

User's Guide



Version 5.6

a product of

S.A.F.E.

**Software Analysis & Forensic Engineering
Corporation**

Table of Contents

CodeSuite	1
BitMatch.....	1
CodeCLOC.....	1
CodeCross.....	2
CodeDiff.....	2
CodeMatch	3
CodeSplit	3
FileCount	3
FileIdentify	4
FileIsolate	4
HTML Preprocessor.....	4
SourceDetective	4
Copyrights, Trademarks, Patents	5
Using CodeSuite	6
System Requirements	6
Licenses	7
The Menu and Toolbar	9
BitMatch	16
Running BitMatch	16
BitMatch Algorithms.....	19
CodeCLOC.....	24
Running CodeCLOC.....	24
CodeCLOC Algorithm	26
CodeCross	31
Running CodeCross	31
CodeCross Algorithm.....	33
CodeDiff	38
Running CodeDiff	38
CodeDiff Algorithm.....	41
CodeMatch	45
Running CodeMatch	45
CodeMatch Algorithms	48

CodeSplit.....	56
Running CodeSplit.....	56
FileCount.....	62
Running FileCount.....	62
FileIdentify.....	63
Running FileIdentify.....	63
FileIsolate.....	65
Running FileIsolate.....	65
HTML Preprocessor.....	68
Running HTML Preprocessor.....	68
Statistics.....	70
Calculating Statistics.....	70
SourceDetective.....	71
Running SourceDetective.....	71
Exporting Databases.....	73
Filters.....	73
Sorting Databases.....	80
HTML Reports.....	81
CLOC Spreadsheets.....	82
Distribution Spreadsheets.....	86
Creating PID Spreadsheets.....	89
Search Spreadsheets.....	92
Summary Spreadsheet.....	96
Creating XML Databases.....	98
Languages.....	99
Languages Supported.....	99
Advanced Topics.....	100
CodeSuite Database Format.....	100
CodeSuite Filter Format.....	105
Command Line Interface.....	107
XML File Format.....	115
Contacting SAFE Corporation.....	131
Contacting SAFE Corporation.....	131
Index.....	132

CodeSuite

CodeSuite® is a collection of computer code analysis tools. The individual tools that comprise the suite of tools include BitMatch®, CodeCLOC®, CodeCross®, CodeDiff®, CodeMatch®, CodeSplit®, HTML Preprocessor™, FileCount™, FileIdentify™, FileIsolate™, and SourceDetective®, all of which are described below.



BitMatch uses fast, simple algorithms to compare thousands of executable binary files in multiple directories and subdirectories to thousands of other executable binary files or source code files in order to determine which files are the most highly correlated. BitMatch is particularly useful for finding programs that have been copied, but where you only have access to the program executable binary files and not the source code.

BitMatch compares every file in one directory with every file in another directory, including all subdirectories if requested. BitMatch produces a database that can then be exported to an HTML basic report that lists the most highly correlated pairs of files. You can click on any particular pair listed in the HTML basic report see an HTML detailed report that shows the specific items in the files (strings or identifiers) that caused the high correlation.

BitMatch examines all text strings, comments, and identifier names that it can find in the executable files in order to determine copying. If a specific user message or a unique subroutine name is found in two files, there is a possibility that one was copied from the other. Note that BitMatch gives only a rough determination whether copying took place. False positives and false negatives are both possible. CodeMatch is needed to compare source code to help make a definitive determination.



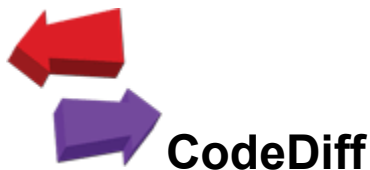
CodeCLOC uses a fast, simple algorithm to calculate development progress across two different versions of software, measuring code changes, the amount of original code in your new code, and the amount of original code. CodeCLOC allows measurement of software changes and software intellectual property changes for calculating such things as transfer pricing.

CodeCLOC compares every file in one directory with every file in another directory, including all subdirectories if requested. CodeCLOC produces a database that can then be exported to an HTML basic report that lists all files in one folder that have statements that match comments in files in the other folder. You can click on any particular file pair listed in the HTML basic report see an HTML detailed report that shows the specific lines in the files that match. CodeCLOC also produces spreadsheets showing detailed statistics about code growth from version to version.



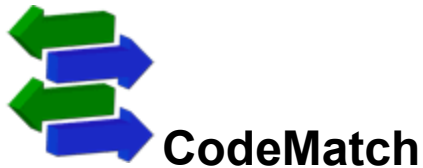
CodeCross uses a fast, simple algorithm to compare thousands of source code files in multiple directories and subdirectories to find programming statements in one file that have been commented out of another file -- a possible sign of copying.

CodeCross compares every file in one directory with every file in another directory, including all subdirectories if requested. CodeCross produces a database that can then be exported to an HTML basic report that lists all files in one folder that have statements that match comments in files in the other folder. You can click on any particular file pair listed in the HTML basic report see an HTML detailed report that shows the specific lines in the files that match.



CodeDiff uses a fast, simple algorithm to compare thousands of source code files in multiple directories and subdirectories to find files that are exact matches or nearly exact matches. CodeDiff looks for identical lines in pairs of source code files. While not as sophisticated or as accurate as CodeMatch, CodeDiff runs much faster. CodeDiff is particularly useful for comparing files where it is already known that many of the files are nearly identical. CodeDiff can be run as a precursor to running CodeMatch when attempting to find source code plagiarism.

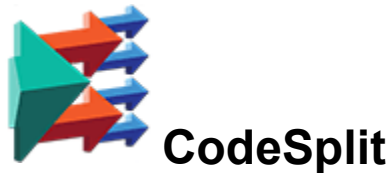
CodeDiff compares every file in one directory with every file in another directory, including all subdirectories if requested. CodeDiff produces a database that can then be exported to an HTML basic report that lists the most similar pairs of files based on matching lines of source code in the files. You can click on any particular pair listed in the HTML basic report see an HTML detailed report that shows the specific lines in the files that are different.



CodeMatch compares thousands of source code files in multiple directories and subdirectories to determine which files are the most highly correlated. This can be used to significantly speed up the work of finding source code plagiarism, because it can direct the examiner to look closely at a small amount of code in a handful of files rather than thousands of combinations. CodeMatch is also useful for finding open source code within proprietary code, determining common authorship of two different programs, and discovering common, standard algorithms within different programs.

CodeMatch compares every file in one directory with every file in another directory, including all subdirectories if requested. CodeMatch produces a database that can then be exported to an HTML basic report that lists the most highly correlated pairs of files. You can click on any particular pair listed in the HTML basic report see an HTML detailed report that shows the specific items in the files (statements, comments, strings, identifiers, or instruction sequences) that caused the high correlation.

CodeMatch uses unique algorithms to find various different ways that source code files are correlated. These algorithms can find directly copied source code and even source code that has been modified to avoid detection.



CodeSplit takes all source code files in multiple directories and subdirectories and splits them into basic elements (statements, comments, strings, and identifiers). The resulting database can be used in conjunction with SourceDetective to find Internet evidence that the source code was derived from third-party code found on the Internet.

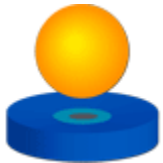


FileCount is a utility that counts the number of files, non-blank lines, and bytes in a large set of files in a directory tree. FileCount is useful when using CodeDiff to generate statistics about a set of source code files.



FileIdentify

FileIdentify is a utility that examines all of the file types in a given directory, or an entire directory tree, and reports the associated programming languages if known.



FileIsolate

FileIsolate is a utility that allows files to be selectively deleted from a large group of files in an entire directory or directory tree. FileIsolate is useful when examining a large number of files but only certain files are of interest and all other files can be deleted to make searches faster.



HTML Preprocessor

HTML Preprocessor is a utility that transforms HTML pages into pure HTML files, pure text files, and pure script files that are amenable to analysis by CodeSuite, DocMatch, and other source code analysis tools.



SourceDetective

SourceDetective is a utility that searches the Internet for all references to matching statements, comments, and identifiers found in a CodeSuite database. SourceDetective is used to determine whether statements, comments, and identifiers found in two sets of files are commonly used or not, depending on how many references can be found on the Internet.

Copyrights, Trademarks, Patents

Copyrights

The materials in this user's guide are copyright 2005-2025 by Software Analysis and Forensic Engineering Corporation.

All written materials from SAFE Corporation regarding CodeSuite, including the material in this User's Guide and the source code for all versions of CodeSuite are the copyright of SAFE Corporation.

Trademarks

SAFE Corporation, the SAFE Corporation logo, the SAFE Corporation brand, CodeSuite, the CodeSuite logo, BitMatch, CodeCLOC, CodeCross, CodeDiff, CodeMatch, CodeSplit, FileCount, FileIdentify, FileIsolate, HTML Preprocessor, SourceDetective, and all other SAFE Corporation product names referenced herein are registered trademarks or trademarks of SAFE Corporation. All other brand and product names mentioned herein are trademarks of their respective owners.

Patents

CodeSuite is covered by U.S. patents 7,503,035, 7,823,127, 8,255,885, 8,261,237, 8,495,586, 9,003,366, 9,043,375, and 9,053,296.

Using CodeSuite

System Requirements

CodeSuite will run on any computer using any of the following versions of the Microsoft Windows operating system:

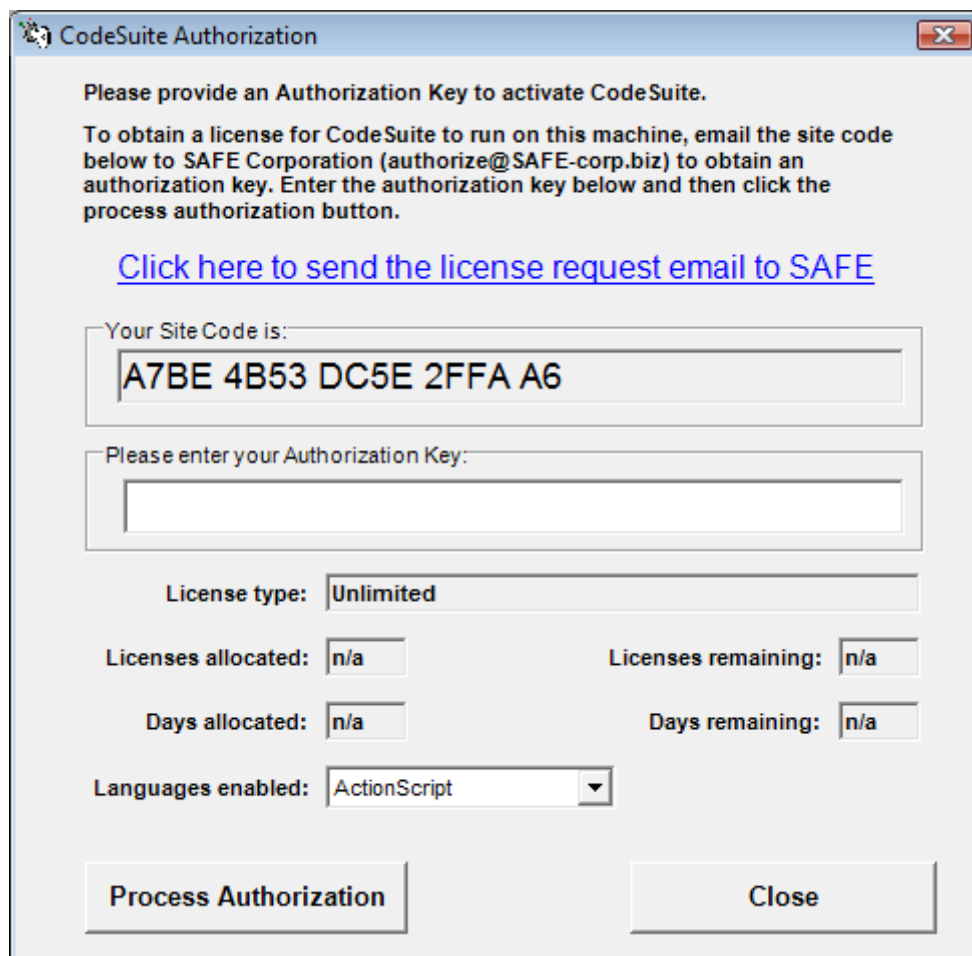
- Windows Vista
- Windows 7
- Windows 8
- Windows 10
- Windows 11

Note that CodeSuite will not run on a virtual system and may not run on some systems using a remote desktop.

Licenses

Licenses must be purchased from SAFE Corporation. Some functions of CodeSuite including FileCount, FileIdentify, FileIsolate, HTML Preprocessor, generating HTML reports and spreadsheets, and filtering existing databases do not require a license.

To request licenses, open the authorization form shown below from the Help menu. Send the site code to SAFE Corporation and the number of licenses requested, along with appropriate payment. SAFE Corporation will send back an Authorization Key that must be entered into the field in the form. Press the process authorization button and the form will show the following information. Licenses are enabled for only one PC and cannot be transferred to another PC.



