary that either the calling or the called function are defined during file scan. In that case there will be no valid filenames and file locations for these functions.

-charwarn (CFT, CST, DFT, FFT, JCT, LFT)

Warning if unknown/illegal characters are found in the source code during analysis. By default, there is no now no longer such a warning.

-cmdline (CFT, CST, DFT, FFT, JCT, LFT)

Print the command line options at the beginning of the output file as a remark for the generation rules for that output file. All options and files specified on the command line, in the environment variable, the command list and the file list files are listed.

-daVinci[x] (CFT, CST, DFT, FFT, JCT, LFT)

This option generates a daVinci output file with function resp. data type call graph information. The daVinci output file is named "prog.daVinci" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT) by default unless the optional extension 'x' is used to specify another filename. There are two special cases for 'x' which can be defined additionally. '-daVinci+' prints for each item the filename and line number where it is defined. For CST and JCT, '-daVinci*' prints for each structure, union and class its data type members.

The generated output files are used by daVinci, a freely available tool for the automatic layout and displaying of complex graphs. DaVinci reads a daVinci specification file and visualizes the graph. There are only versions for various Unix OS available of which the LINUX version was used to test and verify the output generated by the SXT programs. DaVinci is developed by the Computer Science Department of the University of Bremen, Germany. More information about daVinci can be found at <http://www.tzi.de/~davinci/>.

-defmacro (LFT)

Recognize the 'defmacro' keyword in SCHEME source code.

-dirstat (CFT, CST, DFT, FFT, JCT, LFT)

Write directory statistics with information about number of files in directory, number of bytes, number of lines and other information.

-dn (CFT, CST, DFT, FFT, JCT, LFT)

Set the maximum function/structure/union nesting level for output generation to 'n' (DEFAULT: maximum value n = 999). This means that the request for displaying a deeper level will be rejected and the output call tree will be truncated at the given level.

-e[char] (CFT, CST, DFT, FFT, JCT, LFT)

Generate formatted ASCII text files with function / data type list, file list and directory list. All entries are separated by the optional 'char' character, if 'char ' is not defined, the tabulator character is used as DEFAULT SEPARATOR. If spaces are needed as separating characters, you have to write -e" ". Such prepared files can be used directly as input to other programs like word processors (e.g. MS-WORD for WINDOWS) or spreadsheet calculators (e.g. MS-EXCEL), for example for documentation purposes. The following files are created (similar names for other SXT programs):

CFTITEMS.TXT:

Contents: function name, return type, filename, line #, total # of function bytes, # of function comment bytes, # of function lines, # of control statements, # of brace levels

CSTITEMS.TXT:

Contents: data type name, filename line #

CFTFILES.TXT and CSTFILES.TXT:

Contents: filename, # of lines, file size in bytes, # of comment bytes, # of functions / data types, # of multiple defined items

CFTDIRS.TXT and CSTDIRS.TXT:

Contents: directory name, total # of files, total file size in bytes, total # of lines, total # of functions/ data types, total # of code bytes, code ratio in %

-f (CFT, CST, DFT, FFT, JCT, LFT)

Generate an output list in short form, only with the function/ data type names, no further description of the internal function/ data type elements.

-filetree (CFT, CST, FFT)

This option displays the file hierarchy tree for every source file. The look of the file hierarchy tree is similar to the function hierarchy tree. Unless option -M displays the 'flat' source vs. include file relationship, this option shows the real nested source file include file hierarchy tree. This option is useful to see how the files are related and nested. Note that this option will only work correctly if options -P or -I are used, but not on preprocessed files. There may also be problems with files produced by source code generators like LEX/FLEX or YACC/BISON.

-font@file (SXT WINDOWS VERSIONS ONLY)

Specifies a 'file' with font color description. This option overrides the default color and style values. A font file from the SCC Source Code Colorizer named 'SCCFONT' with the default color values is provided as an example. For best coloring results use a screen color palette selection with more than 256 colors like high color or true color.

The following color definition values are available:

- color_background
- color_comment
- color_keyword
- color_library
- color_preprocessor
- color_string
- color_character
- color_number
- color_parentheses
- color_braces
- color_brackets
- color_separator
- color_delimiter

Colors are defined with color_... = #rrggbb, where #rrggbb is the HTML syntax for color description (rr = red, gg = green, bb = blue). The colors range from #000000 (= black) to #FFFFFF (= white).

-funcdef=x (CFT)

Define a function name and its location in a source file. This option can be used to specify functions that are not recognized during normal file scan or if that file is not scanned, although the file does really exist.

The following syntax must be used for the definition:

`-funcdef="return-type function-name <complete-filename, line-number>"`

An example for this option could be:

`-funcdef="int get_token <d:\project1\src\scanner.c, 50>"`

Note that the double quotation marks are necessary on the command line to keep all things together as one option. There are no plausibility checks performed on this information, if any error is found the option is ignored.

With this option the user can specify by himself a function ("get_token") with its return type ("int") and the filename ("d:\project1\src\scanner.c") and line number ("50") inside this file where the function is defined. The filename must be absolute, this means including drive and directory. Otherwise it cannot be found later.

Functions defined with this option will be added to the scan results and therefore also written to the database and can be retrieved from there, for example for source code browsing.

However, there is one disadvantage: If files being declared that way and not really analyzed there will be no valid file information (size, lines, code ratio, ...). All these values are zero.

-g[name] (CFT, CST, DFT, FFT, JCT, LFT)

Read a previously generated database (see option -G). The additional parameter 'name' (path and filename) is used as an unique base name for the set of database files (up to 6 significant characters), the DEFAULT NAME 'CXT' is used if no name is specified (similar for other versions: DXT for DFT, FXT for FFT, ...). If 'name' ends with a (back-) slash, it is used as a path name. Every source file will be tested for changes of file creation time and file size and a warning message will be given to inform the user. It is currently not possible to update the results of such files.

The 32bit Windows 95 and Windows NT versions of CFT and CST support with option -g also the reading of Microsoft Browse Info Files (extension '.BSC') which can be generated with the Microsoft Visual C++ Compiler. If you want to read a '.BSC' browse info file you must specify the complete filename with drive, path and file extension '.BSC' with option -g, otherwise the CFT / CST own database files with extension '.DBF' are assumed.

A Microsoft Visual C++ '.BSC' Browse Info File is a database that contains information from the compilation and linking process with all function and data types used in a specific project. CFT and CST can read these database files and handle the results in the same way as they do it with their own analysis results. You can also save these results with option -G.

However, there are some restrictions due to missing information that is not available from the browse info files or that is incomplete or not exact:

- CFT does not display function return types,
- CFT does not display the complete call graphs with all calls, each called function is displayed only once for the calling function even if called multiple (this is similar to option -l1),
- CFT does not display the called functions in the correct sequence, the function calls are displayed in arbitrary order,
- CFT may truncate very long function names (especially method names in C++ class templates or names with nested name spaces),

- CST does not display the complete data type contents, each type is displayed only once,
- CST does not display the correct data type contents, the data types are displayed in arbitrary order,
- CST does not display the correct variable names of data type members,
- CFT / CST do not display the include file relationships,
- the filenames are probably incomplete if relative paths were used for compilation, this problem can be solved in some cases by specifying include paths with option -I to allow CFT / CST to search itself for the files,
- CFT / CST do not display the correct values for many useful statistical values like defined functions / types per file, number of control statements, code size, code ratio, function size or nesting level, LOC,
- the reported line numbers for functions and data types may not be exact, they differ usually with +/- one line.

The reading of '.BSC' database files is quite slow because all information must be read before any other processing can be done and the information retrieval is quite complex. Also, each file found in the database is scanned (if present) to get the missing information about line numbers and code / comment size, which takes also much time, especially for large projects.

However, it is possible to convert the '.BSC' contents to the SXT specific database format (extension '.DBF') by using the options

`-g<BSC-filename> -G<optional SXT database name> -N`

to read a BSC-file (option -g) and write a SXT database (option -G) without any other output (option -N). This SXT specific database can be read much faster. For example, a 9 MBytes large '.BSC' database took more than 10 minutes to read while the '.DBF' converted database was read in less than 10 seconds.

Note that this additional option -g functionality was only tested with '.BSC' files generated by MS Visual C++ 5.0, there is no guarantee that this will work with earlier or later Browse Info File versions.

-h[elp] (CFT, CST, DFT, FFT, JCT, LFT)

See option -?.

-ignoflow (CFT, CST, DFT, FFT, JCT, LFT)

Ignore internal buffer overflow and try to re-synchronize the source code scanner. This option is useful if during source processing the error message "internal buffer overflow" occurs that leads to a program abort. This error message is usually caused by very long strings that exceed the internal buffer length. To avoid the program abort, option -ignoflow ("ignore overflow") can be used.

-iname (CFT, DFT, FFT, LFT)

Ignore function member 'name' in output call tree. It will not be displayed and will be skipped instead if found as a function member. For DFT and FFT the item names are considered uppercase. This option can be useful if, for example, functions are used only for test purposes and are of no further interest for the user and should be ignored in the output call tree. It is also possible to specify library functions so that the output call tree contains only user defined functions.

-inctrace (CFT, CST, FFT)

Trace inclusion of files during preprocessing and analyzing. Displays information about the file and file location where another file is included, the include nesting level and when an include level is left. Useful as an additional progress information.

-l[1] (CFT, DFT, FFT, LFT)

Option -l lists every called function inside a function body only once in case of repeated consecutive calls (DEFAULT: display every occurrence). If a function is called more than once without any other call in between, there will be only one reference of that function call in the output call tree. Option -l1 lists every called function inside a function body exactly once. This option results in shorter output files and gives a general overview.

-lbpfiler# (CFT, CST, DFT, FFT, JCT, LFT)

This option sets the metric limit "bytes per file" to the value #. Files which exceed this values are written to the ouput file "progLIMIT.LST" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT).

-lbpfunc# (CFT, DFT, FFT, LFT)

This option sets the metric limit "bytes per function" to the value #. Functions which exceed this value are written to the ouput file "progLIMIT.LST" (where 'prog' is one of CFT, DFT, FFT or LFT).

-lcpfiler# (CFT, CST, DFT, FFT, JCT, LFT)

This option sets the metric limit "comment portion per file" to the value #. Files which have less comment portion than the given value are written to the ouput file "progLIMIT.LST" (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT).

-lcpfunc# (CFT, DFT, FFT)

This option sets the metric limit "control statements per function" to the value #. Functions which exceed this value are written to the ouput file "progLIMIT.LST" (where 'prog' is one of CFT, DFT or FFT).

-leptype# (CST, JCT)

This option sets the metric limit "elements per data type" to the value #. Data types which exceed this value are written to the output file "progLIMIT.LST" (where 'prog' is one of CST or JCT).

-lfpfile# (CFT, DFT, FFT, LFT)

This option sets the metric limit "functions per file" to the value #. Files which exceed this value are written to the output file "progLIMIT.LST" (where 'prog' is one of CFT, DFT, FFT or LFT).

-llpfile# (CFT, DFT, FFT, LFT)

This option sets the metric limit "lines per file" to the value #. Files which exceed this value are written to the output file "progLIMIT.LST" (where 'prog' is one of CFT, DFT, FFT or LFT).

-llpfunc# (CFT, DFT, FFT, LFT)

This option sets the metric limit "lines per function" to the value #. Functions which exceed this value are written to the output file "progLIMIT.LST" (where 'prog' is one of CFT, DFT, FFT or LFT).

-ltpfile# (CST, JCT)

This option sets the metric limit "data types per file" to the value #. Files which exceed this value are written to the output file "progLIMIT.LST" (where 'prog' is one of CST or JCT).

-mtype (CST, JCT)

Start the data type tree chart with data type 'type' (-mtype). If -m+ is specified, the output starts with the topmost data type, this is the data type which is in the highest level of the hierarchy tree chart. The default output is in lexicographical order of the displayed data types. Useful if a selected structure/union should be displayed at the beginning of the output file.

-m[name] (CFT, FFT)

-mname (DFT, LFT)

Start the function call tree dump with function 'main' (CFT -m), 'PROGRAM' (FFT -m) or 'name' (-mname), name is case sensitive. If -m+ is specified, the output starts with the topmost function, this is the function which is in the highest level of the hierarchy call tree. If this option is not set, the default is lexicographical order of the displayed functions.