The image shows a Windows-style dialog box titled "CodeSuite Authorization". It contains the following elements:

- Text:** "Please provide an Authorization Key to activate CodeSuite. To obtain a license for CodeSuite to run on this machine, email the site code below to SAFE Corporation (authorize@SAFE-corp.biz) to obtain an authorization key. Enter the authorization key below and then click the process authorization button."
- Link:** A blue underlined link: "Click here to send the license request email to SAFE".
- Text Field:** "Your Site Code is:" followed by a text box containing "A7BE 4B53 DC5E 2FFA A6".
- Text Field:** "Please enter your Authorization Key:" followed by an empty text box.
- License type:** A dropdown menu showing "Unlimited".
- Licenses allocated:** A text box showing "n/a".
- Licenses remaining:** A text box showing "n/a".
- Days allocated:** A text box showing "n/a".
- Days remaining:** A text box showing "n/a".
- Languages enabled:** A dropdown menu showing "ActionScript".
- Buttons:** "Process Authorization" and "Close".

License Type

The license can be one of three types.

- **File size based.** Used to examine a fixed amount of bytes of source code. Licenses are used up as source code is examined. SourceDetective searches of the Internet also use up licenses.
- **Time based.** Used to examine any amount of code for a fixed number of days. Note that there is still a limit to the number of SourceDetective searches of the Internet that can be performed. If that limit is reached, no more searching can be done for the remainder of the license term unless a new license is purchased.
- **Unlimited.** There is no limit on the number of megabytes that can be examined and there is no expiration date.

Licenses Allocated and Licenses Remaining

These fields indicate the number of licenses that were originally allocated and how many unused licenses remain. These fields are valid only for a megabyte-based license. For other licenses, the fields are not applicable ("n/a").

Days Allocated and Days Remaining

These fields indicate the number of days that were originally allocated for the license and how many days remain on the license. These fields are valid only for a time-based license. For other licenses, the fields are not applicable ("n/a").

Languages Enabled

This pulldown list shows all of the programming languages that are enabled for analysis by the license.

See the SAFE Corporation website for license costs, as they may change.

The Menu and Toolbar

The CodeSuite menu and toolbar is shown below. Each menu selection is described in more detail below. If the menu selection can also be found on the toolbar, the corresponding toolbar icon is shown.



File Menu

The following selections are found on the File menu.



File->Open database...

This menu selection opens an existing CodeSuite database file.



File->Open filter...

This menu selection opens an existing CodeSuite filter file. For more information see the section entitled Using Filters.



File->Close all

This menu selection closes all open CodeSuite database files and filter files.



File->Export database->Filtered database

This menu selection creates a new database file by applying the open filter to the open database. For more information see the section entitled Using Filters.

**File->Export database->HTML report**

This menu selection converts the open CodeSuite database to a readable HTML basic report file with links to many readable HTML detailed report files. For more information, see the section entitled Creating HTML Reports.

**File->Export database->Sorted database**

This menu selection converts the open CodeSuite database to a new database in which all file pairs are sorted by their scores, from highest to lowest. This also means that the HTML reports generated from these databases will be sorted, making it easier to start examining file pairs starting with those that have the highest overall scores.

**File->Export database->Spreadsheets->CLOC Spreadsheet**

This menu selection takes the open CodeCLOC database and creates a spreadsheet containing a statistical analysis of the distribution of the file scores. For more information, see the section entitled Creating CLOC Spreadsheets.

**File->Export database->Spreadsheets->Distribution Spreadsheet**

This menu selection takes the open CodeSuite database and creates a spreadsheet containing a statistical analysis of the distribution of the file scores. For more information, see the section entitled Creating Distribution Spreadsheets.

**File->Export database->Spreadsheets->PID Spreadsheet**

This menu selection takes the open CodeSuite database and creates a spreadsheet containing a listing off all partially matching identifiers. For more information, see the section entitled Creating PID Spreadsheets.



File->Export database->Spreadsheets->Search Spreadsheet

This menu selection takes the open CodeSuite database and creates spreadsheets containing information on the number of Internet hits for statements, comments, and identifiers. For more information, see the section entitled Creating Search Spreadsheets.



File->Export database->Spreadsheets->Summary Spreadsheet

This menu selection takes the open CodeSuite database and creates a spreadsheet containing a summary statistical analysis of the file scores. For more information, see the section entitled Creating Summary Spreadsheets.



File->Export database->XML

This menu selection takes the open CodeSuite database and converts it to an XML file. For more information, see the section entitled Creating XML databases.



File->Save filter...

Save the open filter file.



File->Save filter as...

Save the open filter file with a new name.



File->Recent databases...

Open one of the most recently opened database.

**File->Recent filters...**

Open one of the most recently opened filters.

File->Exit

Exit the program.

View Menu

The following selections are found on the File menu.

View->Toolbar

This menu selection toggles the toolbar on and off.

View->Status Bar

This menu selection toggles the status bar on and off.

Tools Menu**Tools->BitMatch**

This menu selection brings up the BitMatch form. See the section entitled Running BitMatch for more information.

**Tools->CodeCLOC**

This menu selection brings up the CodeCLOC form. See the section entitled Running CodeCLOC for more information.



Tools->CodeCross

This menu selection brings up the CodeCross form. See the section entitled Running CodeCross for more information.



Tools->CodeDiff

This menu selection brings up the CodeDiff form. See the section entitled Running CodeDiff for more information.



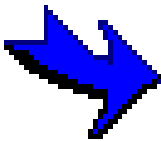
Tools->CodeMatch

This menu selection brings up the CodeMatch form. See the section entitled Running CodeMatch for more information.



Tools->CodeSplit

This menu selection brings up the CodeSplit form. See the section entitled Running CodeSplit for more information.



Tools->Restart...

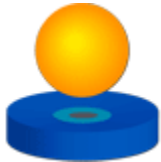
This menu selection lets you select a CodeSuite database that was interrupted before completion. When the database is selected, CodeSuite will restart the comparison where it left off previously and will complete the comparison and generate a full database file.

**Tools->FileCount**

This menu selection brings up the FileCount form. See the section entitled Running FileCount for more information.

**Tools->FileIdentify**

This menu selection brings up the FileIdentify form. See the section entitled Running FileIdentify for more information.

**Tools->FileIsolate**

This menu selection brings up the FileIsolate form. See the section entitled Running FileIsolate for more information.

**Tools->HTML Preprocessor**

This menu selection brings up the HTML Preprocessor form. See the section entitled Running HTML Preprocessor for more information.

**Tools->SourceDetective**

This menu selection brings up the HTML Preprocessor form. See the section entitled Running HTML Preprocessor for more information.

Tools->Statistics

This menu selection brings up the database statistics form. See the section entitled Calculating Statistics for more information.

Tools->Filters

This menu selection brings up the filter form. See the section entitled Using Filters for more information.

Help Menu



Help->Contents

This menu selection brings up this user's guide.



Help->Authorize

This menu selection brings up the authorization form for entering licenses to enable the various tools. See the section entitled Licenses for more information.

Help->About

This menu selection gives the version number and other information about CodeSuite.

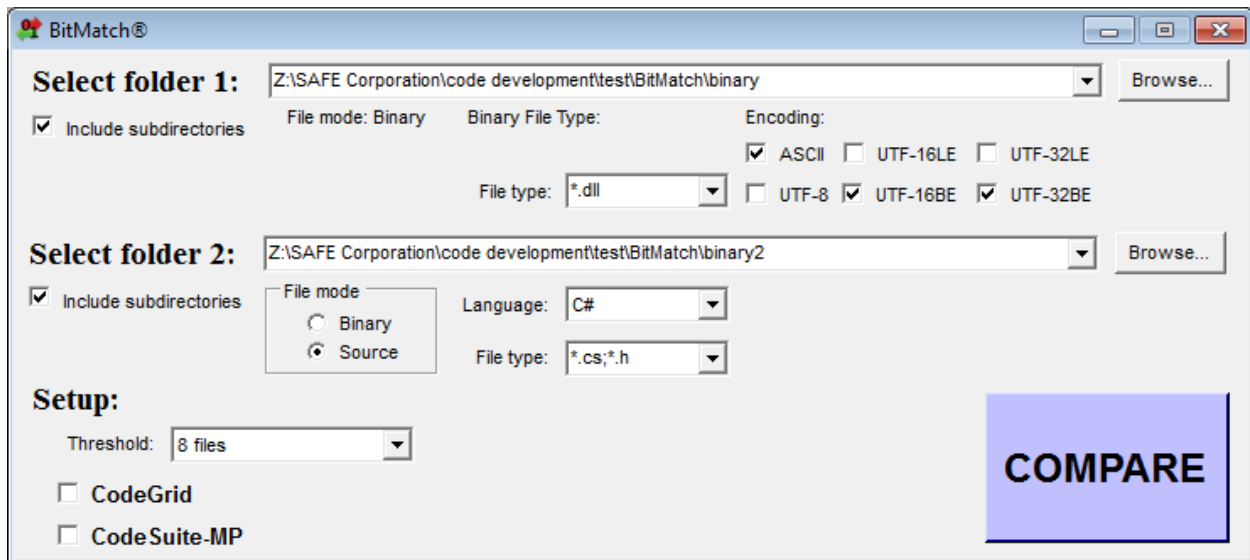
Status Bar

The status bar, located at the bottom of the CodeSuite window, is divided into four sections. From left to right, the first section shows the name of the CodeSuite database file that is open, the second section shows the name of the CodeSuite filter file that is open, the third section shows the current date, and the fourth section shows the current time.

BitMatch

Running BitMatch

BitMatch compares strings and identifier names from executable binary files to other executable binary files or source code files. Following that are step-by-step instructions for running BitMatch.



Step 1

Select the first folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 2

Specify the types of binary files in the first folder to compare. You can type over the suggested file types with your own file types. Separate different file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 3

Select the second folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 4

Select the encoding of the files. If you know the encoding of the binary file, select it. If you do not know the encoding, you can use the following information to extract information from different possible encodings.

ASCII

ASCII encodes each character into a single byte. ASCII is used to extract text sequences based on printable character codes of decimal 32 to 126, plus horizontal tab. This mode was the default mode in prior versions of CodeSuite.

ASCII was used on DOS systems, early Linux systems, and many later systems. UTF-8 is common on Windows and Linux systems. Macintosh systems before OS X used an extension of ASCII called Mac OS Roman.

UTF-8

UTF-8 incorporates all of the ASCII characters, but provides for multi-byte character sequences to specify additional Unicode characters.

UTF-8 is popular for Internet browsing applications, which often require multi-language support. UTF-8 is also common on Windows, Linux, and Unix systems. Mac OS X uses UTF-8.

UTF-16LE and UTF-16BE

Both UTF-16 formats utilize sequences of two consecutive bytes to form a single 16-bit character. In UTF-16LE, "LE" for "little endian," the low order byte occurs before the high order byte. In UTF-16BE, "BE" for "big endian," the high byte precedes the low order byte.

UTF-16LE is common on Windows systems. UTF-16BE is common on Java platforms.

UTF-32LE and UTF-32BE

Both UTF-32 formats utilize four successive bytes to form a 32-bit character. In UTF-32LE, the bytes are sequenced from the low order byte to the high order byte. In UTF-16BE bytes are sequenced from the high order byte to the low order byte.

Step 5

Select whether the second folder contains binary files or source code files. If the second folder contains source code files, select the source code language.

Step 6

Specify the types of files in the second folder to compare. Separate different file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 7

Select the reporting threshold from the pulldown menu. This determines how many files are reported. BitMatch reports only the most similar files. By setting the number of files to report to a large number you may get a very large database. By setting the number of files to report to a small number, the database will be smaller, but it may not include all the similar files that you would like to see.

Step 8

Check the CodeGrid box to run the comparison on a grid of computers if you have a license to do so. Check the CodeSuite-MP box to run the comparison on a multicore computer if you have a license to do so. When the CodeSuite-MP box is checked, a textbox will appear allowing you to enter the number of processes to run simultaneously.

Step 9

Click on the compare button. The number of licenses, if any, that are required for this run of BitMatch will be shown. You will have the ability to cancel the BitMatch run at this point without using up licenses.

You will be then asked to name the database that will be generated. To generate readable HTML reports from the database, see the section entitled Creating HTML Reports.

BitMatch Algorithms

The Algorithms

BitMatch searches binary files for all uninterrupted sequences of text characters. It then uses two CodeMatch algorithms to determine similarity between two source code files, first treating the text strings as program strings and then treating the text strings as identifiers. These algorithms are described below. When multiple files are compared, each match is given a weight and all weights are combined into a single matching score called the correlation score. The file pairs are then ranked by BitMatch score so that you can examine the most similar files.

Comment/string matching

BitMatch looks for identical comments and strings, ignoring whitespace. Comment lines and strings that contain only programming language keywords are still considered matches.

Identifier matching

BitMatch finds every instance in each file where identifiers match exactly. It eliminates programming language keywords and only reports matches for non-keyword identifiers such as variable names and function names.

BitMatch also finds every instance where an identifier in one file is part of a larger identifier in the other file. For example, the variable name "Index" in one file would partially match the variable names "NewIndex" and "Index1" in the other file. BitMatch eliminates programming language keywords and only reports matches for non-keyword identifiers such as variable names and function names.