Usually, the complete function call tree should start with the 'main' function so that every subfunction is a (sub-) member of 'main'. This option is useful for windows

programs to start the output with the initial 'WinMain' function (-mWinMain) instead of 'main'. It can also be used to start the output with the initial assembler start-up code being executed before the 'main'-function is called.

-n[a] (CFT, CST, DFT, FFT, JCT, LFT)

Display the most critical function call path respectively display the data structure/union with the maximum nesting level. The modifier 'a' is used to display every function/structure with its users/callers (DEFAULT: display only deepest call path). This option helps to determine the complexity of the function call/data structure hierarchy and finds recursions over several call/nesting levels. Note that for functions the maximum call path being displayed is the result of the static source code analysis. During program execution the call path can be even deeper if functions are called indirectly with function pointers or similar mechanisms.

-noundef (CFT, DFT, FFT, LFT)

Ignore undefined items (functions, subroutines) in output call tree. Display only those which are really defined in the analyzed source code.

-ofile (CFT, CST, DFT, FFT, JCT, LFT)

Write the generated analysis results to file 'file'. DEFAULT BEHAVIOR: The filenames are 'prog.LST' (where 'prog' is one of CFT, CST, DFT, FFT, JCT or LFT). Possible overwriting of an existing output file with the same name other than the default one will be detected and prompted for user reconfirming. The resulting output file is an ASCII text file with no formatting characters which can be printed with every printer, viewed and/or edited with every text editor and taken as input to word processors, for example for documentation purposes.

-oom (CST, JCT)

Write object-oriented class metrics to a file named "xxxOOM.TXT" (xxx = CST or JCT). The meaning of the OO class metrics being used are described in the chapter "Source Code Metrics".

-qn (FFT)

Set the number of continuation lines to 'n' (DEFAULT: 19 lines). The number must be in the range from 0 to 99.

-tabwarn (CFT, CST, DFT, FFT, JCT, LFT)

Warning if tabulator characters are found in the source code during analysis.

-time (CFT, CST, DFT, FFT, JCT, LFT)

Print runtime information about the times consumed for source analysis, preprocessing, output dump, database reading and writing and for other miscellaneous jobs plus the total time. The results are given in the format MINUTE:SECOND.MILLISECOND. Note that the timing values may not be correct if they are the result from an analysis that uses database input.

-touch (CFT, CST, DFT, FFT, JCT, LFT)

If the file information in a database concerning the file date and the file size is out-of-date, it will be updated if the database is read (option -g) and option -touch is set. Useful to avoid the warnings e.g. with the BRIEF macros. However, this option should only be used if the contents of the related source files has not changed. This option does not reprocess and analyze the related sources, there will be no update of the results!

-u (CFT, DFT, FFT, LFT)

List undefined functions. These functions are probably library functions, defined in other files which have not been scanned or are unresolved externals found by the linker.

-vpath (CFT, CST, FFT)

Set a specific path for the intermediate precompiler output file. 'path' can be a path name or a drive letter with path name, e.g. 'c:\tmp' or 'g:\' (not 'g:' without '\', this is an error). This option is useful to speed up execution speed when the intermediate file can be stored on a RAM-disk so that file access to the precompiled file is much faster than on a hard disk. Tests have shown that the analysis time can be reduced for more than 25%. Environment variables like 'TMP' or 'TEMP' to set the path for temporary files are not evaluated.

-y (CFT, CST, DFT, FFT, JCT, LFT)

Display cross link list of files which contain referencing and referenced functions/data types of functions/data types of a specific file. This option shows the relations in the following form:

```

    1. referencing file
    ...
    n. referencing file
file
    1. referenced file
    ...
    m. referenced file
```

This option is useful if you want to find out the file relationships. This information can be used to isolate specific files from a project, e.g. library files. It is also useful if you

want to separate a function and want to know which other files are needed because they contain called functions. Note that there may be problems with files produced by source code generators like LEX/FLEX or YACC/BISON.

-z (CFT, CST, DFT, FFT, JCT, LFT)

Generate a function/data type call cross reference table. For every function/data type the location of its definition (file, line) and a complete list of its calls/references, sorted by files and line numbers is given in the following form:

1. function/data type (reference #) [file #], line #

[file #]: line #, ...

...

2. ...

...

The functions/data types are displayed in lexicographical order. At the end of the section is the cross reference file list. Note that this option can be extremely time consuming if the number of source files is very large!

-\$ (FFT)

Recognize \$ as identifier character, not as delimiter. Useful for source code written for VAX (VMS)- or IBM (MVS)-Fortran.

-? (CFT, CST, DFT, FFT, JCT, LFT)

Shows the command line syntax and gives a short, but complete help information about the accepted commands and their syntax in thematically sorted order.

-?? (CFT, CST, DFT, FFT, LFT)

Shows the command line syntax and gives a short, but complete help information about the accepted commands and their syntax in lexicographically sorted order.

COMMAND LINE FILES

cmdfile (CFT, CST, DFT, FFT, JCT, LFT)

Specifies a file with (additional) command line options. This might be useful if the command line would be too long because of the number of options and files declared or if you are usually using the same options which can then be stored in a command file. The initial '\$'-character is required to mark a command file, for LINUX the initial character must be '=' because '\$' is used by the shell. Inside a command file it is possible to put comment lines, they start with a '#' sign in the first column and are skipped until end-of-line.

filelist (CFT, CST, DFT, FFT, JCT, LFT)

A file with a list of source file(s) to be processed, wildcards are accepted. The list file should have every file on a single line. The rules for files containing assembler code and path translation are described above. The initial '@'-character is required to mark a file list file. The '+' sign for subdirectory processing is also possible inside the file list file. Inside a file list file it is possible to put comment lines, they start with a '#' sign in the first column and are skipped until end-of-line.

[+]file (CFT, CST, DFT, FFT, JCT, LFT)

The name of a source file to be processed. More than one file can be specified on the command line. The default assumption for the given files is that they contain C source code. Assembler source files are only recognized by the file extension '.ASM' (80x86 MASM/TASM) and '.S' (Intel 80960, GNU, UNIX).

The '+' sign indicates that, starting from the given directory, all subdirectories should be searched recursively for the given filename search pattern. This addition is useful if a large software project is divided into several modules with separate subdirectories for each module. In that case only the starting (root-) directory with the requested filename search pattern must be specified to search the current directory and all subdirectories.

If the filename or the include file specification inside a file contains a relative path ('./', '..\', '../' or '..\') it will be translated into an absolute path starting from the current working directory respectively in case of include files depending on the path of the parent file. Command line wildcards '*' and '?' are possible and will be accepted.

REMARKS ON USING OPTIONS

NONE OF THE DESCRIBED OPTIONS IS PREDEFINED SO IT IS UP TO THE USER HIMSELF TO CUSTOMIZE HIS PREFERRED PROCESSING BEHAVIOR AND OUTPUT STYLE BY ADDING CONTROL OPTIONS NEEDED THEREFORE.

This assumption seems to be the best way to give users the freedom of making their own decisions about the features they really need for doing their work, although the large number of options may be confusing for beginners. Therefore, take some time to learn about CFT and CST and their features, read this manual carefully and make your own experience with this software.

If you start using CFT and CST you can take the following command lines as examples and try other options to get a feeling for what they are useful and how they affect the output.

```
CFT -m -u -M -P -T<type> -cs -Cs -n -Zs -G <file[s]>
CST -M -P -T<type> -cs -Cs -n -Zs -G <file[s]>
```

It is possible to declare more than one source file, command file and list file on the command line. In that case they will be processed in the order they appear. Files and options can be placed in mixed order on the command line, there is no

recommended order for them because all options (also those inside command files!) will be processed before any source files are scanned.

The maximum command line length for DOS is 127 characters, so this is a system dependent 'natural' limit for the options and filenames being declared. If you have more items to declare, place them into command list files and file list files, which do not have such limitations.

Options can also be defined by the environment variables CFT and CST like

```
SET CFT=...  
SET CST=...
```

To separate single options in the environment string, spaces are required. See also the description for the -D option for remarks on environment variable definitions.

The rules for the interpretation of options is

- if defined, all options in the environment variables CFT (for CFT) or CST (for CST) will be taken,
- the command line options and the option files will be interpreted in the order they appear.

If an option is declared different more than once then previous declarations will be overwritten by the newer one.

If options are represented by a single character with no additional optional values possible like -a or -u, they can be grouped together with a single leading '-' in front like '-auM', which is the same as '-a -u -M'. The last option however, can have additions, for example '-auMmWinMain' which can be evaluated to '-a -u -M -mWinMain'. If an option can have an additional parameter, the parameter must be specified without a space between the option character. Leaving this space means that no additional parameter is given for this option.

Filenames being composed of drive letter (not for LINUX), directory name, filename and file extension, in the following referred simply as 'path name', are treated by some special procedures to force a unique style of their internal representation:

- path names are considered case insensitive under DOS, Windows 95 / Windows NT and OS/2, so there is no difference in upper case / lower case / mixed case path names, the path names remain unchanged as they are specified by the user or defined within source files,
- path names are considered case sensitive under LINUX, so that it makes a difference to use upper or lower case characters for path names (a file search for 'c*.c' is different to 'C*.c'), the path names remain unchanged as they are specified by the user or defined within source files,

- path names containing './', '\.', '../' and '..\' (so called 'relative paths') are expanded and transformed into 'absolute paths' (including drive letter (not for LINUX), directory and filename and file extension),
- the recommended directory delimiter is '/' (UNIX-style), if a '\' (DOS-style) is recognized in a path name, it will be replaced by '/',
- path names are always expanded and transformed into the default style

<DRIVE LETTER>:<DIRECTORY PATH>/<FILENAME>

to get a unique representation for every filename that must be handled during processing,

- filenames are operating system dependent:
 - DOS: maximum length: 12 characters (8.3 format)
 - Windows 3.1: maximum length: 12 characters (8.3 format)
 - Windows NT: maximum length: 255 characters
 - Windows 95: maximum length: 255 characters
 - OS/2: maximum length: 255 characters
 - LINUX: maximum length: 255 characters
- (for Windows 95 / NT, OS/2 and LINUX, the complete path name itself, including the filename, can have a maximum of 255 characters)

If you want to perform database generation (option -G) for different projects, you are responsible to separate them and avoid overwriting of existing databases. This can be done either by giving the databases different names so that the database files can be placed all in the same directory, or every database must be written into its own directory. If you want to access the databases be sure to use the correct name and / or path, also within the BRIEF or MicroEMACS editors.

COMMAND LINE EXAMPLES

1. CFT -m -u *.c

This program invocation of CFT processes all files with the extension ".c" in the current directory and generates an output file starting with the "main"-function (option -m) for the output tree. All functions will be shown in lexicographical order (-a), also undefined ones (-u).

2. CFT -mWinMain -aMP -TMSC70,L -ld: -cs -Cs -na -ve:\ -C++ *.c ..*.c *.cpp

This invocation is similar to the one described above with some extensions. The source files from the current (*.c, *.cpp) and from the parent (..*.c) directory, they will be preprocessed (-P) with MS-C 7.0 defines for large memory model (-TMSC70,L), the include file path will be taken from the environment variable "INCLUDE" (default for -P) and the path "d:" (-ld:) will also be searched for. The precompiler output is stored in path "e:" (-ve:\). C++ extensions and keywords will be recognized if they occur (-C++). The output will start with the "WinMain"-function (-mWinMain). There will be a sorted call statistic (-cs) and a function summary for every scanned file (-Cs). The critical function call path for all functions will be calculated and displayed (-na) and the included files of every source file will be shown (-M).

3. CST -S"struct _test" *.h -W2 -C++

Start CST to scan all files in the current directory with extension ".h" for data types. The output should be done for the data type 'struct _test' (-S"struct _test"). The warning level is set to "2" (-W2).

4. CFT y.c -R -Dmain=main_entry z.c -P x.c

Start CFT to produce a reverse calling tree (-R) of the functions found in the files "x.c", "y.c" and "z.c" in the current directory. The files will be preprocessed (-P) before file scan, the name "main" will be replaced by "main_entry" during preprocessing (-Dmain=main_entry).

5. CST \$cst1.cmd \$cst2.cmd -ve:\tmp @cstfiles +*.h -olist.v1a

This invocation of CST receives its options from the command files "cst1.cmd" and "cst2.cmd" and stores the preprocessor output in path "e:\tmp" (-ve:\tmp). The files being processed are defined in the source list file "cstfiles" and on the command line by "+*.h". The "+*.h" file specification searches the current directory and all subdirectories for files with the extension ".h". The output file will be named "list.v1a" (-olist.v1a).