Correlation Score

BitMatch produces a total correlation score based on the combination of above algorithms that the user chooses when running BitMatch. The minimum score is 0 while the maximum score is 100.



BitMatch Basic Report

Version: 1.0.1 | Date: 03/16/08 | Time: 15:35:00

SETTINGS | RESULTS | UNCOMPARED FILES | TOTALS

SETTINGS

Compare files in folder	C:\test\BitMatch\binary <i>Including subdirectories</i>
File types	*.exe
To files in folder	C:\test\BitMatch\C <i>Including subdirectories</i>
File types	*.c;*.h
Programming language	C
Reporting file threshold	8 files

RESULTS

C:\test\BitMatch\binary\CodeSuite.exe

Score	Compared to file
27	C:\test\BitMatch\C\LineCount.c
23	C:\test\BitMatch\C\CodeSuite.h
21	C:\test\BitMatch\C\CodeSuite.c

C:\test\BitMatch\binary\test1\test.exe

Score	Compared to file
22	C:\test\BitMatch\C\CodeSuite.c
21	C:\test\BitMatch\C\CodeSuite.h

C:\test\BitMatch\binary\test1\test0.exe

Score	Compared to file
23	C:\test\BitMatch\C\CodeSuite.c
22	C:\test\BitMatch\C\LineCount.c
19	C:\test\BitMatch\C\CodeSuite.h

TOTALS

Total number of bytes in files in folder 1 = 799957

Total number of bytes in files in folder 2 = 292584

Total run time = 50 Seconds



CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation



BitMatch Detailed Report

Version: 1.0.1 | Date: 03/16/08 | Time: 15:35:00

SETTINGS

Compare file 1:	C:\test\BitMatch\binary\LineCount\Debug\LineCount.exe
To file 2:	C:\test\BitMatch\C\LineCount.c
Links to results:	Matching Comments and Strings Matching Identifiers Score

RESULTS


Matching Comments and Strings		
File1 Line#	File2 Line#	Comment/String
5890 9304 9381 9755 9765	140	rB
14154	68	Total number of non-blank lines: %i
14155	67	Total number of blank lines: %i
14156	66	Total number of lines: %i
14157	65	Total number of Kbytes: %i
14158	64	Total number of files: %i




TOP

Matching Identifiers							
FILE	filepattern	folder	LineCount	name	NULL	path	size

stdio	string						
-------	--------	--	--	--	--	--	--

 TOP

Partially Matching Identifiers							
File1 Identifiers							
!CompareString	#File	(Press	.idata	arguments	blank	cchCount1	ClHandle
DLineCount	Domain	Ession	files	FindFirstFile	FindNextFile	HpDestroy	IG_LINE
LCMapStringA	LoadLibrary	MultiByte	osfinfo.c	SandleCount	stdargv.c	subfolders	
File2 Identifiers							
_A_SUBDIR	_finddata	_findfirst	_findnext	_LINE_LEN	argc	argv	attrib
BLineCount	CountLine	describe	extension	fddivide	file_id	FileCount	filestr
finfo	handle	InString	IsBlank	KByteCount	main	NonBlank	prefix
SepString	stdlib						

 TOP

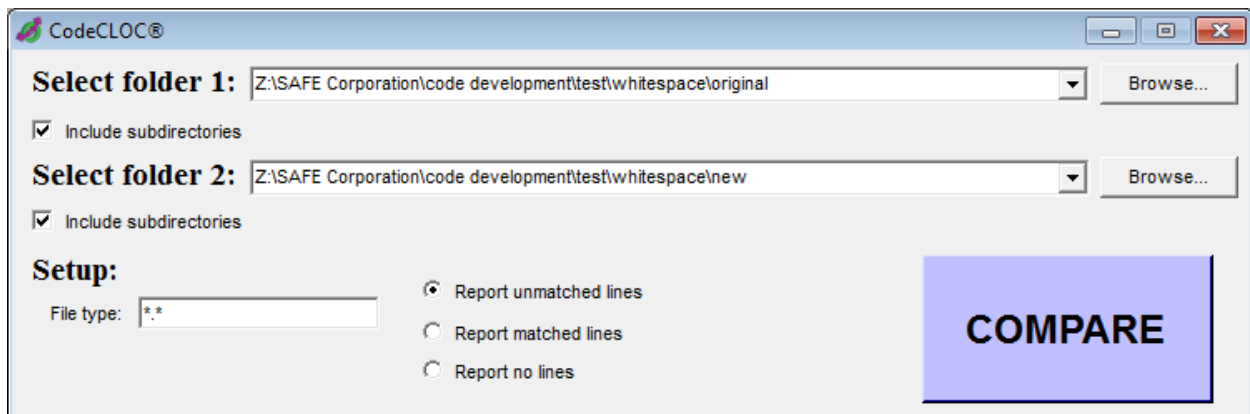
SCORE 71

CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation

CodeCLOC

Running CodeCLOC

CodeCLOC compares two versions of software source code files to determine code growth. A CodeCLOC comparison is essentially a CodeDiff comparison of files with the same name such that each file is used at most once and the scores are optimized such that the overall percentage of similarity between the two sets of files is maximized. Following that are step-by-step instructions for running CodeCLOC.



Step 1

Select the folders containing the versions of software to be compared by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories. Folder 1 contains the initial *base* version of source code files and folder 2 contains the subsequent *examined* version of source code files to compare to it to measure growth.

Step 2

Enter the file types separated by semicolons (e.g., *.cpp;*.c;*.h).

Step 3

Select one of two options for reporting lines in the database.

- **Report unmatched lines.** Report lines in either file that do not have a match in the other file. Note that if a line can be found three times in the first file and five times in the second file, two of the lines in the second file will be reported as unmatched.
- **Report matched lines.** Report lines that can be found in both files.
- **Report no lines.** Do not report any lines.

Step 4

Click on the compare button to start the analysis. A window will pop-up, see below, asking you to name the output file for your analysis. The number of licenses, if any, that are required for this run of CodeCLOC will be shown. You will have the ability to cancel the CodeCLOC run at this point without using up licenses.

You will be then asked to name the database that will be generated. To generate readable HTML reports from the database, see the section entitled Creating HTML Reports. To generate CLOC statistics from the database see the section entitled Creating CLOC Spreadsheets.

CodeCLOC Algorithm

CodeCLOC implements the Changing Lines of Code (CLOC) method for measuring source code growth from one version of a program to another. The CLOC method counts the number of lines of code that have been added, changed, or remain unchanged. These values are then combined to express the change in software as a rate of growth. The results can also be expressed in terms of the decay of the original code, which can be useful as a measure of how much original intellectual property still exists from that original code.

Measurements

The CLOC method uses the CodeDiff and FileCount functionality to create a CodeSuite database from which a specially developed CLOC spreadsheet can be generated.

The FileCount counts files, lines of code (LOC), and number of bytes in a directory tree. CLOC requires the number of program specific files and the number of non-blank lines in the software project's directory tree. CodeDiff exhaustively compares lines of code in one set of source code files to that in another set of source code files. CLOC requires that CodeDiff is used to compare same-name files from the original version to subsequent versions of the software project. CodeCLOC also prepares a post-CodeDiff optimization such that each file is used at most once in the comparison and the file pairs are selected to maximize the total similarity scores between the selected files. The selection and optimization only occurs for multiple files in the versions that have the same name.

Typically, movements of source code between files represents work being performed. Similarly, a file name change represents work being performed, because file names are not generally changed from version to version unless there is a significant change to the functionality of the file. The results of the CodeDiff analysis are then exported into a CodeSuite distribution report that contains the statistical information about changes in the files and LOC.

The generated CLOC spreadsheet shows statistics about the rate of software growth. The software evolution results can be summarized using these different percentages:

- **Percent of new and modified files.** Percentage of files that were added to the examined version or modified from the base to the examined version as a percentage of the total files in the examined version.
- **Percent of new and modified files relative to base version.** Percentage of files that were added to the examined version or modified from the base to the examined version as a percentage of the total files in the base version.

- **Percent of continuing files.** Percentage of files that continued, modified or not, from the base version to the examined version as a percentage of the total files in the examined version.
- **Percent of unchanged continuing files.** Percentage of files that continued unchanged from the base version to the examined version as a percentage of the total files in the examined version.
- **Percent CLOC change.** Percentage of changed lines of code from the base version to the examined version as a percentage of the total files in the examined version.
- **Percent CLOC change relative to base version.** Percentage of changed lines of code from the base version to the examined version as a percentage of the total files in the base version.
- **Percent LOC growth.** Percentage of total lines of code in the examined version as a percentage of the total lines of code in the base version.
- **Percent unchanged continuing lines of code.** Percentage of unchanged lines of code as a percentage of the total lines of code in the examined version.

S.A.F.E.



CodeCLOC Basic Report

Version: 4.0.0 | Date: 03/14/09 | Time: 17:48:01

SETTINGS | RESULTS | UNCOMPARED FILES | TOTALS

SETTINGS

Compare files in folder	C:\test\C\files 1 <i>Including subdirectories</i>
File types	*.*
To files in folder	C:\test\C\files 2 <i>Including subdirectories</i>
File types	*.*
Algorithms selected	<ul style="list-style-type: none"> • Ignoring case • Ignoring whitespace • Percentage of file pairs • Report matched lines
Reporting file threshold	1 file
Reporting score threshold	0

RESULTS

C:\test\C\files 1\aaa.c

Score	Compared to file
100	C:\test\C\files 2\aaa.c
80	C:\test\C\files 2\abc.c
4	C:\test\C\files 2\bpf_dump_semicolons.c
4	C:\test\C\files 2\Copy of semicolon_test.c
4	C:\test\C\files 2\semicolon_test.c

CodeSuite User's Guide

3	C:\test\C\files 2\bpf_dump_strings.c
3	C:\test\C\files 2\Copy of bpf_dump_strings.c

C:\test\C\files 1\aaa_case.c

Score	Compared to file
100	C:\test\C\files 2\aaa.c
80	C:\test\C\files 2\abc.c
4	C:\test\C\files 2\bpf_dump_semicolons.c
4	C:\test\C\files 2\Copy of semicolon_test.c
4	C:\test\C\files 2\semicolon_test.c
3	C:\test\C\files 2\bpf_dump_strings.c
3	C:\test\C\files 2\Copy of bpf_dump_strings.c

C:\test\C\files 1\aaa_with_comments.c

Score	Compared to file
100	C:\test\C\files 2\aaa_with_comments.c

C:\test\C\files 1\all_ints.c

Score	Compared to file
100	C:\test\C\files 2\all_ints.c

C:\test\C\files 1\all_specifiers.c

Score	Compared to file
100	C:\test\C\files 2\all_specifiers.c

TOTALS

Total number of bytes in files in folder 1 = 37651
Total number of bytes in files in folder 2 = 48944
Total run time = 3 Seconds



CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation



CodeCLOC Detailed Report

Version: 1.0.0 | Date: 03/14/09 | Time: 17:48:01

SETTINGS

Compare file 1:	C:\test\C\files 1\aaa_with_comments.c
To file 2:	C:\test\C\files 2\aaa_with_comments.c
Links to results:	Matched lines Score

RESULTS

Matching Lines		
File1 Line#	File2 Line#	Line
1	1	/* This is a comment*/ p = null;
2	2	private String auxonus = null; // This is a comment
3	3	p = /* This is a comment*/ null; /* This is a comment*/
4	4	/* Yes,
5	5	This is a comment*/ p = /* This is a comment*/ null; // This is a comment



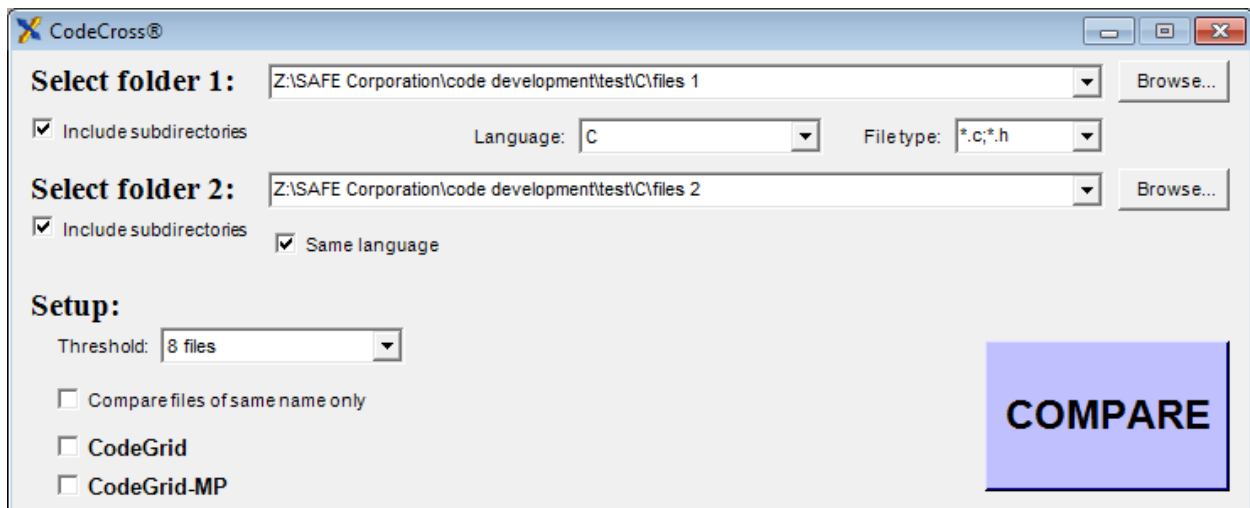
SCORE 100

CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation

CodeCross

Running CodeCross

CodeCross cross-compares statements in one set of files to comments in the other set of files, and vice versa, in order to find code that has been commented out. CodeCross finds areas of source code that were used as guides to develop other source code; it finds signs of copying that CodeMatch can miss. Below is a screen shot of the CodeCross form. Following that are step-by-step instructions for running CodeCross.



The screenshot shows the CodeCross application window. It has a title bar with the CodeCross logo and standard window controls. The main area contains the following elements:

- Select folder 1:** A text field containing "Z:\SAFE Corporation\code development\test\C\files 1" and a "Browse..." button.
- ☒ **Include subdirectories**
- Language:** A dropdown menu showing "C".
- Filetype:** A dropdown menu showing "*.c;*.h".
- Select folder 2:** A text field containing "Z:\SAFE Corporation\code development\test\C\files 2" and a "Browse..." button.
- ☒ **Include subdirectories**
- ☒ **Same language**
- Setup:**
 - Threshold:** A dropdown menu showing "8 files".
 - ☐ **Compare files of same name only**
 - ☐ **CodeGrid**
 - ☐ **CodeGrid-MP**
- A large blue button labeled **COMPARE** is located at the bottom right.

Step 1

Select the first folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 2

Select a source code language from the pulldown menu.

Step 3

Select the files types to compare from the pulldown menu. You can type over the suggested file types with your own file types. Separate multiple file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 4

Select the second folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 5

Choose whether both sets of files are written in the same programming language. If not, uncheck the box and you will be able to select the language and file types for the second set of files. CodeCross can compare files in different programming languages.

Step 6

Choose setup options to be used for comparing files.

Select the reporting threshold from the pulldown menu. This determines how many files are reported. CodeMatch reports only the most highly correlated files. By setting the number of files to report to a large number you may get a very large database. By setting the number of files to report to a small number, the database will be smaller, but it may not include all the similar files that you would like to see.

Step 7

Choose whether to only compare files if they have the same name. This will speed up the comparison significantly because far fewer combinations of files are compared.

Step 8

Check the CodeGrid box to run the comparison on a grid of computers if you have a license to do so. Check the CodeSuite-MP box to run the comparison on a multicore computer if you have a license to do so. When the CodeSuite-MP box is checked, a textbox will appear allowing you to enter the number of processes to run simultaneously.

Step 9

Click on the compare button. The number of licenses, if any, that are required for this run of CodeMatch will be shown. You will have the ability to cancel the CodeMatch run at this point without using up licenses.

You will be then asked to name the database that will be generated. To generate readable HTML reports from the database, see the section entitled Creating HTML Reports.

CodeCross Algorithm

CodeCross compares statements in one file to comments and strings in another file and calculates the number of complete matches as a percentage of the total number of statements, comments, and strings.

CodeCross Score

CodeCross produces a score that is a combined percentage of statements that match comments and strings and a percentage of comments and strings that match statements. The minimum score is 0 while the maximum score is 100.



CodeCross Basic Report

Version: 1.1.0 | Date: 12/29/08 | Time: 19:47:18

SETTINGS | RESULTS | UNCOMPARED FILES | TOTALS

SETTINGS

Compare files in folder	C:\test\CodeCross\files 1 <i>Including subdirectories</i>
File types	*.c;*.h
Programming language	C
To files in folder	C:\test\CodeCross\files 2 <i>Including subdirectories</i>
File types	*.c;*.h
Programming language	C
Reporting file threshold	8 files

RESULTS

C:\code development\test\CodeCross\files 1\bpf_dump_strings.c

Score	Compared to file
71	C:\test\CodeCross\files 2\bpf_dump_identifiers.c
71	C:\test\CodeCross\files 2\bpf_dump_mod.c
71	C:\test\CodeCross\files 2\bpf_dump_semicolons.c
71	C:\test\CodeCross\files 2\semicolon_test.c
12	C:\test\CodeCross\files 2\aaa_commented.c
2	C:\test\CodeCross\files 2\W32NReg_commented.c

C:\code development\test\CodeCross\files 1\bpf_image.c

Score	Compared to file
68	C:\test\CodeCross\files 2\bpf_dump_strings.c
2	C:\test\CodeCross\files 2\W32NReg_commented.c

C:\CodeCross\files 1\bpf_image_commented.c

Score	Compared to file
38	C:\test\CodeCross\files 2\bpf_dump_strings.c
24	C:\test\CodeCross\files 2\W32NReg_commented.c

TOTALS

Total number of bytes in files in folder 1 = 33829

Total number of bytes in files in folder 2 = 25147

Total run time = 2 Seconds



CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation



CodeCross Detailed Report

Version: 1.1.0 | Date: 12/29/08 | Time: 19:47:18

SETTINGS

Compare file 1:	C:\test\CodeCross\files 1\aaa_case.c
To file 2:	C:\test\CodeCross\files 2\aaa_commented.c
Links to Results:	Matching Statements to Comments Matching Comments to Statements Score

RESULTS

Matching Statements to Comments

File1 Line#	File2 Line#	Statement
1	1 4	P = Null;
2	2 5	Private String Auxonus = Null;



TOP

Matching Comments to Statements

File1 Line#	File2 Line#	Comment/String
3	6	* The Regents of the University of California. All rights reserved.
5	8	* Redistribution and use in source and binary forms, with or without
6	9	* modification, are permitted provided that: (1) source code distributions
7	10	* retain the above copyright notice and this paragraph in its entirety, (2)

8	11	* distributions including binary code include the above copyright notice and
9	12	* this paragraph in its entirety in the documentation or other materials
10	13	* provided with the distribution, and (3) all advertising materials mentioning



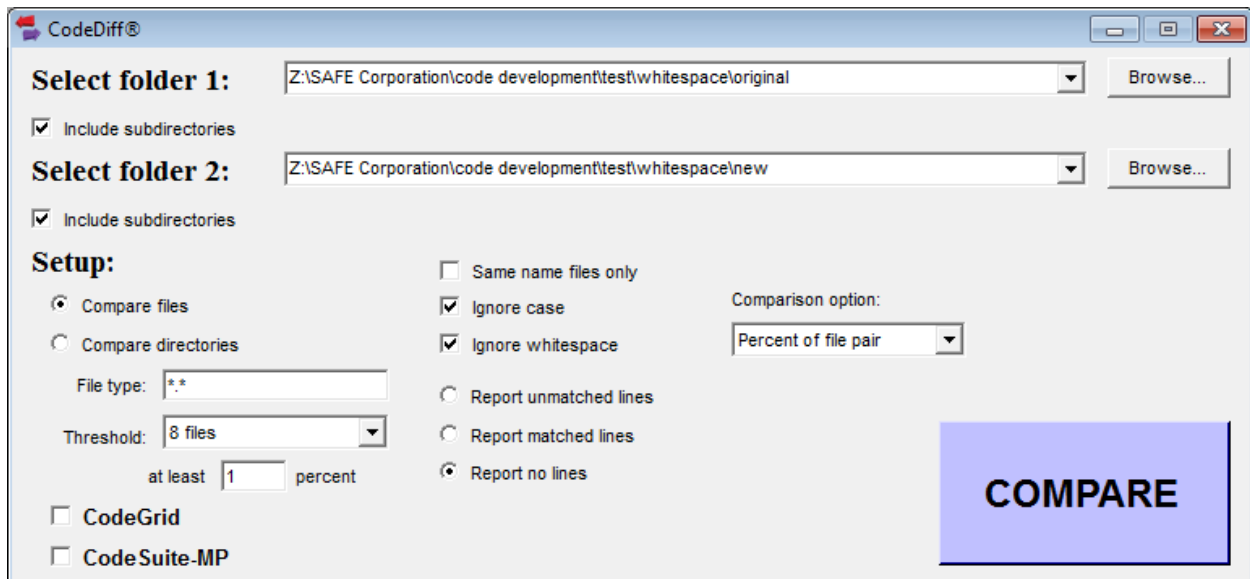
SCORE 100

CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation

CodeDiff

Running CodeDiff

CodeDiff compares files on a line by line basis to determine the percentage similarity. Below is a screen shot of the CodeDiff form. Following that are step-by-step instructions for running CodeDiff.



The screenshot shows the CodeDiff application window. It has a title bar with the CodeDiff logo and standard window controls. The main area is divided into sections: 'Select folder 1:' with a text field containing 'Z:\SAFE Corporation\code development\test\whitespace\original' and a 'Browse...' button; 'Select folder 2:' with a text field containing 'Z:\SAFE Corporation\code development\test\whitespace\new' and a 'Browse...' button; and a 'Setup:' section. In the 'Setup:' section, there are radio buttons for 'Compare files' (selected) and 'Compare directories'. Below these are a 'File type' field with '*.*' and a 'Threshold' dropdown set to '8 files' with a note 'at least 1 percent'. There are also checkboxes for 'Same name files only', 'Ignore case' (checked), 'Ignore whitespace' (checked), 'Report unmatched lines', 'Report matched lines', and 'Report no lines' (selected). A 'Comparison option:' dropdown is set to 'Percent of file pair'. At the bottom left are checkboxes for 'CodeGrid' and 'CodeSuite-MP'. A large blue 'COMPARE' button is on the right.

Step 1

Select the first folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 2

Select the second folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 3

Choose setup options to be used for comparing files.

You have a choice of two options.

- **Compare files.** CodeDiff will compare each pair of files in the directories specified and subdirectories if that option is selected. This option is used for comparing all combinations of files in all directories to find those that are similar.

- **Compare directories.** CodeDiff will compare only files with identical names in directories that have the same name and are in the same place in the directory tree. CodeDiff will also list all files in one directory that have no corresponding files in the other directory. This option is used for comparing entire directory trees to find similar files and missing files.

You have a choice of any combination of the following three options.

- **Same name files only.** CodeDiff will only compare files if they have the same exact name (not case sensitive). This option is not available when comparing directories, because only files of the same name are compared when comparing directories.
- **Ignore case.** CodeDiff will consider two lines matching if they are identical in all other respects even if the letter cases are different.
- **Ignore whitespace.** Before performing a comparison on any lines, CodeDiff will reduce all sequences of whitespace characters (space or tab) to a single space.

You have a choice of two options for reporting lines in the database.

- **Report unmatched lines.** Report lines in either file that do not have a match in the other file. Note that if a line can be found three times in the first file and five times in the second file, two of the lines in the second file will be reported as unmatched.
- **Report matched lines.** Report lines that can be found in both files.
- **Report no lines.** Do not report any lines.

You have a choice of three comparison options.

- **Percentage of file pair.** The percentage generated by CodeDiff will be the percentage of lines in the two files that match with respect to the total number of lines in both files.
- **Percentage of first file.** The percentage generated by CodeDiff will be the percentage of lines in the first file that match a line in the second file with respect to the total number of lines in the first file.
- **Percentage of second file.** The percentage generated by CodeDiff will be the percentage of lines in the second file that match a line in the first file with respect to the total number of lines in the second file.

Step 4

Specify the types of files to compare. Separate different file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 5

Select the reporting threshold from the pulldown menu. This determines how many files are reported. CodeDiff reports only the most similar files. By setting the number of files to report to a large number you may get a very large database. By setting the number of

files to report to a small number, the database will be smaller, but it may not include all the similar files that you would like to see.

Specify the minimum percentage matching to report. CodeDiff reports the percentage of lines that are identical between files, ranging from 0 to 100. If you specify 0 as the minimum threshold, even files that do not have any matching lines will be reported. If you specify 100 as the minimum threshold, only files whose lines all match exactly will be reported.

Step 6

Check the CodeGrid box to run the comparison on a grid of computers if you have a license to do so. Check the CodeSuite-MP box to run the comparison on a multicore computer if you have a license to do so. When the CodeSuite-MP box is checked, a textbox will appear allowing you to enter the number of processes to run simultaneously.

Step 7

Click on the compare button. The number of licenses, if any, that are required for this run of CodeDiff will be shown. You will have the ability to cancel the CodeDiff run at this point without using up licenses.

You will be then asked to name the database that will be generated. To generate readable HTML reports from the database, see the section entitled Creating HTML Reports.

CodeDiff Algorithm

CodeDiff compares each line of code in two sets of files and calculates the number of lines of code that match completely as a percentage of the total number of lines of code. The order of the lines is not considered so if a file were compared to an identical copy where the statements were all in a different order, this would still result in a 100% match.

If CodeDiff is set to ignore case, lines are considered matches even if the letters have different cases.