6. CFT -a -PGNUINC -TGNUC222 -M c:\gnu\src*.c c:\gnu\src*.s -d10

CFT scans all files with extension ".c" and ".s" in the directory "c:\gnu\src". They will be preprocessed with an include file path defined in environment variable "GNUINC" (-PGNUINC) for compiler type "GNUC222" (-TGNUC222). The output contains all functions (-a) and a list of all included files for every source file (-M). The output tree will be truncated if the nesting level is higher than 10 (-d10).

7. CST *.c

CST processes all files with extension ".c" in the current working directory. There are no options specified, so only the options set by the environment variable 'CST', if present, will be used to customize the program execution. As an example the command line options used in example 6. can be defined as environment variable CST by 'SET CST=-aMKPGNUINC -TGNU222 -d10'.

8. CFT -a -PI960INC -TI960,KB *.c *.s

CFT scans all files with extension ".c" and ".s" in the current directory. They will be preprocessed with an include file path defined in environment variable "I960INC" (-PI960INC) for compiler type "I960", 'KB' architecture (-TI960,KB). The output contains all functions (-a).

9. CFT -R -M -gproj40 -Gproj41

CFT reads the database named 'proj40' (-g) and produces as output the reverse function call tree (-R), the (include) file interdependencies (-M) and a new database named 'proj41'.

10. CST -g -Gnew -N

CST reads the default database (-g) and produces as output another database named 'new' (-Gnew). No other output file is generated (-N).

11. CST -N -Otest -O+ test.h

CST reads the file "test.h", generates no output file (-N), but a byte offset calculation file for data type 'TEST' (-Otest) and its enclosed type members (-O+).

8 MISCELLANEOUS

8.1 SOURCE CODE METRICS

8.1.1 INTRODUCTION

The SXT source code analysis tools provide many different source code metrics which can be used as information about the complexity of the analyzed source code. In general, every value that can be obtained from the source code analysis by using a clearly defined rule can be regarded as a metric. Therefore metrics are, for example, the number of files in a project, the size of a file, the number of functions within a file, the lines of code (LOC) within a file, the number of included files, the include file nesting level and so on. Based on these directly obtained metric values there are other metrics which are derived by calculating average or ratio values. Such average values can be used to find out if certain values are within given limits or which item of interest deviates from the average metric value. Another group of derived metric values are the total number of each metric value which gives an impression about the total size of a project.

Source code metrics provide only abstract values according to the rules being used to get them. Whether they are useful or not depends on how they are interpreted and if there are similar values available with which they can be compared to make a decision. The SXT programs provide many metric values for the analyzed source code but they do not give any interpretation of them. Its up to the user himself to use them for decisions and conclusions about his software development and his development process.

8.1.2 BASIC SOURCE CODE METRICS

The following description gives an overview about the basic source code metrics that are provided as results of the analysis. Note that not all of them are generated from each SXT program since some of them are language specific.

Metrics for each file:

- file size
- number of lines
- number of functions / data types inside file
- number of included file
- total number of file inclusions (with multiple inclusions)
- scanned file size (with include files)
- scanned number of lines (with include files)
- code size (excluding comments)
- code ratio (code size / file size)
- lines of code (LOC)
- LOC ratio (LOC / number of lines)
- average number of lines per function

- average source bytes per function
- average comment bytes per function
- average bytes per line per function
- average number of control statements per function
- average brace levels per function
- average number of calls to functions
- average number of calls from functions

Metrics for all files:

- average number of functions / data types per file
- average number of bytes per file
- average number of lines per file
- average number of included files

Metrics for each function:

- number of lines
- source bytes
- comment bytes
- bytes per line
- number of control statements
- number of brace levels
- number of calls to this function
- number of calls from this function

Metrics for all functions:

- number of functions
- number of function definitions
- number of overloaded functions
- number of multiple defined functions
- number of undefined functions
- maximum call tree depth
- average number of control statements
- average number of brace levels

Metrics for each data type:

- number of lines
- number of elements
- number of substructures
- number of uses

Metrics for all data types:

- number of data types
- number of data type definitions
- number of multiple defined data types
- maximum data type nesting depth
- average number of elements
- average number of substructures

Metrics for complete project:

- total number of files
- total number of functions / data types
- total number of source files
- total number of include files
- total source file size
- total number of include file size
- total scanned file size
- total scanned number of lines
- average number of included files
- total program code size
- total program comment size
- average code / file size ratio
- total LOC
- average LOC

8.1.3 OBJECT-ORIENTED SOURCE CODE METRICS

The SXT programs CST and JCT provide additional information about object-oriented source code metrics for C++ and Java as described in the article "Automated Metrics and Object-Oriented Development" by J. Bansiya and C. Davis (more information at <http://indus.cs.uah.edu>), published in the December 1997 issue of Dr. Dobb's Journal (<http://www.ddj.com>). The so called QMOOD metrics described there plus some additional source code metrics that might be useful can either be viewed in a special list view window of the SXT windows versions, in an own HTML-page or are written to a text file if option -oom is set. Following is a short description of the used abbreviations for the object-oriented source code metrics and their meaning. For a better understanding of these object-oriented source code metrics and their use for software development read the Dr. Dobb's Journal article. Note that not all of these metrics can be provided for Java since some of them are C++ specific.

Class specific metrics (given for each class):

MFA	Measure of Function Abstraction Ratio of the number of methods inherited by a class to the total number of methods accessible by members in the class
MAA	Measure of Attribute Abstraction Ratio of the number of attributes inherited by a class to the total number of attributes in the class
DAM	Data Access Metric Ratio of the number of private and protected attributes to the total number of attributes declared in the class

OAM	Operation Access Metric Ratio of the number of public methods to the total number of methods declared in the class
DOI	Depth of Inheritance Length of the inheritance path from the root to a class
NOC	Number of Children (subclasses) Count of the number of immediate children (subclasses) of a class
NOA	Number of Ancestors (superclasses) Count of the number of distinct classes (superclasses) that a class inherits
DAC	Direct Attribute Based Coupling Direct count of the number of different class types declared as attributes inside a class
NOM	Number of Methods Count of all methods defined in a class
NPrivM	Number of Private Methods Count of the number of private methods defined in a class
NProtM	Number of Protected Methods Count of the number of protected methods defined in a class
CIS	Class Interface Size (Number of Public Methods) Count of the number of public methods defined in a class
NOperM	Number of Operator Methods Count of the number of operator methods defined in a class
NOI	Number of Inline Methods Count of the number of inline methods defined in a class
NOP	Number of Polymorphic (Virtual) Methods Count of the number of polymorphic (virtual) methods defined in a class
NPM	Number of Parameters per Methods Average of the number of parameters per method in a class
NOD	Number of Attributes Count of the number of attributes in a class
NAD	Number of Abstract Data Types Count of the number of user-defined (abstract) data types used as attributes in a class
NRA	Number of Reference Attributes Count of the number of pointers and references used as attributes in a class
NPrivA	Number of Private Attributes Count of the number of private attributes in a class
NProtA	Number of Protected Attributes Count of the number of protected attributes in a class
NPA	Number of Public Attributes Count of the number of public attributes in a class
CEC	Class Entropy Complexity Complexity of a class based on the occurrence of different types in a class definition

System level metrics:

DSC	System Size in Class Count of the total number of classes
NOH	Number of Hierarchies Count of the number of class hierarchies (root class with derived classes)
NIC	Number of Independent Classes Count of the number of standalone classes that are not derived from other classes and are not inherited by other classes
NSI	Number of Single Inheritance Count of the number of classes that use single inheritance
NMI	Number of Multiple Inheritance Count of the number of classes that use multiple inheritance
NAC	Number of Abstract Classes Count of the number of classes with abstract methods

Derived system level metrics:

ADI	Average Depth of Inheritance Average depth of inheritance of all classes, computed by dividing the sum of all inheritance path lengths by the number of classes
AWI	Average Width of Inheritance Average depth of children (subclasses) per class, computed by dividing the sum of the number of children over all classes by the number of classes
ANA	Average Number of Ancestors Average number of classes from which a class inherits (superclasses)
MFA	Average MFA for all classes
MAA	Average MAA for all classes
DAM	Data Access Metric Ratio of the number of private and protected attributes to the total number of attributes declared in the class
OAM	Operation Access Metric Ratio of the number of public methods to the total number of methods declared in a class

Average class external metrics:

(derived from previously calculated values)

DOI	Average DOI for all classes
NOC	Average NOC for all classes
NOA	Average NOA for all classes
DAC	Average DAC for all classes

Average class internal metrics:

(derived from previously calculated values)

NOM	Average NOM for all classes
NOPrivM	Average NOPrivM for all classes
NOProtM	Average NOProtM for all classes
CIS	Average CIS for all classes
NOOperM	Average NOOperM for all classes
NOI	Average NOI for all classes
NOP	Average NOP for all classes
NPM	Average NPM for all classes
NOD	Average NOD for all classes
NAD	Average NAD for all classes
NRA	Average NRA for all classes
NPrivA	Average NPrivA for all classes
NProtA	Average NProtA for all classes
NPA	Average NPA for all classes
CEC	Average CEC for all classes

8.2 OUTPUT DESCRIPTION

This section gives an overview about the files being generated by CFT and CST and the interpretation of the results. Different files are produced as output depending on the options being set by the user. Usually, if -N is not set, all information are written to the default output file (CFT.LST, CST.LST, ...) or to the file specified by the -o option. The internal structure of these files and their meanings are described below. If database generation is enabled with option -G, several files are produced. They all have a common database name to identify the files that are related with a project. The file extension '.DBF' marks the dBASE compatible database files, the file with the extension '.CMD' contains the command line options and the file with the extension '.SRC' contains all source files that were processed. For further information refer to the corresponding section in the syntax description.

8.2.1 CFT OUTPUT

The output file is divided into several sections. Some of the sections listed are generated by default (-), others are optional (o) and only displayed if they are enabled by a command line option. Also, the default sections can be customized to produce the desired output. The sections generated for CFT are (in the order they appear):

- file header
- function call tree/called-by hierarchy listing (-R, -a, -m, -f, -dn, -V, -l)
- function summary
- multiple defined functions and their location (only if detected)
- overloaded functions and their location (only if detected)
- o undefined functions (-u)
- o function call statistics (-c[s])
- o function caller/member relations (-Z[s])
- o function call cross reference table (-z)
- o critical function call path (-n[a])
- o source file - include file dependency (-M)
- o function tables for source files (-C[s])
- file information summary

Each function is displayed like:

```
int test() (1) <DMPCA> <TEST.C, 100>
```

with the following meanings

- int : function return type
- test() : function name
- (1) : function reference number
- <DMPCA> : found as (one or more of)
D = definition,

- M = macro,
- P = prototype,
- C = function call,
- A = assembler function
- <TEST.C, 100>: filename, line number

The line number is the line where the function definition block starts with its initial '{' and not the line where the function name resides. I think that this is the best solution because it is the point where we go really inside the function block. This convention is also used by source level debuggers which point on the line with the opening brace on function entry.

8.2.2 CST OUTPUT

The output file is divided into several sections. Some of the sections listed are generated by default (-), others are optional (o) and only displayed if they are enabled by a command line option. Also, the default sections can be customized to produce the desired output. The sections generated for CST are (in the order they appear):

- file header
- data structure call tree/called-by hierarchy listing (-R, -a, -m, -f, -dn)
- data type summary
- multiple defined data types and their location (only if detected)
- o data type call statistics (-c[s])
- o data type caller/member relations (-Z[s])
- o data type call cross reference table (-z)
- o maximum data type nesting (-n[a])
- o source file - include file dependency (-M)
- o data type tables for source files (-C[s])
- file information summary

Each data type is displayed like:

```
struct _test (1) <BSUCE> <TEST.C, 90> <TEST.C, 60>
```

with the following meanings

- struct _test : type specifier
- (1) : reference number
- <BSUCE> : data type (one/none of):
 - B = basic type (void, char, int, ...),
 - S = struct,
 - U = union,
 - C = class,
 - E = enum

- <TEST.C, 90> : filename, line number of type definition (only printed if necessary)
- <TEST.C, 60> : filename, line number of basic type definition

The two locations for the data type can occur if the data type is first defined and later assigned via 'typedef' or by '#define' (if -P is not set) to another data type name:

```
test.c: ...
line 60: struct xyz {...};
...
line 90: typedef struct xyz struct _test;
...
```

Their definition is on different lines but both data type names refer to the same data structure.

Like the convention used for functions, the line number is the line where the structure, union, enumeration or class type definition block starts with its initial '{' and not the line where the type name resides.

8.2.3 OUTPUT INTERPRETATION

Besides the hierarchical structure chart of the function and data type relationships, the resulting output contains several useful information about the program which can be used for optimization, reuse or maintenance purposes. Identifying the most frequently called functions is a good way to find candidates for further optimization. Low-level functions with many callers but no called subfunctions are ideal for reuse. Functions with no callers may be obsolete if the function is also not called via function pointers, and can be discarded therefore. The chance to find errors in complex functions with many lines of source code, many called functions and a lot of control statements is much bigger than in simple functions.

8.3 TOOLS FOR DATABASE PROCESSING

To access information stored in a database, the following utilities are available for the SXT programs:

CFTN	C Function Tree Navigator
CSTN	C Structure Tree Navigator
DFTN	DBASE Function Tree Navigator
FFTN	FORTRAN Function Tree Navigator
JCTN	JAVA Class Tree Navigator
LFTN	LISP Function Tree Navigator

They can be used to recall the filename and line number of a specific item (function or data type) from the database. If the requested item is found in the database, it will be displayed with its location where it is defined or where it is found for the first time if there was no definition found during processing.

As an additional feature editors like BRIEF 3.0/3.1, QEDIT 2.1/3.0, MicroEMACS 3.11 or CodeWright 5.0 can be invoked directly with the information to open the target file and to move the cursor to the line where the searched item is located. For BRIEF there are several macros available to perform searching inside the editor. A new edit window with the file at the location of the requested item will be opened if the search was successful. Also both MicroEMACS editor versions for DOS and WINDOWS are supported. For LINUX, EMACS and XEMACS can be used. Some of these actions are also possible for QEDIT, with slight limitations due to the macro programming capabilities.

Other user programmable editors which should be able to work with CFTN and CSTN are e.g. CRISP (BRIEF compatible), EPSILON, ME, KEDIT, Multi-Edit, JED, GNU-EMACS DOS-ports like DEMACS or OEMACS, the Microsoft editor M or integrated development environments like Borland IDE or Microsoft PWB (this list may not be complete). You can try to integrate CFTN and CSTN into these systems by using the BRIEF, QEDIT or MicroEMACS macro files as examples for your own integration development.

The version numbers for the editors mentioned in this manual indicate those versions for which the described capabilities have been tested.

PRECOMPILED SOURCE FILES

Sometimes, if the precompiler option -P was used to process the C/C++ source files related with the database, the results of searches seem to be wrong. This can happen if an identifier in the source code is in fact defined as a macro and has been exchanged during preprocessing so that the resulting source processed by the analyzer is different from the original source and the cursor will point to an obviously wrong location or the search will fail. An identifier which is in fact a macro name is

unknown and not accessible after precompiling. It is also possible that a function being used in the original source code could not be found in the database. The reason is that the function is in fact a 'function like' macro and was replaced during preprocessing. If different named macros are defined equal, a search for an item may point to another location than the requested. If the -P option is not set, the same item can have several 'alias'-names due to macro defining. If the source code contains explicit #line numbers, searching for a specific line may also fail. Keep these exceptions in mind for a correct interpretation of the results when using the database.

IMPORTANT NOTICE

Recalling information from the database may not be valid if files being processed were edited and changed after the database generation has been performed. Errors can result like pointing to wrong files and/or lines if source lines have been deleted or inserted, failed searches if names have changed or failed accesses to files which may have been renamed, moved or deleted. To avoid these errors, a consistency check for the file creation date/time and file size will be performed by the recall programs. If inconsistencies are recognized, the user will be informed that the database is not up-to-date and should be updated by processing the source files again.

NOTICE TO DEVELOPERS USING THE DATABASE FROM THEIR OWN PROGRAMS:

There is absolutely no guarantee that the internal database structure will remain unchanged in future versions. The internal database structure may be changed by the author without prior notice if this is necessary to extend the program functionality.

COMMAND LINE SYNTAX DESCRIPTION

SYNTAX: **CFTN** [options] pattern
 CSTN [options] pattern
 DFTN [options] pattern
 FFTN [options] pattern
 JCTN [options] pattern
 LFTN [options] pattern

OPTIONS

-Eeditor

Specifies the editor command line for option -e, overwrites the default and the environment values. See the section about environment variables for further information about the required format.

-F

Print all filenames which are related with the database. This option is useful to get a complete overview about all files of the project.

-a

Print all function/data type names. Useful to generate a list of items, for example as input to other programs.

-B

Same as -a, but prints additionally the internal database record number. Used by BRIEF macros.

-bform

Run search in batch-mode, this means that, if the requested item was found, the location will be displayed on a single line as "file name line number" (DEFAULT STYLE), otherwise there will be no output that the search failed. The output style can be changed by specifying 'form' to overwrite the default style. Like for option -E you can specify the exact locations where the filename and line number should be inserted by defining a format string with %s and %d (See also the section about environment variables). For example, the format to generate a command line for invoking BRIEF, QEDIT, MicroEMACS, Codewright oder EMACS / XEMACS would look like

cstn -b"b -m\"goto_line %d\" %s"	(BRIEF)
cstn -b"q %s -n%d"	(QEDIT)
cstn -b"me -G%d %s"	(MicroEMACS)
cstn -b"cw %s -G%d"	(CodeWright)
cstn -b"emacs +%d %s"	(EMACS)
cstn -b"xemacs +%d %s"	(XEMACS)

This option gives you a great flexibility in generating an output for your own purposes, for example to write a batch file or for further use in other programs.

-e

If the requested item is found, an editor will be invoked to display the file containing the requested item. There are three different ways to specify the editor command line (evaluated in that order):

- use option -E,
- define the environment variables CFTNEDIT, CSTNEDIT or CXTNEDIT (resp. similar ones for the other SXT packages),
- if nothing is specified, BRIEF as the default editor (if present) for DOS / Windows 95 / Windows NT will be invoked with the filename and line number of the item to move the cursor to its location. Under LINUX, EMACS will be used as default editor. Ensure that the PATH environment variable is set correctly, including the path for the BRIEF directory.

-fname

Use 'name' as base name (path and filename) for database files. It is also possible to use environment variables (CFTNBASE, CSTNBASE, CXTNBASE, ...) for the definition of the database names. If -f and environment variables are not set, a DEFAULT NAME will be used (see also option -G from the SXT syntax description). This allows the use of different databases, for example, generated for different projects. See also the section about environment variables for further information.

-r#

This option prints the location for a selected item with matching pattern and record number #. This option requires -b. Used by BRIEF macros.

-Ritem

Print a cross reference list of every occurrence of 'item' with complete filename and line number.

-Dfile

Print a list with the contents of 'file'.

-o[name]

Print output to file 'name'. If 'name' is not specified, DEFAULT NAMES are used: CFTN.OUT resp. CSTN.OUT, DFTN.OUT,

pattern

The item to search for in the database. This can either be a function name (CFTN) or a data type name (CSTN). There are three different ways of searching depending how 'pattern' is given:

pattern	exact search,
pattern*	the beginning of the item must match with pattern
*pattern	a substring must match with pattern

If the item to search for consists of more than one word (contains spaces), the search pattern must be 'quoted' like "struct _jobuf" to ensure that these words are interpreted as single pattern.

RETURN VALUES

The following values are returned to DOS or the calling program to report the result of the database search:

- 100 searched item not found,
- 101 searched item found,
- 102 searched item found, but the source file may have been changed (creation date and/or file size are not equal) since the creation of the database (database is not up-to-date).

The returned value can be used to decide what action should be done for different results, for example, if the database is not up-to-date.

ENVIRONMENT VARIABLES

CFTNEDIT, CSTNEDIT, CXTNEDIT:

The editor to invoke can be defined either by option `-e` or by defining the environment variables `CFTNEDIT` (for `CFTN`), `CSTNEDIT` (for `CSTN`) or the commonly used variable `CXTNEDIT` (for both `CFTN` and `CSTN`) with the format string of the editor of your choice. The format string can be used to specify the place where the filename and the line number should be inserted to give additional information to the editor. Use `%s` for the filename and `%d` for the line number. For example, the invocation of the default editor `BRIEF` could be defined like

```
SET CFTNEDIT=b -m"goto_line %d" %s
SET CSTNEDIT=b -m"goto_line %d" %s
SET CXTNEDIT=b -m"goto_line %d" %s
```

where `'b'` is the `BRIEF` editor, `'-m'` specifies the macro being invoked when `BRIEF` starts, the macro name `'goto_line'` with `'%d'` as the place to insert the line number and `'%s'` as the place for the filename. Note that this example cannot be used on the command line with `-E` option because of the quotes. It is possible to change the order of `%d` and `%s` if another editor is used.

Here are additional configuration examples for other popular editors (examples are given for `CFTN`, similar for `CSTN`):

EDIT (MS-DOS 5.0):	SET CFTNEDIT=edit %s or -E"edit %s" or SET CFTNEDIT=edit or -Eedit
VDE 1.62:	SET CFTNEDIT=vde %s or -E"vde %s" or SET CFTNEDIT=vde or -Evde
QEDIT 2.1/3.0:	SET CFTNEDIT=q %s -n%d or -E"q %s -n%d"
MicroEMACS 3.11:	SET CFTNEDIT=me -G%d %s or -E"me -G%d %s"
CodeWright 5.0:	SET CFTNEDIT=cw %s -G%d
EMACS:	SET CFTNEDIT=emacs +%d %s
XEMACS:	SET CFTNEDIT=xemacs +%d %s

The described notation allows the user to customize `CFTN` and `CSTN` with his preferred editor and to perform additional actions during invocation. If your editor supports macro programming like `BRIEF` you are free to write your own macros to do similar things like the `CXT.CM` macro given for `BRIEF 3.0/3.1` does. I think this is the

most flexible way to give users control about this option and to help them working with their preferred programming environment and development tools.

CFTNBASE, CSTNBASE, CXTNBASE:

These environment variables can be used to specify the name of the database. Similar to the editor environment variables, CFTNBASE and CSTNBASE are related to CFTN and CSTN and CXTNBASE is used for both. For example, to specify the database 'proj1' located in directory 'd:\develop\projects' type

```
SET CFTNBASE=d:\develop\projects\proj1
SET CSTNBASE=d:\develop\projects\proj1
```

for a separate definition or

```
SET CXTNBASE=d:\develop\projects\proj1
```

for a common definition of the database name.

COMMAND LINE EXAMPLES

1) **CFTN ***

Displays all functions in lexicographical order with their return types, filenames and line numbers. Gives a short overview about all functions being found.

2) **CSTN -e ***

Edit all data types in lexicographical order, use default or by environment variable CSTNEDIT or CXTNEDIT defined editor.

3) **CFTN -fproject1 -Evde -e main**

Search database named 'project1' for function 'main' and edit with editor 'vde'.

4) **CSTN -b "union REGS"**

Search for data type 'union REGS' and display, if found, the filename and line number

5) **CSTN -e -E"q %s -n%d" -fcft tmbuf**

Search database 'cst' for data type 'tmbuf' and invoke, if found, the editor 'q' (QEDIT 2.1/3.0) with the filename and line number

6) **CFTN -e -E"xemacs +%d %s" main**

Search database 'cft' for function 'main' and invoke, if found, the XEMACS editor with line number and filename

SEARCHING INSIDE BRIEF (Version 3.0 / 3.1)

This feature is one of the most powerful enhancements for the BRIEF editor and offers the user full control over the complete source code of software projects no matter how big they are and how many files they include. It extends the BRIEF editor to a comfortable hypertext source code browser and locator system. The browser allows its user to find and read various important program constructs like functions and data types in several files simultaneously and moving between them. The complete project with several source and include files appears as if it were a 'whole-part'. The browser helps the programmer to learn about the existing program structures and supports him in developing new and maintaining existing code. The programmer can use the generated output files CFT.LST or CST.LST (or the one he created with the -o option) to walk along the hierarchy tree chart and to select from there the function or data type that should be displayed in detail.