If CodeDiff is set to ignore whitespace, all sequences of whitespace (spaces and tabs) are converted to a single space before the comparison is performed.

If CodeDiff is set to generate the percentage of file pairs, it will generate the percentage of lines in the two files that match with respect to the total number of lines in both files. If CodeDiff is set to generate the percentage of the first file, it will generate the percentage of lines in the first file that match a line in the second file with respect to the total number of lines in the first file.

Similarity Score

CodeDiff produces a similarity score that is a percentage of matching line within the files. The minimum score is 0 while the maximum score is 100.



CodeDiff Basic Report

Version: 4.0.0 | Date: 03/14/09 | Time: 17:48:01

SETTINGS | RESULTS | UNCOMPARED FILES | TOTALS

SETTINGS

Compare files in folder	C:\test\C\files 1 <i>Including subdirectories</i>
File types	*.*
To files in folder	C:\test\C\files 2 <i>Including subdirectories</i>
File types	*.*
Algorithms selected	<ul style="list-style-type: none"> • Ignoring case • Ignoring whitespace • Percentage of file pairs • Report matched lines
Reporting file threshold	8 files
Reporting score threshold	1

RESULTS

C:\test\C\files 1\aaa.c

Score	Compared to file
100	C:\test\C\files 2\aaa.c
80	C:\test\C\files 2\abc.c
4	C:\test\C\files 2\bpf_dump_semicolons.c
4	C:\test\C\files 2\Copy of semicolon_test.c
4	C:\test\C\files 2\semicolon_test.c

CodeSuite User's Guide

3	C:\test\C\files 2\bpf_dump_strings.c
3	C:\test\C\files 2\Copy of bpf_dump_strings.c

C:\test\C\files 1\aaa_case.c

Score	Compared to file
100	C:\test\C\files 2\aaa.c
80	C:\test\C\files 2\abc.c
4	C:\test\C\files 2\bpf_dump_semicolons.c
4	C:\test\C\files 2\Copy of semicolon_test.c
4	C:\test\C\files 2\semicolon_test.c
3	C:\test\C\files 2\bpf_dump_strings.c
3	C:\test\C\files 2\Copy of bpf_dump_strings.c

C:\test\C\files 1\aaa_with_comments.c

Score	Compared to file
100	C:\test\C\files 2\aaa_with_comments.c

C:\test\C\files 1\all_ints.c

Score	Compared to file
100	C:\test\C\files 2\all_ints.c

C:\test\C\files 1\all_specifiers.c

Score	Compared to file
100	C:\test\C\files 2\all_specifiers.c

TOTALS

Total number of bytes in files in folder 1 = 37651
Total number of bytes in files in folder 2 = 48944
Total run time = 3 Seconds



CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation



CodeDiff Detailed Report

Version: 4.0.0 | Date: 03/14/09 | Time: 17:48:01

SETTINGS

Compare file 1:	C:\test\C\files 1\aaa_with_comments.c
To file 2:	C:\test\C\files 2\aaa_with_comments.c
Links to results:	Matched lines Score

RESULTS

Matching Lines		
File1 Line#	File2 Line#	Line
1	1	/* This is a comment*/ p = null;
2	2	private String auxonus = null; // This is a comment
3	3	p = /* This is a comment*/ null; /* This is a comment*/
4	4	/* Yes,
5	5	This is a comment*/ p = /* This is a comment*/ null; // This is a comment



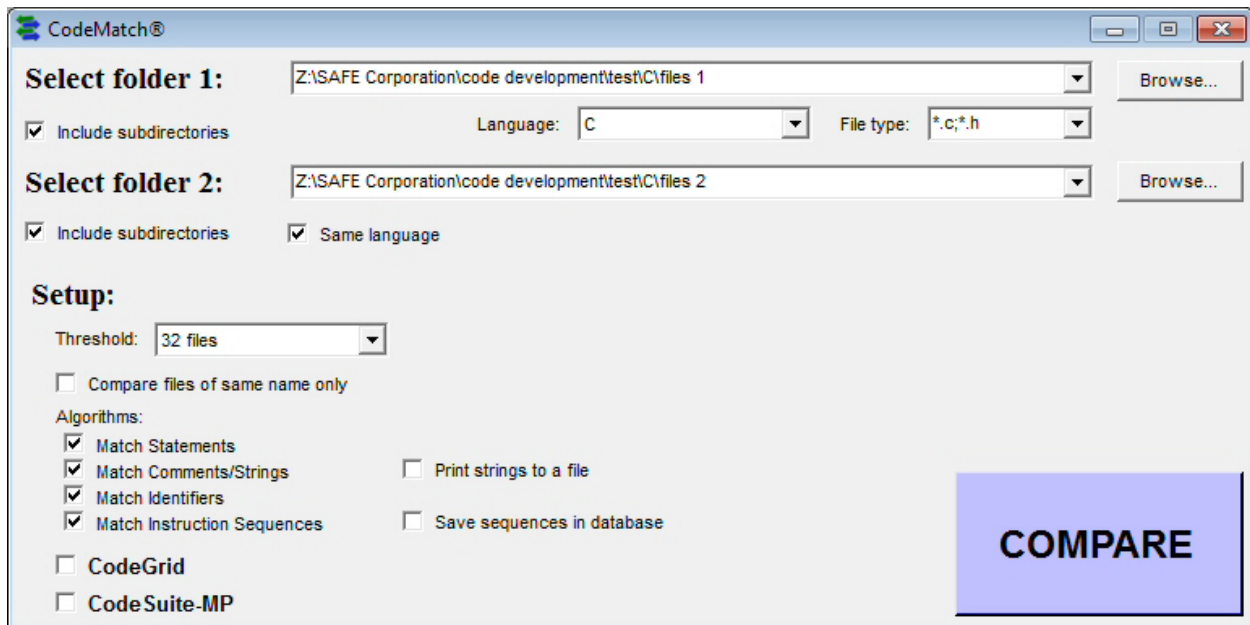
SCORE 100

CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation

CodeMatch

Running CodeMatch

CodeMatch compares files using a set of algorithms to determine their correlation. Below is a screen shot of the CodeMatch form. Following that are step-by-step instructions for running CodeMatch.



The screenshot shows the CodeMatch application window. It has a title bar with the CodeMatch logo and standard window controls. The main area is divided into sections for selecting folders, setting options, and a large 'COMPARE' button.

Select folder 1: Z:\SAFE Corporation\code development\test\C\files 1 [Browse...]

☒ Include subdirectories Language: C File type: *.c;*.h

Select folder 2: Z:\SAFE Corporation\code development\test\C\files 2 [Browse...]

☒ Include subdirectories ☒ Same language

Setup:

Threshold: 32 files

☐ Compare files of same name only

Algorithms:

☒ Match Statements ☐ Print strings to a file

☒ Match Comments/Strings

☒ Match Identifiers ☐ Save sequences in database

☒ Match Instruction Sequences

☐ CodeGrid

☐ CodeSuite-MP

COMPARE

Step 1

Select the first folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 2

Select a source code language from the pulldown menu.

Step 3

Select the files types to compare from the pulldown menu. You can type over the suggested file types with your own file types. Separate multiple file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 4

Select the second folder for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 5

Choose whether both sets of files are written in the same programming language. If not, uncheck the box and you will be able to select the language and file types for the second set of files. CodeMatch can compare files in different programming languages.

Step 6

Choose setup options to be used for comparing files.

Select the reporting threshold from the pulldown menu. This determines how many files are reported. CodeMatch reports only the most highly correlated files. By setting the number of files to report to a large number you may get a very large database. By setting the number of files to report to a small number, the database will be smaller, but it may not include all the similar files that you would like to see.

Step 7

Choose whether to only compare files if they have the same name. This will speed up the comparison significantly because far fewer combinations of files are compared.

Step 8

Choose setup options and algorithms to be used for comparing files. You have a choice of any combination of four algorithms. For more information on each of these algorithms, see the section entitled CodeMatch Algorithms.

- **Statement Matching**
- **Comment/String Matching**
- **Identifier Matching**
- **Instruction Sequence Matching**

Note that when you select instruction sequence matching, you will have the option to record the instruction sequences in the resulting database. Putting the sequences in the database makes it easier to determine whether the sequences are interesting but will produce a significantly larger database.

Also note that when you select comment matching, you will have the option to print strings into a file. This allows you to create a text file with a list of all strings that are found in both sets of code being compared.

Step 9

Check the CodeGrid box to run the comparison on a grid of computers if you have a license to do so. Check the CodeSuite-MP box to run the comparison on a multicore computer if you have a license to do so. When the CodeSuite-MP box is checked, a textbox will appear allowing you to enter the number of processes to run simultaneously.

Step 10

Click on the compare button. The number of licenses, if any, that are required for this run of CodeMatch will be shown. You will have the ability to cancel the CodeMatch run at this point without using up licenses.

You will be then asked to name the database that will be generated. To generate readable HTML reports from the database see the section entitled Creating HTML Reports.

CodeMatch Algorithms

The Algorithms

CodeMatch uses several algorithms to determine similarity between two source code files. These algorithms are described below. When multiple files are compared, each match is given a weight and all weights are combined into a single matching score called the correlation score. The file pairs are then ranked by correlation score so that you can examine the most similar files.

Statement matching

CodeMatch looks for identical program statements (i.e., functional source code), ignoring whitespace and eliminating comments and strings. Statements that contain only programming language keywords are not considered matching. For statements to be considered matches, they must contain at least one identifier (non-keyword) such as a variable name or function name.

Comment/string matching

CodeMatch looks for identical comments and strings, ignoring whitespace. Comment lines and strings that contain only programming language keywords are still considered matches.

Instruction sequence matching

CodeMatch looks for sequences of instructions that match. CodeMatch notes the longest such sequence in each pair of files. A sequence matches if the initial programming language statement on each line is identical, regardless of what follows it. Even if variable names are altered in one file, CodeMatch will report similarities in the files. The following shows an example of two identical instruction sequences in C:

```
// File 1
if (x == 5)
{
    // Loop on j here
    for (j = 0; j < Index; j++)
        printf("x = %i", j);
}
else
    break; // Here's the break

// File 2
if (xyz < 2)
    for (jjj = 0; jjj < i; jjj++)
    {
```

```
        printf("Hello world\n");
    }
    else
        break;
```

Identifier matching

CodeMatch finds every instance in each file where identifiers match exactly. It eliminates programming language keywords and only reports matches for non-keyword identifiers such as variable names and function names.

CodeMatch also finds every instance where an identifier in one file is part of a larger identifier in the other file. For example, the variable name "Index" in one file would partially match the variable names "NewIndex" and "Index1" in the other file.

CodeMatch eliminates programming language keywords and only reports matches for non-keyword identifiers such as variable names and function names.

Correlation Score

CodeMatch produces a total correlation score based on the combination of above algorithms that the user chooses when running CodeMatch. The minimum score is 0 while the maximum score is 100.



CodeMatch Basic Report

Version: 5.3.1 | Date: 08/28/08 | Time: 11:33:11

SETTINGS | RESULTS | UNCOMPARED FILES | TOTALS

SETTINGS

Compare files in folder	C:\test\C\files 1 <i>Including subdirectories</i>
File types	*.c;*.h
Programming language	C
To files in folder	C:\test\C\files 2 <i>Including subdirectories</i>
File types	*.c;*.h
Programming language	C
Algorithms selected	<ul style="list-style-type: none"> • Statement Matching • Comment/String Matching • Identifier Matching • Instruction Sequence Matching
Reporting file threshold	8 files

RESULTS

C:\test\C\files 1\aaa.c

Score	Compared to file
82	C:\test\C\files 2\aaa.c
82	C:\test\C\files 2\abc.c
71	C:\test\C\files 2\aaa_with_comments.c
15	C:\test\C\files 2\svn\bpf_image.c

CodeSuite User's Guide

9	C:\test\C\files 2\all_specifiers.c
9	C:\test\C\files 2\bpf_dump_semicolons.c
9	C:\test\C\files 2\bpf_dump_strings.c
9	C:\test\C\files 2\semicolon_test.c

C:\test\C\files 1\aaa_case.c

Score	Compared to file
82	C:\test\C\files 2\aaa.c
82	C:\test\C\files 2\abc.c
71	C:\test\C\files 2\aaa_with_comments.c
15	C:\test\C\files 2\W32NReg.c
15	C:\test\C\files 2\svn\W32NReg (no comments).c
15	C:\test\C\files 2\svn\W32NReg (variable names changed).c
15	C:\test\C\files 2\svn\W32NReg.c
13	C:\test\C\files 2\svn\bpf_image.c

C:\test\C\files 1\aaa_whitespace.c

Score	Compared to file
82	C:\test\C\files 2\aaa.c
82	C:\test\C\files 2\abc.c
71	C:\test\C\files 2\aaa_with_comments.c
15	C:\test\C\files 2\svn\bpf_image.c
9	C:\test\C\files 2\all_specifiers.c
9	C:\test\C\files 2\bpf_dump_semicolons.c
9	C:\test\C\files 2\bpf_dump_strings.c
9	C:\test\C\files 2\semicolon_test.c