The following features are implemented as macros:

- searching for a specific item, tagged or marked
- building menus of all defined items
- building menus of all references to a specific item
- building menus of all processed files
- building menus of all items defined in the current file
- searching for a specific item cross reference number
- changing the database name

Every function and data type can be accessed with just a keystroke by moving the cursor on it ("tagging") and executing a macro to locate the item and zoom into the file where it is defined. The user does no longer have to remember the filenames and locations where the functions and data types are defined nor does he have to change the files, directories and drives to access the files manually.

It is possible to build interactive dialog menus with all functions or data types in lexicographical order and to select an item to display. This is very useful to get a quick overview about all accessible functions and data types of the whole project. It is also possible to build an interactive dialog menu with all filenames in lexicographical order which are stored in the database and to select one file to open for edit. Other menus are available for file contents lists and item cross references. All information to perform these actions are stored in the databases generated by processing the files related with the project.

To invoke CFTN and CSTN inside BRIEF, the macro file CXT.CM must be loaded (with <F9> CXT.CM), which makes the implemented macros available. These macros are

MACRO NAME KEY ASSIGNMENT (defined in CXTKEYS.CM)

cft	Shift F1
cftmenu	Shift F2

cftxrefmenu	Shift F3
cftxrefmenuagain	Shift F4
cftdefmenu	Shift F7
cftfilemenu	Shift F8
cftfind	Shift F11
cftbase	Shift F12
cst	Ctrl F1
cstmenu	Ctrl F2
cstxrefmenu	Ctrl F3
cstxrefmenuagain	Ctrl F4
cstdefmenu	Ctrl F7
cstfilemenu	Ctrl F8
cstfind	Ctrl F11
cstbase	Ctrl F12
cxtbase	Alt Tab
cxtsearchxref	Ctrl Tab
cxthelp	<unassigned>

This macro key assignment list is also available within BRIEF as a help screen which can be invoked by the macro 'cxthelp'. The CXT help information is not part of the BRIEF help system because this would need modifications of the original BRIEF help files.

Instead of loading the file CXT.CM and typing the macro names manually, you can load the macro file CXTKEYS.CM which performs automatic loading of the CXT.CM file if any of the above listed macros is invoked with a hot-key. To simplify working with this package, the CXTKEYS.CM macro file also contains key assignments for the macros. These hot-keys offer a "point and shoot" hypertext like feeling. The macro source file CXTKEYS.CB contains the source code for CXTKEYS.CM so that you are able to make changes like the key assignments for your personal needs or to move the initialization function to the BRIEF start-up macro file (For further information about BRIEF macros see the BRIEF manuals). To load these macros and to execute CFTN and CSTN, which are invoked from inside BRIEF, be sure to set the directory path correctly. It is also necessary to allow access to the macro file DIALOG.CM which contains the functions for dialog menu building and processing.

A search can be started by simply moving the cursor on the item to search for or by marking a block with the item (necessary if search pattern contains more than one word like 'struct xyz') and then running one of the following macros (or press hot-keys):

<F10> cft (function search)
 <F10> cst (data type search)

It is also possible to type the name of the item to search for manually. To do this you must run one of the following macros:

<F10> cftfind <item> (function search)
<F10> cstfind <item> (data type search)

If the search was successful, a new window with the file containing the item will be opened and the cursor will be placed at the line where the item is located. If inconsistencies have been detected, the user will be informed. If the requested item or the source file containing the item is not found, a message will be given. The macros for building the function and data type dialog menu are

<F10> cftmenu (function menu)
<F10> cstmenu (data type menu)

You can scroll through the entries and select an item which should be displayed. To access databases other than the default ones, there are two ways to change the base names:

- Set the environment variables CFTNBASE, CSTNBASE or CXTNBASE (see description above). By loading the macro file CXT.CM these variables will be used for initialization.
- To change the base names from inside BRIEF, there are three macros to do this. They overwrite the initial values given by the environment variables:

<F10> cftbase change base name for function search
<F10> cstbase change base name for data type search
<F10> cxtbase change both CFT and CST base name

With these features it is possible to set default values for the database files or to change between different databases without leaving BRIEF which gives the user a maximum of flexibility. You can display a menu list with all source files being scanned for the database by typing

<F10> cftfilemenu (CFT file menu)
<F10> cstfilemenu (CST file menu)

With this feature you can get a quick overview about all files related with the database. Other menu driven options concern the displaying of all cross references to a specific item (see macro 'cst' for information about marking) with the macros

<F10> cftxrefmenu (CFT cross reference menu)
<F10> cftxrefmenuagain (show previous menu again)
<F10> cstxrefmenu (CST cross reference menu)
<F10> cstxrefmenuagain (show previous menu again)

and the displaying of a file contents list for the actual source file with the macros

<F10> cftdefmenu (CFT file menu)

<F10> cstdefmenu (CST file menu)

To search for the first appearance of a specific cross reference number like '(123)' in a CFT or CST output listing file, move the cursor to the reference number and type

<F10> cxtsearchxref (search cross reference)

The macro extracts the complete number and searches for its first occurrence by starting from the beginning of the output file. With this macro you can move quickly from any reference to its initial description.

All the above described macro functions are defined in the BRIEF macro file CXT.CB. These macros make extensive use of the several options of CFTN resp. CSTN, which are described earlier in detail.

SEARCHING INSIDE QEDIT (2.1 and 3.0)

The popular shareware editor QEDIT with its macro programming capabilities allows, like the BRIEF editor, the searching of functions and data types from inside the editor. The following examples for QEDIT macros act, with slight limitations, like the BRIEF macros 'cft' and 'cst':

CFT function searching, assigned to <SHIFT F9>:

```
#f9 MacroBegin MarkWord Copy Dos 'cftn -b ' Paste '>tmp' Return Return EditFile  
'tmp' Return AltWordSet MarkWord Copy DefaultWordSet EditFile Paste Return  
EditFile 'tmp' Return EndLine CursorLeft MarkWord Copy Quit NextFile GotoLine  
Paste Return
```

CST data type searching, assigned to <SHIFT F10>:

```
#f10 MacroBegin MarkWord Copy Dos 'cstn -b ' Paste '>tmp' Return Return  
EditFile 'tmp' Return AltWordSet MarkWord Copy DefaultWordSet EditFile Paste  
Return EditFile 'tmp' Return EndLine CursorLeft MarkWord Copy Quit NextFile  
GotoLine Paste Return
```

These QEDIT macro definitions can be placed into the 'qconfig.dat' configuration file and added to 'q.exe' with the 'qconfig.exe' configuration utility (For additional details about QEDIT macro programming see the QEDIT documentation). The two macros perform the following actions: mark the current word, execute the CFTN or CSTN database search for the marked word via dos and redirect the output to file 'tmp', read target filename from 'tmp' and open target file, read line number from 'tmp' and go to the selected line.

These macros are working almost similar to those used from BRIEF, but they have some limitations in their functionality due to the limited capabilities of the QEDIT macro programming language:

- there is no error check for a correct cursor location,
- the searched item must always be a single word like 'main' or 'size_t', a combined pattern like 'struct iobuf' cannot be searched,
- there is no error check if the search was successful or failed or the database is not up-to-date,
- if the target file is the same as that from which the search started and other additional files are also open (QEDIT ring buffer), probably a wrong file will be accessed,
- the name of the database cannot be changed, the searches are performed either with the default database or those defined by the environment variables.

SEARCHING INSIDE MicroEMACS (Version 3.11, DOS & WINDOWS)

The latest editor which is now supported with macros for database access is MicroEMACS 3.11. The macro file is named CXT_ME.CMD and should be placed in the MicroEMACS directory. This macro file works with the DOS and the WINDOWS version of MicroEMACS 3.11.

The following macros are available:

- | | | |
|---|---------|--|
| - | cft | function search for tagged item |
| - | cst | data type search for tagged item |
| - | cftmark | function search for marked item |
| - | cstmark | data type search for marked item |
| - | cftfind | function search for user defined item |
| - | cstfind | data type search for user defined item |
| - | cftfile | list of all CFT files |
| - | cstfile | list of all CST files |
| - | cftbase | set CFT database name |
| - | cstbase | set CST database name |
| - | cxtbase | set both CFT and CST database name |

They can be invoked by loading the macro file CXT_ME.CMD with

ESC CTRL+S CXT_ME.CMD

and running the macro with

ESC CTRL+E <macro name>

If the macros are used with the MicroEMACS WINDOWS version, you may have to change the DOSEXEC.PIF file, which is part of the MicroEMACS 3.11 distribution package. During the CXT macro execution, the shell command may stop after execution and waits for the <return> key pressed to continue. To avoid this interruption, you can enable it by editing the PIF file and select "Close window after execution". The environment variables CFTNBASE, CSTNBASE and CXTNBASE

are used in the same way as in the BRIEF version. Key-assignments to macro procedure names are not performed, if you prefer hot-keys, you are free to do this for yourself.

In the MicroEMACS WINDOWS version, however, the user accessible macros can be integrated into the "Miscellaneous" pull-down menu (thanks to the incredible macro programming capabilities of MicroEMACS!). To view the generated output file with its semi graphic frames, change the font type and select for example the 'TERMINAL' font from the OEM font list which supports semi graphic characters.

8.4 INTEGRATION INTO PROGRAM DEVELOPMENT ENVIRONMENTS

Invoking CFT and CST directly from inside editors or integrated programming environments (IDE) and displaying the results can be a very useful feature during program development. With advanced IDE's like that of Borland C++ or Microsoft PWB this is an easy task.

The Borland IDE has in its system menu a section with 'transfer items. It contains programs that can be invoked from inside the IDE like TASM or GREP. To add CFT and CST as new entries you have to go to the OPTIONS menu and open 'TRANSFERS...'. Choose a free entry in the table and select EDIT. A window will open with 3 edit lines. In first line called 'Program Title' you must write 'C~FT' resp. 'C~ST' as the name being displayed in the transfer section. The '~' prepends the hot-keys 'F' and 'S'. In the second line called 'Program Path' you must write 'CFTIDE' resp. 'CSTIDE', maybe with the complete path, if necessary. 'CFTIDE' and 'CSTIDE' are batch files which perform the invocation of CFT resp. CST together with the necessary options. These batch files are part of the CXT package, you can change the options defined there if you need other ones. In the third line called 'Command Line' you must write the macro commands '\$EDNAME \$NOSWAP \$CAP EDIT'. These macros transfer the filename in the current edit window (\$EDNAME) to the batch file, suppress window swapping (\$NOSWAP) and capture the processing results in an own edit window (\$CAP EDIT). The last step is to save these entries, then the integration is completed and CFT and CST can be used as if they were built-in functions. The processing results are shown in an edit window which can be scrolled, resized or moved. By adding CFT and CST to the IDE it is much easier for the programmer to use these tools.

8.5 EXECUTION SPEED

All SXT programs are disk storage based programs because the source and include files, the intermediate precompiler file and the output file must be read from and written to hard disk. This means that the execution speed depends at first on the speed of the physical storage medium and not (only) on the speed of the CPU.

During program execution with preprocessing (option -P), most of the time will be consumed to preprocess the given input files and the related include files and to generate the preprocessor output file. The scanning for functions or data types takes only a small amount of time. The function/data type relations are computed while the output is generated and written to disk, there is no precomputing necessary.

The function for critical call path/nesting level detection depends only on the number of functions or structures and not on the call/declaration nesting complexity. The execution time grows linear with the number of items (functions/structures) to process and is very fast!

Be aware of the fact that the processing of a large number of files can take quite a long time (from several minutes up to hours on lower performance machines!), especially if option -P for preprocessing is enabled.

The generation of the output file and writing to disk can also take some time if the number of items to display is large and the nesting structure is complex. If the number of items and source files is very large, one of the most time consuming options is the function/data type file reference (option -z) and caller/member reference (option -Zs). The writing and reading of the database files (options -G and -g) takes also some time due to the large number of different information. The same applies to the various output formats (HTML, RTF, RATIONAL, VCG, GML, DOT, daVinci) which are very time consuming.

Don't panic if there seems to be no disk access for a longer time, the reason is just that there may be time consuming computations and that the output will be buffered internally to reduce the number of disk accesses and therefore speed up the output!

To speed up the SXTWIN program execution you should switch off the message output and analysis progress information (filename and line number) with the 'Info' menu to avoid time consuming screen updates. The program can also be run 'iconized', this also avoids screen updates.

For more detailed information about the program efficiency see appendix 4.

8.6 PROBLEMS

8.6.1 FREQUENTLY ASKED QUESTIONS (FAQ)

ARE THERE ANY RESTRICTIONS IN THE USE OF THE ANALYSIS RESULTS?

No restrictions for REGISTERED users, they can use the results for all purposes like program documentation, customer information or debugging as long as a notice about the name and copyright of the used SXT program is given.

WHY IS NO OS/2 IPF SOURCE CODE FOR OS/2 INFORMATION FILES GENERATED?

The OS/2 IPF functionality, comparable with the MS-Windows Help system, has limitations which are not acceptable for SXT programs. According to the OS/2 3.0 online help information, it is not possible to generate a single IPF source file larger than 64K and a single source line cannot have more than 255 characters. Both limits are regularly exceeded by the SXT programs.

WHY ARE SEVERAL DATABASE FILES FOR EVERY PROJECT GENERATED?

Separating the analysis items (identifier names, filenames, relationships, ...) of one project into several closely related database files is the best way to achieve minimum storage requirements and to optimize disk usage. This way of storage has no redundancies compared to storage in a single database file.

WHY IS THERE NO CROSS REFERENCE FOR VARIABLES INCLUDED?

This would need much additional memory and slows down the analysis process. There would also be a lot of multiple defined names in different contexts to be managed if several files are analyzed. A lot of tools exist which perform this task quite good e.g. the integrated compiler environments or other source code analyzers.

WHY ARE CFT AND CST NOT COMBINED IN ONE PROGRAM?

Historical and practical reasons: the CFT development was started before CST and both programs are optimized for their own special purposes. Combining them would complicate them and slow down the analysis process. Also the memory requirements would grow.

8.6.2 TROUBLE SHOOTING

This section contains information about problems and the reasons which may occur during the use of SXT programs. It is recommended that users should read the complete documentation to have an overview about the features before they start using the programs and run into any unexpected troubles. See also the chapter about 'PROGRAM LIMITATIONS' and the document 'PROBLEMS.DOC'.

UNEXPECTED RESULTS WHILE RUNNING THE SXT COMMAND LINE TEXT MODE VERSIONS UNDER WINDOWS 3.1

The 386 versions cannot run under Windows 3.1, they are using the CPU exclusive and can therefore not co-exist with Windows, only the real mode versions can. In Windows enhanced mode (virtual 386 mode), the real mode versions cannot run simultaneously in several independent DOS-windows if they are working in the same directory or use the same temporary directory, because the temporary intermediate files may have the same names and will conflict due to multiple concurrent accesses. This may also happen if the same files are scanned or included. The use of SHARE.EXE is recommended to catch such file access conflicts, see your DOS manual for information about the installation of SHARE.

MICROEMACS FOR WINDOWS SEEMS TO HANG DURING DATABASE ACCESS AND DOES NOT RETURN

The reason is usually quite simple: The shell call to DOS through DOSEXEC.PIF waits for a keystroke to continue execution and to return to WINDOWS. You may change this behavior by editing the DOSEXEC.PIF file (see MicroEMACS section for further information).

A PROGRAM CANNOT BE EXECUTED

The program path is not specified in the environment variable PATH, the programs are not yet installed in the specified directory, attempt to start a 386 protected mode version on a 80286 (or lower) computer.

EXECUTION STOPPED WITH MESSAGE "OUT OF MEMORY"

An attempt to allocate memory has failed. Try to remove unnecessary memory resident TSR programs and/or use the protected mode versions if you have an 80386 or higher. If this message happens for the protected mode versions, there is not enough free disk space for the swap file. Set the temporary directory, defined by the 'TMP' resp. 'TEMP' environment variable, to another drive with more free disk space.

WRITING THE OUTPUT FILE TAKES A LONG TIME

A large number of information must be handled, slow CPU and/or hard disk. Use option -v to redirect intermediate files to a faster RAM-disk (if such is present).

THE BRIEF MACROS CANNOT BE EXECUTED

The macro file is not loaded, other macros with the same names or assigned keys already exist.

THE BRIEF OR MICROEMACS MACROS CANNOT BE LOADED

The path to the macro file location must be specified when loading the macros, if they are not in the default directory for the editor.

THE BRIEF MACROS DO NOT FIND ANY FUNCTIONS OR DATA TYPES

There is no access to CFTN, CSTN, DFTN (...), due to incorrect path specification, no database is present, the path to the database files is incorrect, the database name is incorrect.

THE BYTE OFFSET CALCULATION FILE "CST_OFFS.C" CANNOT BE COMPILED
Several reasons: Necessary data types or include files are not specified or the CST processing was done with include files other than those being used for compiling. If the number of data type information is too large, some compilers cannot compile the large number of statements in a single file generated from CST ('out of heap space', 'code segment too large' or other messages like that). In that case you may have to split the file into several smaller files or reduce the number of data types to display.

LOCATING ITEMS IN THE BRIEF EDITOR POINTS TO WRONG PLACES

Searching items from within the BRIEF editor points to wrong lines, the requested item is not present there or the file seems to be corrupted. This can have several reasons: The file is not up-to-date and has been changed since the database generation so that the line references are no longer valid. Another reason can be that the source file has explicit #line numbers as it is usual for files produced by source code generators like YACC/BISON or LEX/FLEX. A third reason may be that the source file was generated on an UNIX system and has therefore only LF instead of CR+LF as end-of-line delimiter so that BRIEF cannot display the file correctly, the file seems to be written in a single line. If possible convert such files to DOS text format with UNIX2DOS or other utilities.

THE SOURCE ANALYSIS WITH THE SXT WINDOWS VERSIONS IS VERY SLOW

Windows adds a lot of overhead to the program execution and there is no possibility to avoid this, but there are three things to speed up the SXTWIN programs. First, switch off the message and progress information output ('Info' menu) to avoid time consuming screen updates. Second, if possible, specify a RAM-disk drive for the temporary files (option -v). Third, run the program as an icon, because this avoids any screen updates.

8.7 REFERENCES

8.7.1 BOOKS AND ARTICLES

Brian W. Kernighan, Dennis M. Ritchie: "The C Programming Language", Prentice Hall, Englewood Cliffs, Second Edition 1988

Samuel P. Harbison, Guy L. Steele Jr.: "C: A Reference Manual", Prentice Hall, Englewood Cliffs, Third Edition 1991

Bjarne Stroustrup: "The C++ Programming Language", Addison-Wesley, Second Edition 1992

Bjarne Stroustrup: "The Design and Evolution of C++", Addison-Wesley, 1994

Margaret A. Ellis, Bjarne Stroustrup: "The Annotated C++ Reference Manual" (ARM), Addison-Wesley, Second Edition 1991

"Rationale for American National Standard for Information Systems - Programming Language C" (can be obtained via anonymous FTP from ftp.uu.net in '/doc/standards/ansi/X3.159-1989/rationale.PS.Z')

"Working Paper for Draft Proposed International Standard for Information Systems - Programming Language C++", AT&T, ANSI committee X3J16, ISO working group WG21, January 28, 1993

"Working Paper for Draft Proposed International Standard for Information Systems - Programming Language C++", AT&T, ANSI committee X3J16, ISO working group WG21, April 28, 1995

"Working Paper for Draft Proposed International Standard for Information Systems - Programming Language C++", AT&T, ANSI committee X3J16, ISO working group WG21, December 1996

Bjarne Stroustrup, Keith Gorlen, Phil Brown, Dennis Mancl, Andrew Koenig: "UNIX System V - AT&T C++ Language System, Release 2.1 - Selected Readings", AT&T, 1989

Goldberg, A.: "Programmer as Reader", IEEE Software, September 1987

L.W. Cannon, R.A. Elliot, L.W. Kirchhoff, J.H. Miller, J.M. Milner, R.W. Mitze, E.P. Schan, N.O. Whittington, H. Spencer, D. Keppel, M. Brader: "Recommended C Style and Coding Standards", Technical Report, in the Public Domain, Revision 6.0, July 1991 (revised and updated version of the 'AT&T Indian Hill style guide', can be obtained via anonymous FTP from cs.washington.edu in '~ftp/pub/cstyle.tar.Z')

A. Dolenc, A. Lemmke, D. Keppel, G.V. Reilly: "Notes on Writing Portable Programs in C", Technical Report, in the Public Domain, Revision 8, November 1990 (can be obtained via anonymous FTP from cs.washington.edu in '~ftp/pub/cport.tar.Z')

M. Henricson, E. Nyquist: "Programming in C++, Rules and Recommendations", Technical Report, in the Public Domain, Ellemtel Telecommunication Systems Laboratories, Alvsjö/Sweden, Document No. M 90 0118 Uen, Rev. C (can be obtained via anonymous FTP from various sites as 'rules.ps.Z' or 'c++rules.ps.Z')

H. Wehnes: "FORTRAN 77", 3. Auflage, Carl Hanser Verlag, 1984

H. Sigl: "dBase/Foxbase/Clipper - Globalreferenz", Addison Wesley, 1989

Larisch, D.: "Das grosse Buch zu Clipper 5.0", Data Becker, 1991

Borchers, W.; Dilger, M.; Vonhoegen, H.: "Das grosse Buch zu dBase 5 für Windows", Data Becker, 1994

Gosling, J.; Joy, B.; Steele, G.: "The Java Language Specification", Version 1.0, August 1996 (first printing)

Lindholm, T.; Yellin, F.: "The Java Virtual Machine Specification", September 1996 (first printing)

"A Beginner's Guide to HTML", available via World Wide Web from '<http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html>'

Bansiya, J.; Davis, C.: "Automated Metrics and Object-Oriented Development", Dr. Dobb's Journal December 1997

8.7.2 COMPILER REFERENCE MANUALS

Compiler (on-line) reference manuals and related documentation (language references, language implementations and extensions):

- Microsoft C 5.1
- Microsoft C 6.0
- Microsoft C/C++ 7.0
- Microsoft Visual C++ 1.5 Professional
- Microsoft C/C++ for Windows NT (Beta Release 3/93)
- Microsoft Visual C++ 1.0 for Windows NT (Beta Release 6/93)
- Microsoft Visual C++ 1.1 for Windows NT
- Microsoft Visual C++ 2.2
- Microsoft Visual C++ 4.0
- Microsoft Visual C++ 4.1
- Microsoft Visual C++ 4.2
- Microsoft Visual C++ 5.0

- Microsoft C for SCO UNIX System V Rel. 3.2
- Microsoft Macro Assembler MASM 5.1
- Borland Turbo C++ 1.0
- Borland C++ 2.0
- Borland C++ 3.1
- Borland C++ 1.0 for OS/2
- Borland Turbo Assembler TASM 2.0
- Intel 80860 Metaware High C i860 APX (UNIX-hosted)
- Intel 80960 C-Compiler (ic960, ec960)
- Intel 80960 Assembler (asm960)
- Watcom C/C++ 10.0a
- GNU-960 Tools (UNIX-hosted)
- GNU-C Compiler 2.2.2 / 2.4.5 / 2.5.7 / 2.6.3 / 2.7.0 / 2.7.2.1 (C, C++, Objective-C), GNU-C DOS and OS/2 ports (DJGPP, EMX), LINUX version
- GNU Assembler
- AT&T C++ 2.1 CFRONT (C++ to C translator) for SCO UNIX System V Rel. 3.2
- IBM C-Compilers (CC, XLC) for IBM RS 6000 RISC stations, AIX 3.15
- HP C-Compilers (CC, C89) for HP Apollo 9000 RISC stations, HP-UX 9.0
- SUN SunOS 4.1 C-Compiler
- Digital Equipment Corporation (DEC) VAX C
- Digital Equipment Corporation (DEC) VAX FORTRAN
- Ashton Tate dBase III plus
- SUN Microsystems Java Development Kit JDK 1.1.5
- SUN Microsystems Java Development Kit JDK 1.1.7
- Microsoft Visual J++ 1.1

8.8 TRADEMARKS

All brand or product names are trademarks (TM) or registered trademarks (R) of their respective owners.