C:\test\C\files 1\aaa_with_comments.c

Score	Compared to file
87	C:\test\C\files 2\aaa_with_comments.c
71	C:\test\C\files 2\aaa.c
69	C:\test\C\files 2\abc.c

15	C:\test\C\files 2\svn\bpf_image.c
8	C:\test\C\files 2\all_specifiers.c
8	C:\test\C\files 2\W32NReg.c
8	C:\test\C\files 2\svn\W32NReg (no comments).c
8	C:\test\C\files 2\svn\W32NReg (variable names changed).c

C:\test\C\files 1\all_ints.c

Score	Compared to file
82	C:\test\C\files 2\all_ints.c
17	C:\test\C\files 2\all_specifiers.c
11	C:\test\C\files 2\W32NReg.c
11	C:\test\C\files 2\svn\W32NReg (no comments).c
11	C:\test\C\files 2\svn\W32NReg.c

TOTALS

Total number of bytes in files in folder 1 = 37651
Total number of bytes in files in folder 2 = 37627
Total run time = 16 Seconds



CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation

**CodeMatch Detailed Report**

Version: 5.3.1 | Date: 08/28/08 | Time: 11:33:11

SETTINGS

Compare file 1:	C:\test\C\files 1\bpf_image.c
To file 2:	C:\test\C\files 2\svn\bpf_image.c
Links to results:	Matching Statements Matching Comments and Strings Matching Instruction Sequences Matching Identifiers Partially Matching Identifiers Score

RESULTS

Matching Statements		
File1 Line#	File2 Line#	Statement
22	22	#include <windows.h>
23	23	#include <sys/types.h>
35	35	char *fmt, *op
36	36	static char image[256]
37	37	char operand[64]
39	39	v = p->k
40	40	switch (p->code) {
199 204 209 214	199	case BPF_ALU BPF_OR BPF_X:

254	254 259 264 269 270	case BPF_ALU BPF_NEG:
-----	---------------------------------	-----------------------



Matching Comments and Strings

File1 Line#	File2 Line#	Comment/String
2	2	* Copyright (c) 1990, 1991, 1992, 1994, 1995, 1996
3	3	* The Regents of the University of California. All rights reserved.
5	5	* Redistribution and use in source and binary forms, with or without
6	6	* modification, are permitted provided that: (1) source code distributions
7	7	* retain the above copyright notice and this paragraph in its entirety, (2)
8	8	* distributions including binary code include the above copyright notice and
9	9	* this paragraph in its entirety in the documentation or other materials
10	10	* provided with the distribution, and (3) all advertising materials mentioning
11	11	* features or use of this software display the following acknowledgement:



Matching Instruction Sequences

File1 Line#	File2 Line#	Number of matching instructions
22	22	202
43	129	71
46	51	64
46	56	60
46	61	56
46	66	52
46	71	48
46	76	44
46	81	40
46	86	36
46	91	32



Matching Identifiers

256	64	BPF_A	BPF_ABS	BPF_ADD	BPF_ALU	BPF_AND	BPF_B
BPF_CLASS	BPF_DIV	BPF_H	bpf_image	BPF_IMM	BPF_IND	bpf_insn	BPF_JA
BPF_JEQ	BPF_JGE	BPF_JGT	BPF_JMP	BPF_JSET	BPF_K	BPF_LD	BPF_LDX
BPF_LEN	BPF_LSH	BPF_MEM	BPF_MISC	BPF_MSH	BPF_MUL	BPF_NEG	BPF_OP
BPF_OR	BPF_RET	BPF_RSH	BPF_ST	BPF_STX	BPF_SUB	BPF_TAX	BPF_TXA
BPF_W	BPF_X	code	fnt	image	INT	jf	jt
op	operand	stdio	string	sys	types	windows	



Partially Matching Identifiers

File1 Identifiers

0x00FF	BPF_ALU	bpf_filter	BPF_IMM	BPF_IND	BPF_LEN	BPF_MEMWORDS	BPF_RET
BPF_ST	BPF_SUB	EXTRACT_LONG	INT	netlong	types	UCHAR	W32N_htonl
winsock							

File2 Identifiers

0x0004	0x0005	__stdcall	_TEXT	_W32N_ADA	_WAdapter0	_WAdapter1	_WAdapter2
dwDataLen	DWORD	dwType	ERR_IMPLIED	ERR_SUCCESS	H_LOCAL	hAdapter	hClassNet
KEY_READ	LONG	pAdapterInfo	PCHAR	PW_ADAPTER	QueryValue	TChar	VER_WIN32
W0Adapter	W0Window	W0Windows	W32N_Adapt	W32N_NET	WINCARDS	wsprintf	



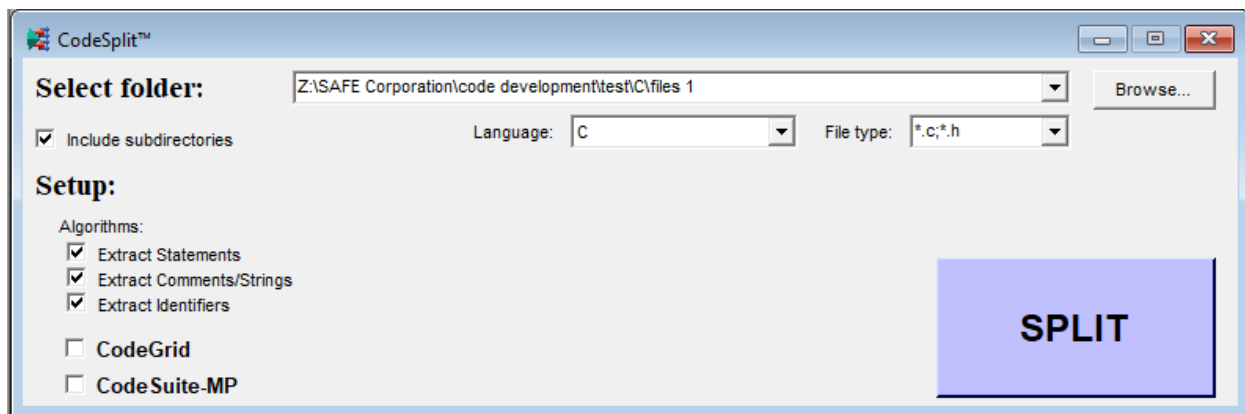
SCORE 100

CodeSuite copyright 2003-2010 by Software Analysis and Forensic Engineering Corporation

CodeSplit

Running CodeSplit

CodeSplit takes all source code files in multiple directories and subdirectories and splits them into basic elements (statements, comments, strings, and identifiers). Following that are step-by-step instructions for running CodeSplit.



Step 1

Select the folder of files for splitting for comparison by clicking on the browse button or entering the path in the text field. Check the box to include files in all subdirectories.

Step 2

Select a source code language from the pulldown menu.

Step 3

Select the files types to compare from the pulldown menu. You can type over the suggested file types with your own file types. Separate multiple file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 4

Choose setup options to be used for splitting source code files. You have a choice of any combination of three options to split statements, comments, strings, and identifiers from the source code.

- **Extract Statements**
- **Extract Comments/Strings**
- **Extract Identifiers**

Step 5

Check the CodeGrid box to run the split on a grid of computers if you have a license to do so. Check the CodeSuite-MP box to run the split on a multicore computer if you have a license to do so. When the CodeSuite-MP box is checked, a textbox will appear allowing you to enter the number of processes to run simultaneously.

Step 6

Click on the split button. The number of licenses, if any, that are required for this run of CodeSplit will be shown. You will have the ability to cancel the CodeSplit run at this point without using up licenses.

You will be then asked to name the database that will be generated. To generate readable HTML reports from the database, see the section entitled Creating HTML Reports.



CodeSplit Basic Report

Version: 1.0.0 | Date: 06/29/18 | Time: 08:41:28

SETTINGS

Split files in folder	C:\test\C\files 1 <i>Including subdirectories</i>
File types	*.c;*.h
Programming language	C
Algorithms selected	<ul style="list-style-type: none"> • Extract Statements • Extract Comments and Strings • Extract Identifiers

RESULTS

Z:\SAFE Corporation\code development\test\C\files 3\aaa.c

Report

Z:\SAFE Corporation\code development\test\C\files 3\abc.c

Report

Z:\SAFE Corporation\code development\test\C\files 3\abc.c

Report

Z:\SAFE Corporation\code development\test\C\files 3\abc.c

Report

TOTALS

Total number of bytes in files in folder 1 = 37651

Total run time = 16 Seconds



CodeSuite copyright 2003-2018 by Software Analysis and Forensic Engineering Corporation



CodeSplit Detailed Report

Version: 1.0.0 | Date: 06/29/18 | Time: 08:41:28

SETTINGS

Analyzing file:	Z:\SAFE Corporation\code development\test\C\files 3\aaa.c
Links to results:	Statements Comments and Strings Identifiers

RESULTS

Statements	
Line#	Statement
22	#include <windows.h>
23	#include <sys/types.h>
35	char *fmt, *op
36	static char image[256]
37	char operand[64]
39	v = p->k
40	switch (p->code) {
199 204 209 214	case BPF_ALU BPF_OR BPF_X:
254	case BPF_ALU BPF_NEG:



TOP

Comments and Strings

Line#	Comment/String
2	* Copyright (c) 1990, 1991, 1992, 1994, 1995, 1996
3	* The Regents of the University of California. All rights reserved.
5	* Redistribution and use in source and binary forms, with or without
6	* modification, are permitted provided that: (1) source code distributions
7	* retain the above copyright notice and this paragraph in its entirety, (2)
8	* distributions including binary code include the above copyright notice and
9	* this paragraph in its entirety in the documentation or other materials
10	* provided with the distribution, and (3) all advertising materials mentioning
11	* features or use of this software display the following acknowledgement:



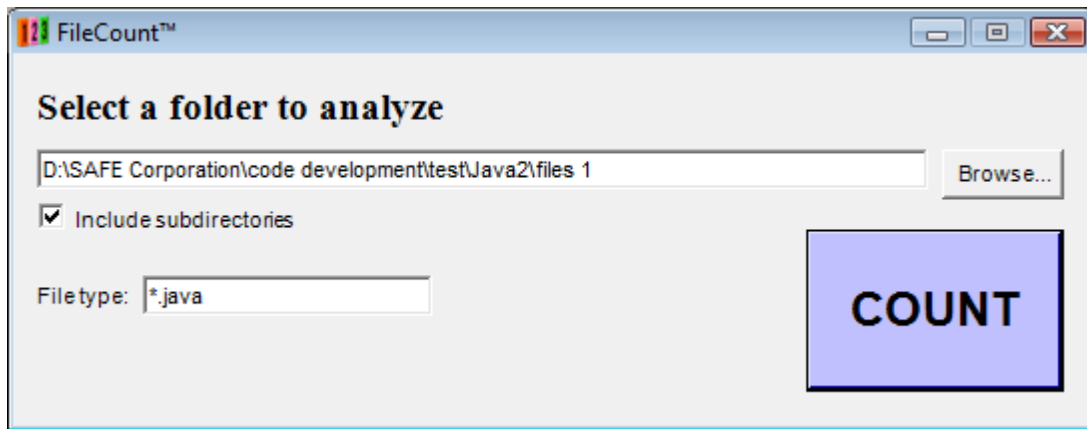
Identifiers							
256	64	BPF_A	BPF_ABS	BPF_ADD	BPF_ALU	BPF_AND	BPF_B
BPF_CLASS	BPF_DIV	BPF_H	bpf_image	BPF_IMM	BPF_IND	bpf_insn	BPF_JA
BPF_JEQ	BPF_JGE	BPF_JGT	BPF_JMP	BPF_JSET	BPF_K	BPF_LD	BPF_LDX
BPF_LEN	BPF_LSH	BPF_MEM	BPF_MISC	BPF_MSH	BPF_MUL	BPF_NEG	BPF_OP
BPF_OR	BPF_RET	BPF_RSH	BPF_ST	BPF_STX	BPF_SUB	BPF_TAX	BPF_TXA
BPF_W	BPF_X	code	fnt	image	INT	jf	jt
op	operand	stdio	string	sys	types	windows	



FileCount

Running FileCount

FileCount is a utility that counts the number of files, non-blank lines, and bytes in a large set of files in a directory tree. FileCount is useful when using CodeDiff to generate statistics about a set of source code files.



Step 1

Select the folder where the files are that need to be counted by clicking on the browse button or entering the path in the text field. Check the box to include all subdirectories.

Step 2

Type in the file types. Separate different file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 3

Press the count button. FileCount will then search the directory and all subdirectories, if specified, counting all of the files that meet the file type, and counting the total number of non-blank lines and bytes. When complete, a dialog box will appear with these counts.

FileIdentify

Running FileIdentify

FileIdentify searches a directory or directory tree and lists all of the file types found, based on the file name extensions. It also reports all known programming language files based on the file types. Below is a screen shot of the FileIdentify form and step-by-step instructions for running FileIdentify.



Step 1

Select the folder where the files are located that you want to analyze. Check the box to include all subdirectories if you want to analyze files in the subfolders also.