The following products and names are Copyright (C) Juergen Mueller (J.M.), all rights reserved world-wide:

CXT (TM) C EXPLORATION TOOLS
CFT (TM) C FUNCTION TREE GENERATOR
CFTN (TM) C FUNCTION TREE NAVIGATOR
CST (TM) C STRUCTURE TREE GENERATOR
CSTN (TM) C STRUCTURE TREE NAVIGATOR

CXTWIN (TM) C EXPLORATION TOOLS for Windows
CFTWIN (TM) C FUNCTION TREE GENERATOR for Windows
CSTWIN (TM) C STRUCTURE TREE GENERATOR for Windows

DXT (TM) DBASE EXPLORATION TOOLS
DFT (TM) DBASE FUNCTION TREE GENERATOR
DFTN (TM) DBASE FUNCTION TREE NAVIGATOR

DXTWIN (TM) DBASE EXPLORATION TOOLS for Windows
DFTWIN (TM) DBASE FUNCTION TREE GENERATOR for Windows

FXT (TM) FORTRAN EXPLORATION TOOLS
FFT (TM) FORTRAN FUNCTION TREE GENERATOR
FFTN (TM) FORTRAN FUNCTION TREE NAVIGATOR

FXTWIN (TM) FORTRAN EXPLORATION TOOLS for Windows
FFTWIN (TM) FORTRAN FUNCTION TREE GENERATOR for Windows

JXT (TM) JAVA EXPLORATION TOOLS
JCT (TM) JAVA CLASS TREE GENERATOR
JCTN (TM) JAVA CLASS TREE NAVIGATOR

JXTWIN (TM) JAVA EXPLORATION TOOLS for Windows
JCTWIN (TM) JAVA CLASS TREE GENERATOR for Windows

LXT (TM) LISP EXPLORATION TOOLS
LFT (TM) LISP FUNCTION TREE GENERATOR
LFTN (TM) LISP FUNCTION TREE NAVIGATOR

LXTWIN (TM) LISP EXPLORATION TOOLS for Windows
LFTWIN (TM) LISP FUNCTION TREE GENERATOR for Windows

These packages are part of

SXT (TM) SOFTWARE EXPLORATION TOOLS

SXTWIN (TM) SOFTWARE EXPLORATION TOOLS for Windows

which provide a similar set of functionalities for the source code analysis of different programming languages.

See PRODUCT.DOC for a complete overview of the SXT packages and the different supported platforms.

9 APPENDIX

9.1 APPENDIX 1: C/C++ PRECOMPILER DEFINES

The following list shows the built-in precompiler defines for the supported compiler types (option -T). It contains the default defines and the optional memory model and architecture defines.

Other default compiler defines which are usually declared by some of the compilers are not automatically defined by the -T option. These are defines for compilation like WINDOWS, __WINDOWS__, _Windows, DLL or __DLL__, for optimization like __OPTIMIZE__ or __FASTCALL__ or others like those about target- (operating-) systems like NT, MIPS, UNIX, unix, __unix__, i386, __i386__, GNUDOS, BSD, VMS, USG, DGUX or hpux. Other sometimes predefined macros are __STRICT_ANSI__ or __CHAR_UNSIGNED__. If necessary, they can be user defined on the command line with the -D option.

The macro name __cplusplus will be defined if the command line option '-C++' is set to enable C++ processing.

MSC51 (Microsoft C 5.1):

Default defines: MSDOS, M_I86
C++ specific defines: (none)
Memory model defines: M_I86SM, M_I86MM, M_I86CM, M_I86LM, M_I86HM

MSC70 (Microsoft C/C++ 7.0):

Default defines: MSDOS, M_I86, _MSC_VER (= 700)
C++ specific defines: (none)
Memory model defines: M_I86TM, M_I86SM, M_I86MM, M_I86CM, M_I86LM, M_I86HM

MSVC15 (Microsoft Visual C++ 1.5):

Default defines: MSDOS, M_I86, _MSC_VER (= 800)
C++ specific defines: (none)
Memory model defines: M_I86TM, M_I86SM, M_I86MM, M_I86CM, M_I86LM, M_I86HM

MSVC22 (Microsoft Visual C++ 2.2):

Default defines: _MSC_VER (= 900), _M_IX86 (= 400), _WIN32
C++ specific defines: (none)
Memory model defines: (not necessary)

MSVC40 (Microsoft Visual C++ 4.0):

Default defines: _MSC_VER (= 1000), _M_IX86 (= 400), _WIN32
C++ specific defines: (none)
Memory model defines: (not necessary)

MSVC41 (Microsoft Visual C++ 4.1):

Default defines: `_MSC_VER` (= 1010), `_M_IX86` (= 400), `_WIN32`
 C++ specific defines: (none)
 Memory model defines: (not necessary)

MSVC42 (Microsoft Visual C++ 4.2):

Default defines: `_MSC_VER` (= 1020), `_M_IX86` (= 400), `_WIN32`
 C++ specific defines: (none)
 Memory model defines: (not necessary)

MSVC50 (Microsoft Visual C++ 5.0):

Default defines: `_MSC_VER` (= 1100), `_M_IX86` (= 500), `_WIN32`
 C++ specific defines: (none)
 Memory model defines: (not necessary)

TC10 (Borland Turbo C++ 1.0):

Default defines: `__MSDOS__`, `__TURBOC__`
 C++ specific defines: `__TCPLUSPLUS`
 Memory model defines: `__TINY__`, `__SMALL__`, `__MEDIUM__`,
`__COMPACT__`, `__LARGE__`, `__HUGE__`

BC20 (Borland C++ 2.0):

Default defines: `__MSDOS__`, `__BORLANDC__` (= 0x0200),
`__TURBOC__` (= 0x0297)
 C++ specific defines: `__BCPLUSPLUS__` (= 0x0200), `__TCPLUSPLUS__`
 (= 0x0200)
 Memory model defines: `__TINY__`, `__SMALL__`, `__MEDIUM__`,
`__COMPACT__`, `__LARGE__`, `__HUGE__`

BC31 (Borland C++ 3.1):

Default defines: `__MSDOS__`, `__BORLANDC__` (= 0x0410),
`__TURBOC__` (= 0x0410)
 C++ specific defines: `__BCPLUSPLUS__` (= 0x0310), `__TCPLUSPLUS__`
 (= 0x0310)
 Memory model defines: `__TINY__`, `__SMALL__`, `__MEDIUM__`,
`__COMPACT__`, `__LARGE__`, `__HUGE__`

BC100S2 (Borland C++ 1.0 for OS/2):

Default defines: `__OS2__`, `__BORLANDC__` (= 0x0400),
`__TURBOC__` (= 0x0400)
 C++ specific defines: `__BCPLUSPLUS__` (= 0x0320), `__TCPLUSPLUS__`
 (= 0x0320), `__TEMPLATES__`
 Memory model defines: (not necessary)

WATCOMC100 (Watcom C/C++ 10.0):

Default defines: `__WATCOMC__` (= 1000)
 C++ specific defines: `__WATCOM_CPLUSPLUS__` (= 1000)

Memory model defines: (cannot be specified, too many targets)

GNUC222 (GNU C 2.2.2):

Default defines: `__GNUC__` (= 2), `__GNUC_MINOR__` (= 2),
`__VERSION__` (= "2.2.2")
C++ specific defines: `__GNUG__`
Memory model defines: (not necessary)

GNUC263 (GNU C 2.6.3):

Default defines: `__GNUC__` (= 2), `__GNUC_MINOR__` (= 6),
`__VERSION__` (= "2.6.3")
C++ specific defines: `__GNUG__`
Memory model defines: (not necessary)

GNUC272 (GNU C 2.7.2):

Default defines: `__GNUC__` (= 2), `__GNUC_MINOR__` (= 7),
`__VERSION__` (= "2.7.2")
C++ specific defines: `__GNUG__`
Memory model defines: (not necessary)

GNUC2721 (GNU C 2.7.2.1):

Default defines: `__GNUC__` (= 2), `__GNUC_MINOR__` (= 7),
`__VERSION__` (= "2.7.2.1")
C++ specific defines: `__GNUG__`
Memory model defines: (not necessary)

GNUC281 (GNU C 2.8.1):

Default defines: `__GNUC__` (= 2), `__GNUC_MINOR__` (= 8),
`__VERSION__` (= "2.8.1")
C++ specific defines: `__GNUG__`
Memory model defines: (not necessary)

I960 (Intel iC960 3.0):

Default defines: `__i960`
C++ specific defines: (none)
Memory model defines: (not necessary)
Architecture defines: `__i960KA`, `__i960KB`, `__i960SA`, `__i960SB`,
`__i960MC`, `__i960CA`

USER DEFINED ADAPTATION OF CFT AND CST TO PREVIOUSLY LISTED AND OTHER COMPILERS NOT SUPPORTED BY OPTION -T

The adaptation of CFT and CST to C/C++ compilers can be done with the -D, -U and -B options. First you have to find out which precompiler defines the compiler uses by default or are related to command line options according to the settings used for your compilation. For these information see the documentation and help-files for

your compiler. The necessary precompiler defines have to be specified for CFT and CST with -D. Additionally it may be necessary to specify undefines with -U and to declare the basic type size with -B. The following line shows as an example the necessary options for the MS Visual C++ 1.5 compiler (large model):

```
-DMSDOS -DM_I86 -D_MSC_VER=800 -DM_I86LM -B0,1,2,2,4,4,8,10*4,4
```

9.2 APPENDIX 2: RESERVED C/C++ KEYWORDS

The following list shows the keywords being recognized by CFT and CST, the standard C keywords, the C++ keywords and the non-standard keywords which are compiler dependent extensions to the C or C++ language. Standard C keywords are also C++ keywords, always! The C++ keywords are recognized only if option '-C++' is set, otherwise they are treated as identifiers. This list may not be complete or correct due to changes or upcoming new releases of the supported compilers with new extensions or extensions to the language standard. Vendor or compiler specific extensions to the languages as in GNU C (e.g. __alignof, __classof, interface, signature, __FUNCTION__, ...) or undocumented features are ignored.

Standard C keywords: asm, auto, break, case, char, const, continue, default, do, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while

Standard C++ keywords: and, and_eq, bitand, bitor, bool, catch, class, const_cast, delete, double, dynamic_cast, explicit, export, false, friend, inline, mutable, namespace, new, not, not_eq, operator, or, or_eq, private, protected, public, reinterpret_cast, static_cast, template, this, throw, true, try, typeid, typename, using, virtual, wchar_t, xor, xor_eq

Following are some vendor specific C and C++ keywords. It is not known whether all of them are still valid in actual versions.

Vendor specific C keywords (Microsoft, Borland): cdecl, far, fortran, huge, interrupt, near, pascal, __asm, __based, __cdecl, __const, __emit, __export, __far, __fastcall, __finally, __fortran, __huge, __interrupt, __loadds, __near, __saveregs, __segment, __segname, __self, __signed, __stdcall, __syscall, __try, __volatile, __asm, __based, __cdecl, __emit, __export, __far, __fastcall, __fortran, __huge, __interrupt, __loadds, __near, __pascal, __saveregs, __seg, __segment, __segname, __self

Vendor specific C++ keywords (Microsoft, Borland, GNU): classof, dynamic, except, exception, overload, typeof, __alignof, __alignof__, __asm__, __attribute, __attribute__, __classof, __classof__, __const__, __except, __extension__, __headof, __headof__, __inline, __inline__, __label__, __signed__, __typeof, __typeof__, __volatile__

9.3 APPENDIX 3: RESERVED FORTRAN KEYWORDS

The following lists show the keywords and intrinsic functions from FORTRAN 77 and non-standard extensions (VAX, CDC, AIX, UNIX, ...) recognized by FFT. This list may not be complete or correct.