Step 2

Press the go button. You will be asked for the file name and location for a spreadsheet showing all file types and their associated programming languages, if known. FileIdentify will then search the directory and all subdirectories, if specified.

Below is an example of a spreadsheet created by FileIdentify.

	A	B	C
1	Analysis of Extensions		
2	Analysis date	12/16/2012	
3	Folder	Z:\SAFE Corporation\code development\test\FileIdentify\top	
4	Include subfolders	Yes	
5			
6	Files with no extension	0	
7	Files with an empty extension	0	
8	Folder paths too long	0	
9	File paths too long	0	

10			
11	File types	Number of files	Language (if known)
12	.as	37	ActionScript
13	.c	81	C
14	.cdb	6	
15	.csf	1	
16	.flr	1	
17	.gif	47	
18	.htm	181	
19	.jpg	8	
20	.js	413	JavaScript
21	.mako	1	
22	.php	8	PHP
23	.png	49	
24	.swf	517	
25	.txt	66	

The top line shows that the spreadsheet was an analysis of file extensions created by FileIdentify. The second line shows the date that the analysis was run. The third shows the folder name. The fourth line indicates whether or not subfolders were included in the analysis.

Line 6 gives the number of files that had no extension while line 7 gives the number of files that had an empty extension, meaning the file name ended in a dot. Line 8 gives the number of folders that exceeded the maximum number of characters and could thus not be examined while line 9 gives the number of file paths, meaning the folder name plus the file name, that exceeded the maximum number of characters and could thus not be examined.

Lines 12 through 25 show the file types that were found, in column A, the number of files for each file type, in column B, and the programming language, if known, in column C.

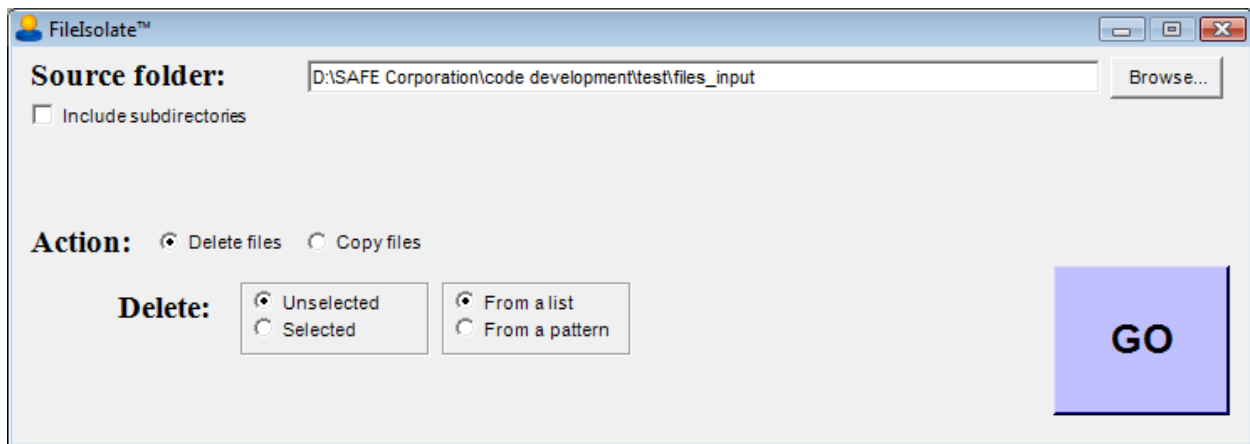
FileIsolate

Running FileIsolate

FileIsolate allows files and file types to be selectively copied or deleted from an entire directory tree.

Deleting files

Below is a screen shot of the FileIsolate form where the option is selected to delete files. Following that are step-by-step instructions for running FileIsolate to delete files.



Step 1

Choose the **Delete files** action.

Step 2

Select the folder where the files are that need to be deleted by clicking on the browse button or entering the path in the text field. Check the box to include all subdirectories.

Step 3

Choose options for deleting files.

- **Unselected files.** Choose this option to delete all files and file types that are not selected.
- **Selected files.** Choose this option to delete all files and file types that are selected.

Choose options for selecting files.

- **From a list of files.** Choose this option to select all files that are named in a text file. You will be prompted for the file containing the list of files. All files that have a name in this list will be selected.
- **From a file pattern.** Choose this option to select files whose names fit a pattern. A field will appear that allows you to type in file patterns. Separate different file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 4

Press the go button. FileIsolate will then search the directory and all subdirectories, if specified. FileIsolate will delete all selected files or delete all files that were not selected, depending on the options specified.

Copying files

Below is a screen shot of the FileIsolate form where the option is selected to copy files. Following that are step-by-step instructions for running FileIsolate to copy files.

The screenshot shows the FileIsolate application window. The 'Source folder' is set to 'D:\SAFE Corporation\code development\test\files_input' and the 'Destination folder' is set to 'D:\SAFE Corporation\code development\test\files_output'. The 'Action' is set to 'Copy files'. Under the 'Copy' section, 'Unselected' is selected. The 'File pattern' is set to '*.txt'. A large blue 'GO' button is visible on the right side of the form.

Step 1

Choose the **Copy files** action.

Step 2

Select the source folder where the files are that need to be copied by clicking on the browse button or entering the path in the text field. Check the box to include all subdirectories.

Step 3

Select the destination folder where the files are to be copied by clicking on the browse button or entering the path in the text field. If the destination folder does not exist, it will be created.

Step 4

Choose options for copying files.

- **Unselected files.** Choose this option to copy all files and file types that are not selected.
- **Selected files.** Choose this option to copy all files and file types that are selected.

Choose options for selecting files.

- **From a list of files.** Choose this option to select all files that are named in a text file. You will be prompted for the file containing the list of files. All files that have a name in this list will be selected.
- **From a file pattern.** Choose this option to select files whose names fit a pattern. A field will appear that allows you to type in file patterns. Separate different file types with a semicolon. Use the * and ? wildcard characters if needed.

Step 5

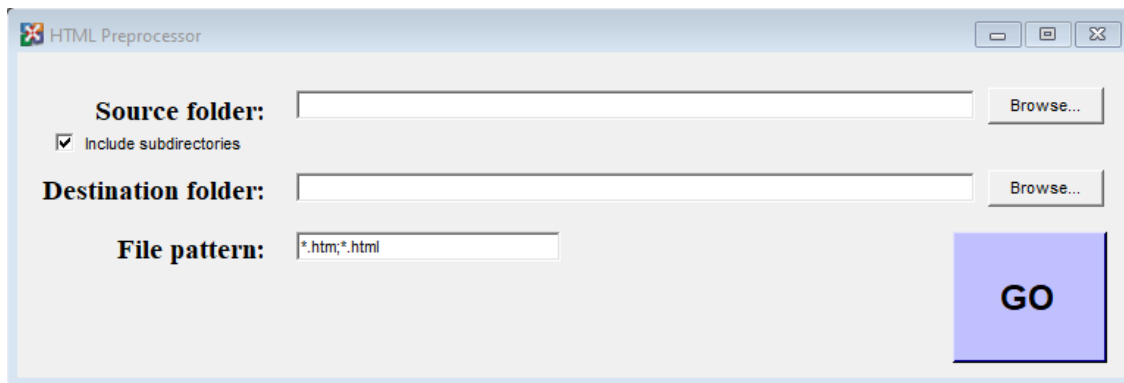
Press the go button. FileIsolate will then search the directory and all subdirectories, if specified. FileIsolate will copy all selected files or copy all files that were not selected, depending on the options specified.

HTML Preprocessor

Running HTML Preprocessor

HTML Preprocessor is a utility that transforms HTML files into files that are amenable to analysis by CodeSuite, DocMatch, and other source code analysis tools. HTML Preprocessor outputs nicely formatted HTML files without embedded scripts, text files containing text blocks within the original HTML files, and script files containing scripts within the original HTML files. The scripts are assumed to be JavaScript, though they may contain other languages if the original HTML files contain other languages.

Below is a screen shot of the HTML Preprocessor form. Following are step-by-step instructions for running HTML Preprocessor.

A screenshot of the HTML Preprocessor application window. The window has a title bar with the text "HTML Preprocessor" and standard window controls. The main area contains three input fields: "Source folder:" with a "Browse..." button, "Destination folder:" with a "Browse..." button, and "File pattern:" with a text box containing "*.htm;*.html". There is a checked checkbox labeled "Include subdirectories" between the first two fields. A large blue "GO" button is located at the bottom right of the form.

Step 1

Select the source folder containing the original HTML files by clicking on the browse button or entering the path in the text field. Check the box to include all subdirectories.

Step 2

Select the destination folder to save the new files by clicking on the browse button or entering the path in the text field.

Step 3

Select the file patterns of the HTML files in the source folder.

Step 4

Press the go button. HTML Preprocessor will then search the directory and all subdirectories, if specified, and divide up the HTML files into pure HTML files, pure text,

and pure script files and save them in the destination folder. The subdirectories will be maintained in the new folder. A file with name `filename.html` will produce three new files with these names:

pure HTML: `filename_out.html`

pure text: `filename_out.txt`

pure script: `filename_out.js`

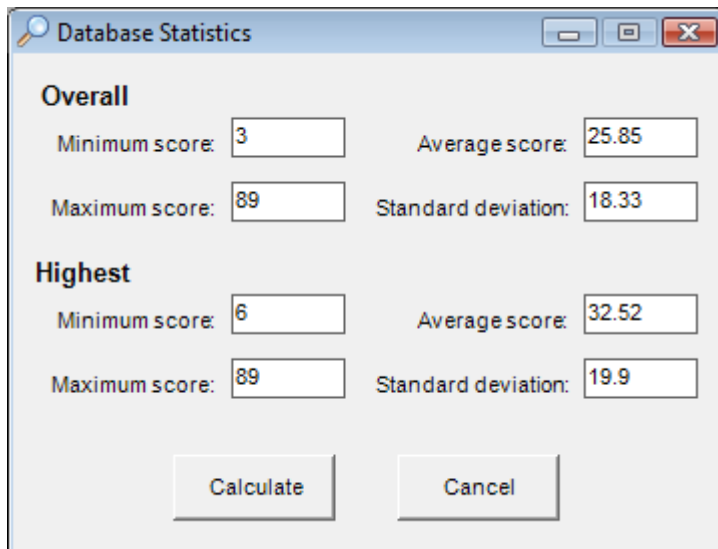
Statistics

Calculating Statistics

The following statistics can be calculated from a CodeSuite database:

- Minimum score of all of the scores in the database.
- Maximum score of all of the scores in the database.
- Average score of all of the scores in the database.
- Standard deviation of all of the scores in the database.
- Minimum score of the highest scoring file pairs for each file in folder 1 in the database.
- Maximum score of the highest scoring file pairs for each file in folder 1 in the database.
- Average score of the highest scoring file pairs for each file in folder 1 in the database.
- Standard deviation of all of the highest scoring file pairs for each file in folder 1 in the database.

Below is a screen shot of the database statistics form. Simply open a database and press the Calculate button.



The screenshot shows a window titled "Database Statistics" with a search icon and standard window controls. It contains two sections: "Overall" and "Highest". Each section has four input fields for "Minimum score", "Maximum score", "Average score", and "Standard deviation". At the bottom are "Calculate" and "Cancel" buttons.

Category	Minimum score	Maximum score	Average score	Standard deviation
Overall	3	89	25.85	18.33
Highest	6	89	32.52	19.9

SourceDetective

Running SourceDetective

SourceDetective considers each statement, comment, and identifier in the database and searches the Internet for references to each one. The number of times a statement, comment, or identifier is found in the search is then inserted into a new copy of the database, leaving the original database intact. Spreadsheets can then be generated to show the number of "hits" for each element. For more information, see the section entitled Creating Search Spreadsheets. Databases can be filtered to remove elements that have large hit numbers, meaning they are commonly found in other programs or documents. For more information, see the section entitled Using Filters.



Step 1

Choose options for searching the Internet.

- **Search for statements.** Choose this option to search the Internet for all statements in the current database. Note that when running SourceDetective on a CodeDiff database, this is the only option that produces results.
- **Search for comments/strings.** Choose this option to search the Internet for all comments and strings in the current database.
- **Search for identifiers.** Choose this option to search the Internet for all identifiers in the current database.
- **Include URLs.** Choose this option to include in the database URLs where the code elements were found. When you select this option, you will be able to select a number for the URLs threshold, which is the maximum number of URLs to report for each code element.

Step 2

Press the go button to begin the search. One search will be performed for each option selected and the number of "hits" will be inserted into the current database.