Keywords (standard & non-standard): ACCEPT, ASSERT, ASSIGN, AUTOMATIC, BACKSPACE, BLOCKDATA, BLOCK, BYTE, CALL, CASE, CHARACTER, CLOSE, COMMON, COMPLEX, CONTINUE, DATA, DELETE, DICTIONARY, DIMENSION, DOUBLECOMPLEX, DOUBLEPRECISION, DOUBLE, DOWHILE, DO, ELSEIF, ELSE, ENDDO, ENDFILE, ENDIF, ENDMAP, ENDSELECT, ENDSTRUCTURE, ENDUNION, ENDWHILE, END, ENTRY, EQUIVALENCE, EXTERNAL, FILE, FORMAT, FUNCTION, GOTO, GO, IF, IMPLICITNONE, IMPLICIT, INCLUDE, INQUIRE, INTEGER, INTRINSIC, LOGICAL, NAMELIST, NONE, OPEN, PARAMETER, PAUSE, POINTER, PRECISION, PRINT, PROGRAM, PUNCH, READ, REAL, RECORD, RETURN, REWIND, REWRITE, SAVE, SELECT, SELECTCASE, STATIC, STOP, STRUCTURE, SUBROUTINE, THEN, TO, TYPE, UNION, UNLOCK, VOLATILE, WHILE, WRITE, \$INCLUDE

Intrinsic functions: ABS, ACHAR, ACOS, AIMAG, AINT, ALOG10, ALOG, AMAX0, AMAX1, AMIN0, AMIN1, AMOD, AND, ANINT, ASIN, ATAN2, ATAN, CABS, CCOS, CEXP, CHAR, CLOG, CMPLX, CONJG, COSH, COS, CSIN, CSQRT, DABS, DACOS, DASIN, DATAN2, DATAN, DBLE, DCMLPX, DCONJG, DCOSH, DCOS, DDIM, DEXP, DFLOAT, DIMAG, DIM, DINT, DLOG10, DLOG, DMAX1, DMIN1, DMOD, DNINT, DPROD, DSIGN, DSINH, DSIN, DSQRT, DTANH, DTAN, EPBASE, EPEMAX, EPEMIN, EPHUGE, EPMRSP, EPPREC, EPTINY, EXP, FLOAT, FPABSP, FPEXP, FPFRAC, FPMAKE, FPRRSP, FPSCAL, IABS, IACHAR, ICHAR, IDIM, IDINT, IDNINT, IFIX, IMAG, INDEX, INT, ISIGN, LEN, LGE, LGT, LLE, LLT, LOG10, LOG, LSHIFT, MAX0, MAX1, MAX, MIN0, MIN1, MIN, MOD, NINT, NOT, OR, REAL, RSHIFT, SIGN, SINH, SIN, SNGL, SQRT, TANH, TAN, XOR, ZABS, ZCOS, ZEXP, ZLOG, ZSIN, ZSQRT,

Non-standard intrinsic functions: ABORT, ACOSD, ASIND, ATAN2D, ATAND, BTEST, CABS1, CAMAX, CAMIN, CASUM, CAXPY, CCOPY, CDABS, CDCOS, CDEXP, CDLOG, CDOTC, CDOTU, CDSIN, CDSQRT, CMAX, CMIN, CNORM2, CNRM2, CNRSQ, COSD, CROTG, CROT, CSCAL, CSET, CSIGN1, CSIGN, CSUM, CSWAP, CVCAL, CZAXPY, DAMAX, DAMIN, DASUM, DATE, DAXPY, DCOPY, DDOT, DMAX, DMIN, DNORM2, DNRM2, DNRSQ, DREAL, DROTG, DROT, DSCAL, DSET, DSUM, DSWAP, DVCAL, DZAXPY, ERRSNS, EXIT, GETARG, GETENV, GMTIME, HFIX, I, IARGC, IAND, IBCHNG, IBCLR, IBITS, IBSET, ICAMAX, ICAMIN, ICMAX, ICMIN, IDAMAX, IDAMIN, IDATE, IDMAX, IDMIN, Ieor, IOR, IQINT, IQNINT, IRAND, ISAMAX, ISAMIN, ISHA, ISHC, ISHFT, ISHFTC, ISMAX, ISMIN, IZAMAX, IZAMIN, IZMAX, IZMIN, JFIX, LTIME, MVBITS, NWORKERS, QEXT, QFLOAT, RAN, RAND, SAMAX, SAMIN, SASUM, SAXPY, SCOPY, SDOT, SECNDS, SIND, SIZEOF, SMAX, SMIN, SNORM2, SNRM2, SNRSQ, SRAND, SROTG, SROT, SSCAL, SSET, SSUM, SSWAP, SVCAL, SYSTEM, SZAXPY, TAND, TIME, ZAMAX, ZAMIN, ZASUM, ZAXPY, ZCOPY,

ZDOTC, ZDOTU, ZEXT, ZMAX, ZMIN, ZNORM2, ZNRM2, ZNRSQ, ZROTG, ZROT, ZSCAL, ZSET, ZSUM, ZSWAP, ZVCAL, ZZAXPY

VAX specific built-in functions: %DESCR, %LOC, %REF, %VAL

9.4 APPENDIX 4: EFFICIENCY

To provide some values about the speed and the efficiency of the programs, tests were performed with the command line versions of CFT and CST. All tests were made with a 166 MHz Pentium with 64 MB RAM and a 9 ms SCSI hard disk under Windows 95.

The first test with CFT 2.57 was done with the source code of the GNU-C compiler version 2.7.2.1 (C and C++ part). The following results have been found:

- 250 files (148 source files and 102 include files) have been scanned
- a total number of 3961 functions has been found
- the critical function call path has a maximum nesting level of 127
- the total size of the 250 files is 12.182 MB with 392297 lines
- the effective size of the preprocessed and scanned source code (source files and their included files) is 53.143 MB with 1466907 lines
- the total time for the complete processing was 4'29" minutes (1'09" for preprocessing, 2'17" for analyzing, 32" for output file writing, 26" for database writing)
- the average speed for this source code was about 760 KB/min. respectively 21000 lines/min (preprocessing) and about 387 KB/min. respectively 10700 lines/min (analysis).

A second test with CST 2.57 was done with the C++ source code of the Microsoft Foundation Class (MFC) 4.2. The following results have been found:

- 406 files (233 source files and 173 include files) have been scanned
- a total number of 2452 data types has been found
- the maximum data type nesting level is 24
- the total size of the 406 files is 8.662 MB with 302100 lines
- the effective size of the preprocessed and scanned source code (source files and their included files) is 2.161 GB (!) with 72639000 lines
- the total time for the complete processing was 64'28" minutes (33'22" for preprocessing, 28'24" for analyzing, 1'33" for output file writing, 59" for database writing)
- the average speed for this source code was about 1.079 MB/min. respectively 36270 lines/min (preprocessing) and about 1.267 MB/min. respectively 42610 lines/min (analysis).

The third and fourth test with were done with a large commercial project written in C and C++. The results for CFT 2.57 are:

- 494 files (268 source files and 226 include files) have been scanned
- a total number of 4110 functions has been found
- the critical function call path has a maximum nesting level of 18
- the total size of the 494 files is 36.214 MB with 911300 lines
- the effective size of the preprocessed and scanned source code (source files and their included files) is 669.242 MB with 12083300 lines
- the total time for the complete processing was 18'28" minutes (10'11" for preprocessing, 6'54" for analyzing, 36" for output file writing, 29" for database writing)
- the average speed for this source code was about 1.090 MB/min. respectively 19700 lines/min (preprocessing) and about 1.612 MB/min. respectively 29120 lines/min (analysis).

The results for CST 2.57 are:

- 494 files (268 source files and 226 include files) have been scanned
- a total number of 2966 data types has been found
- the maximum data type nesting level is 15
- the total size of the 494 files is 36.214 MB with 911300 lines
- the effective size of the preprocessed and scanned source code (source files and their included files) is 669.242 MB with 12083300 lines
- the total time for the complete processing was 18'23" minutes (10'14" for preprocessing, 5'44" for analyzing, 1'38" for output file writing, 36" for database writing)
- the average speed for this source code was about 1.090 MB/min. respectively 19700 lines/min (preprocessing) and about 1.942 MB/min. respectively 35000 lines/min (analysis).

The calculated average values for the analysis speed differ due to the effective size of the 'really' present source code in relation to the size of the comments which can be seen by the code/file size ratio. The speed values do not consider that, if the preprocessing option -P is set, the source code is first preprocessed to a temporary file and then analyzed in a second step so that large parts of the source code are read twice (original and preprocessed code) and written once (intermediate preprocessor output).

With these facts in mind, the analysis speed of CFT and CST seems to be quite acceptable!

9.5 APPENDIX 5: SYSTEM REQUIREMENTS

DOS 16 bit real mode versions:

- IBM-AT or 100% compatible with Intel 80286 or higher, 640 KB RAM, hard-disk, MS-DOS 5.0 or higher

DOS 32 bit protected mode versions:

- IBM-AT or 100% compatible with Intel 80486 or higher, 8 MB, hard-disk, MS-DOS 5.0 or higher

Windows 16 bit versions:

- IBM-AT or 100% compatible with Intel 80386 or higher, 4 MB RAM (8 MB recommended), hard-disk, MS-DOS 5.0 or higher, Windows 3.1 or Windows for Workgroups 3.11 (enhanced mode), VB40016.DLL (Visual Basic 4.0 16-bit Run-Time library installed in the \windows\system directory, required by the DLL sample application SXTNVIEW)

Windows 32 bit versions:

- IBM-AT or 100% compatible with Intel 80486 or higher, 16 MB RAM, hard-disk, Windows 95 or Windows NT 4.0

OS/2 32 bit versions:

- IBM-AT or 100% compatible with Intel 80486 or higher, 16 MB RAM, hard-disk, OS/2 3.0

LINUX 32 bit versions:

- IBM-AT or 100% compatible with Intel 80486 or higher, 16 MB RAM, hard-disk, LINUX Kernel 2.0.32

9.6 APPENDIX 6: INSTALLATION

See the appropriate documentation (INSTALL.DOC, INSTALL.W32, PROBLEMS.DOC, ...) for information about the installation of the SXT programs.

9.7 APPENDIX 7: SXT REVIEWS

- The C Users Journal (Volume 12, Number 1, January 1994) (CXT)
- The C/C++ Users Journal (Volume 12, Number 12, December 1995) (CXT, CXTWIN)
- Professionelle Shareware 1/1993 (Computer Solutions Software GmbH) (CXT)
- Professionelle Shareware 3/1994 (Computer Solutions Software GmbH) (DXT, FXT, LXT)

- Shareware Professionell 11/1994 (Computer Solutions Software GmbH)
(CXTWIN, DXTWIN, FXTWIN, LXTWIN)

9.8 APPENDIX 8: AVAILABILITY

The SHAREWARE versions of the SXT programs can be obtained from various shareware vendors, accessed and downloaded via WWW and FTP from many Internet sites and mailboxes and found on many CD-ROM shareware collections.

The following list is not complete, only the known vendors, WWW- and FTP-sites (usually primary sites where the software is uploaded by the author or which are known to be up-to-date) or CD-ROM's are shown.

VENDORS:

- Computer Solutions Software (CSL) GmbH, Postfach 1180, D-85561 Grafing, GERMANY
- PEARL Agency, Am Kalischacht 4, D-79426 Buggingen, GERMANY
- The C-Users' Group (CUG), 1601 W. 23rd St., Suite 200, Lawrence, KS 66046, U.S.A. (CXT: CUG Library Disk # 391)
- EMS Professional Software, 4505 Buckhurst Ct., Olney, MD 20832-1830, U.S.A.
- Ziff-Davies Interactive, The Riverview Building, One Athenaeum Street, Cambridge, MA 02142, U.S.A. (distribution on CD-ROM or electronically through Public Brand Software, ZiffNet on CompuServe, ZiffNet on Prodigy, and Interchange)
- Limelight Media Inc., P.O. Box 3536, Terre Haute, IN 47803, U.S.A. (Platinum Shareware CD-ROM Series)
- AMUG CD, Inc., 4131 N. 24th Street #A-120, Phoenix, AZ 85016, U.S.A.

WWW / FTP:

The primary upload site for all SXT programs is SIMTEL.NET:

WWW: <http://www.simtel.net/simtel.net>

FTP: [ftp.simtel.net](ftp://ftp.simtel.net)

CD-ROM:

- The C-Users' Group (CUG) Library on CD-ROM
(The C-Users' Group (CUG), 1601 W. 23rd St., Suite 200, Lawrence, KS 66046, U.S.A.)
- CSL-MEGA CD Vol. 6, Vol. 7, Vol. 9
(Computer Solutions Software (CSL) GmbH, Postfach 1180, D-85561 Grafing, GERMANY)

There are also CD-ROM's available containing the complete collection of files from the SIMTEL, GARBO and other FTP-sites which may also include some or all of the SXT programs.

BOOKS:

Several books are available or have been announced which include SXT programs on CD-ROM's or disks:

- 'Windows Programming with Shareware Tools' by Victor Volkman
(Miller Freeman Inc., ISBN # 0-87930-434-0)
(more information at <http://www.HAL9k.com>)
- 'Algorithmen fuer C und C++' (Addison-Wesley)
- 'Windows 95 Secrets' (IDG Books)
- 'NetWarriors in C++: Programming 3D Windows Games' (John Wiley & Sons, Inc.).