Exporting Databases

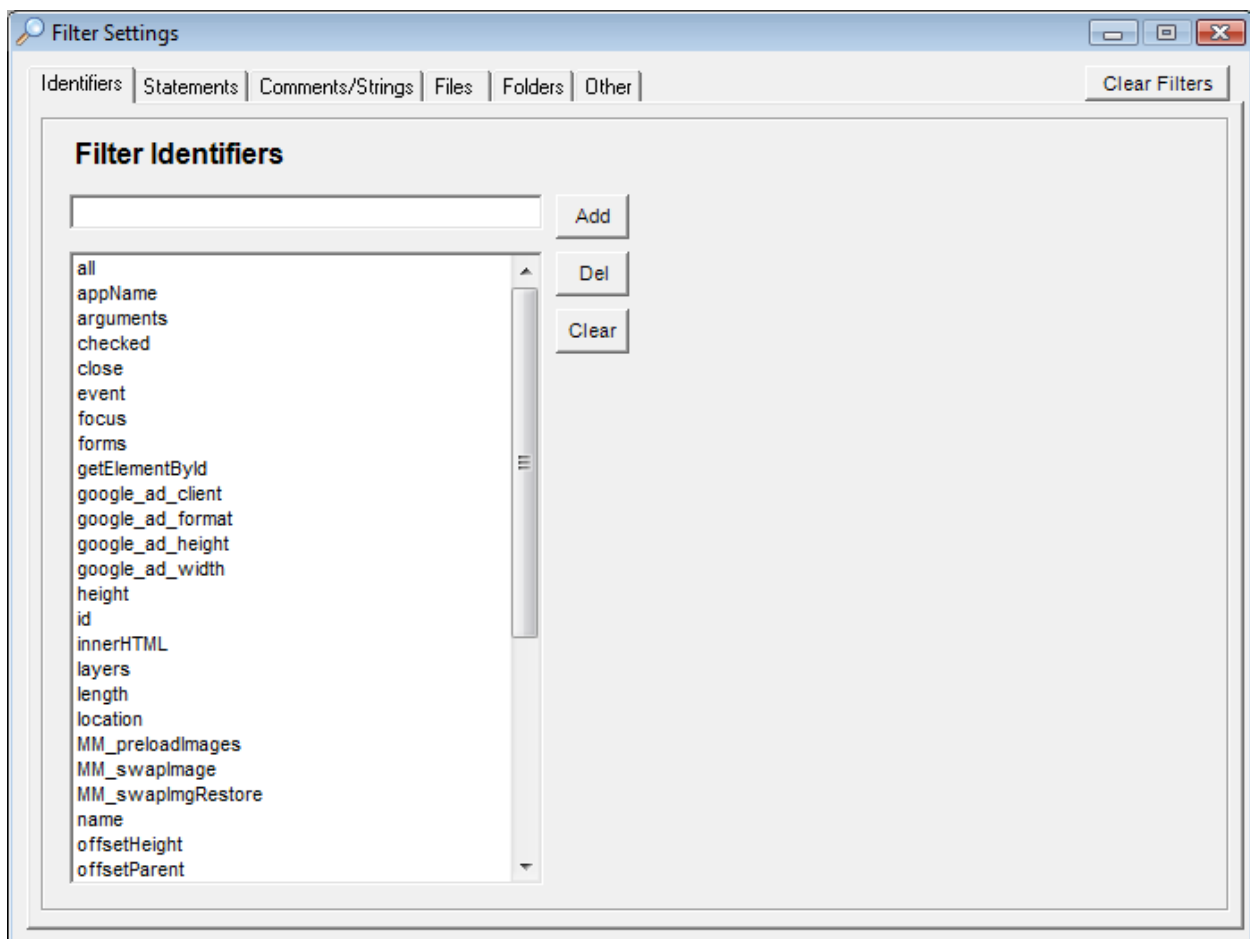
Filters

Filters are used to filter information out of a CodeSuite database that is not relevant to the analysis. For example, you may realize that certain files did not need to be included in the analysis and so you can filter them out. Or you may realize that certain identifier names or comments are very common, and they can be ignored in the analysis. Filtering allows you to remove these elements from the analysis without needing to run the program again. When filtering a database, a new filtered database is created, leaving the original database intact.

The filter form has several tabbed forms that are shown and described below.

Identifier Filters

The identifier filter form is shown below.



The identifier filter is used to filter out specific identifiers from a CodeSuite database. Enter a specific identifier in the field at the top and click on the Add button to add the identifier to the list of identifiers to be filtered. Select one or several identifiers from the list and click on the Del button to remove the identifiers from the list. Click on the Clear button to clear the list. When the database is filtered, the identifiers on the list will be removed from the database and the file pair scores will be adjusted accordingly.

Note that identifiers are only produced in databases generated by BitMatch or CodeMatch but not those generated by CodeDiff, so identifier filtering will have no effect on a CodeDiff generated database.

Statement Filters

The statement filter form is shown below.

Filter Settings

Identifiers **Statements** Comments/Strings Files Folders Other

Clear Filters

Filter Statements

Add
 Del
 Clear

```

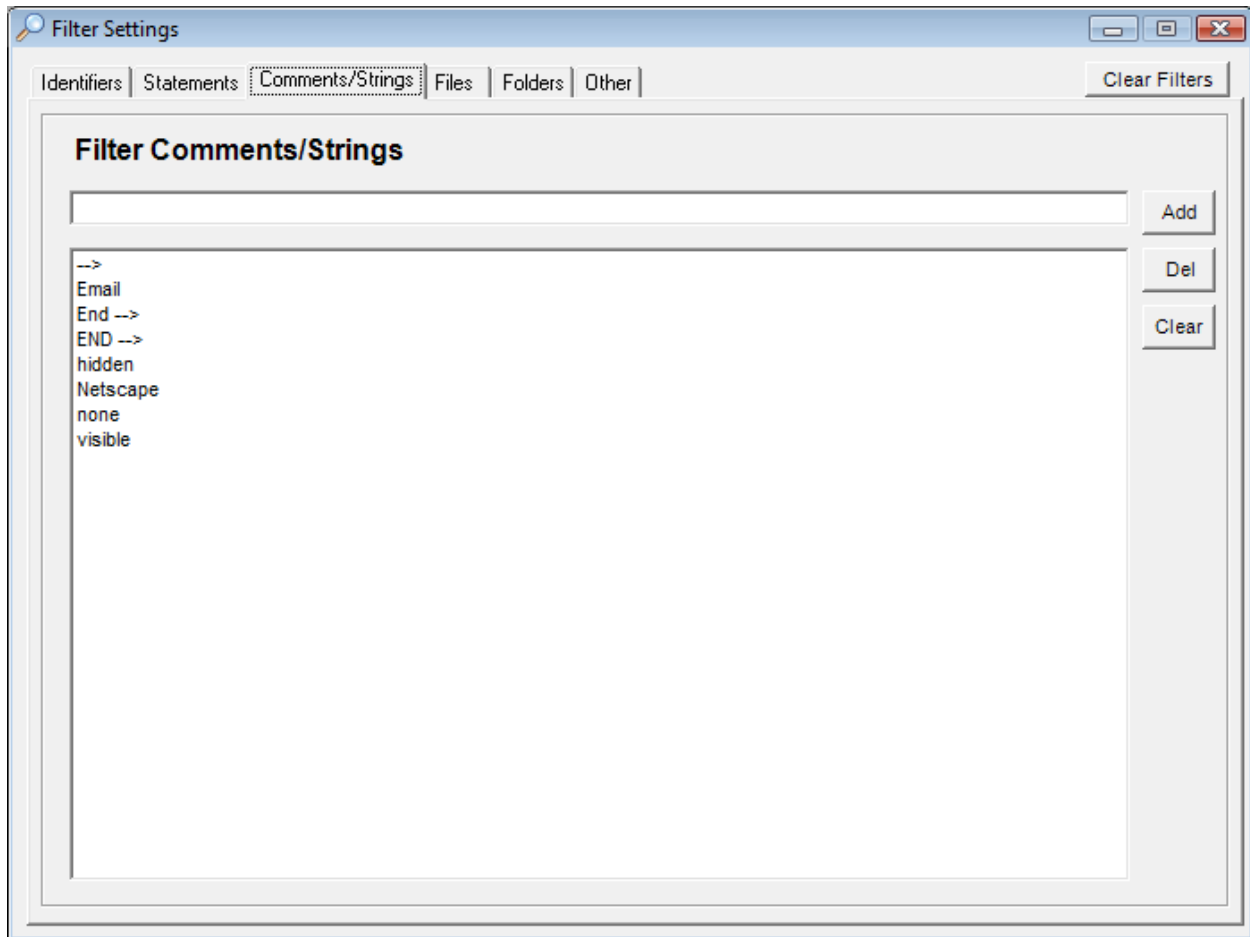
<!-- Begin
<!-- BEGIN
document.write( )
for (i = 0
function MM_preloadImages() {
function MM_swapImage() {
function MM_swapImageRestore() {
google_ad_client =
google_ad_format =
google_ad_height = 90
google_ad_width = 728
i++) {
i=0
window.close()
  
```

The statement filter is used to filter out specific statements from a CodeSuite database. Enter a specific statement in the field at the top and click on the Add button to add the statement to the list of statement to be filtered. Select one or several statements from the list and click on the Del button to remove the statements from the list. Click on the

Clear button to clear the list. When the database is filtered, the statements in the list will be removed from the database and the file pair scores will be adjusted accordingly. Statement filtering always ignores whitespace.

Comment/String Filters

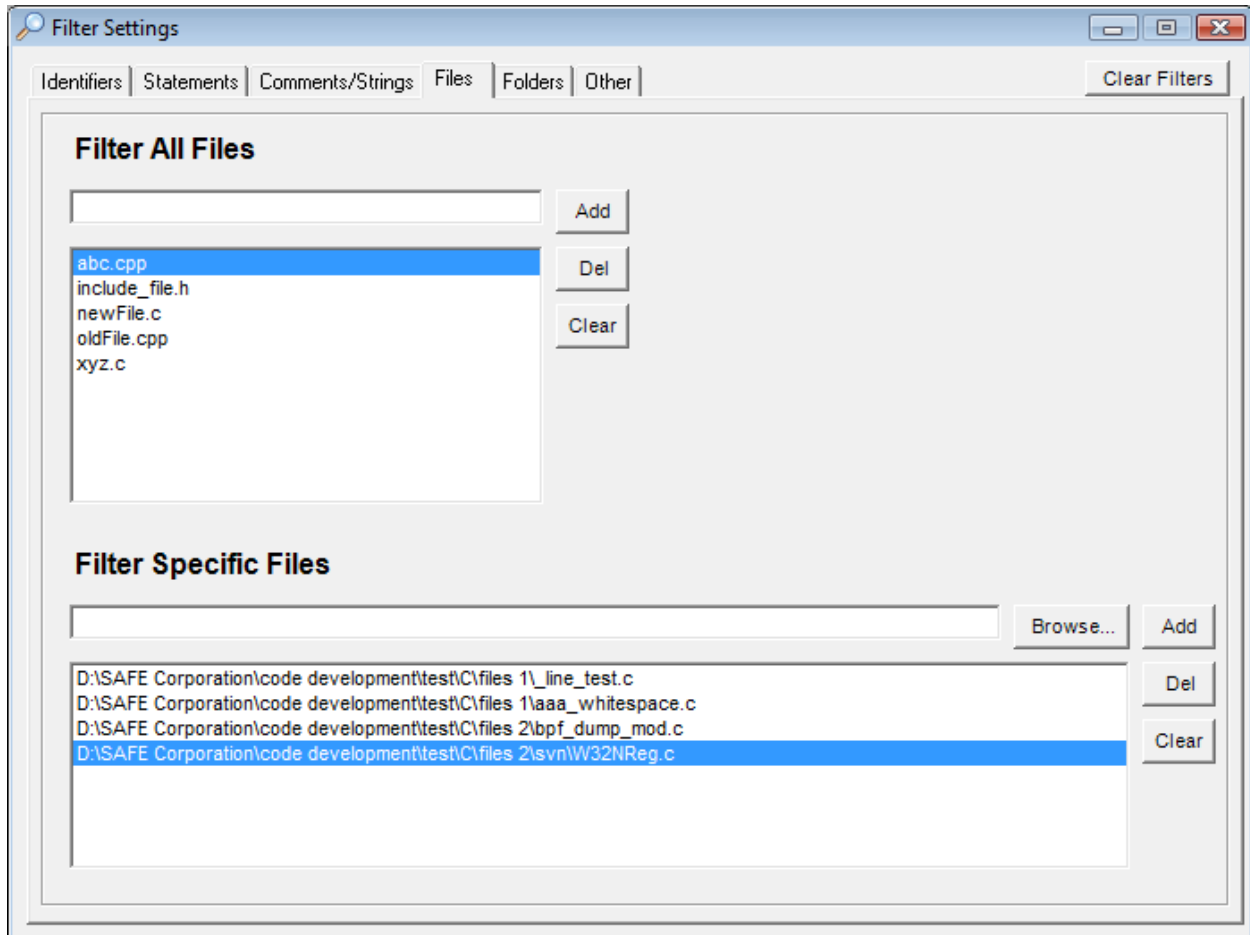
The comment/string filter form is shown below.



The comment/string filter is used to filter out specific comments and strings from a CodeSuite database. Enter a specific comment or string in the field at the top and click on the Add button to add it to the list to be filtered. Select one or several items from the list and click on the Del button to remove them from the list. Click on the Clear button to clear the list. When the database is filtered, the comments and strings in the list will be removed from the database and the file pair scores will be adjusted accordingly. Comment/string filtering always ignores whitespace.

File Filters

The file filter form is shown below.



Filter All Files

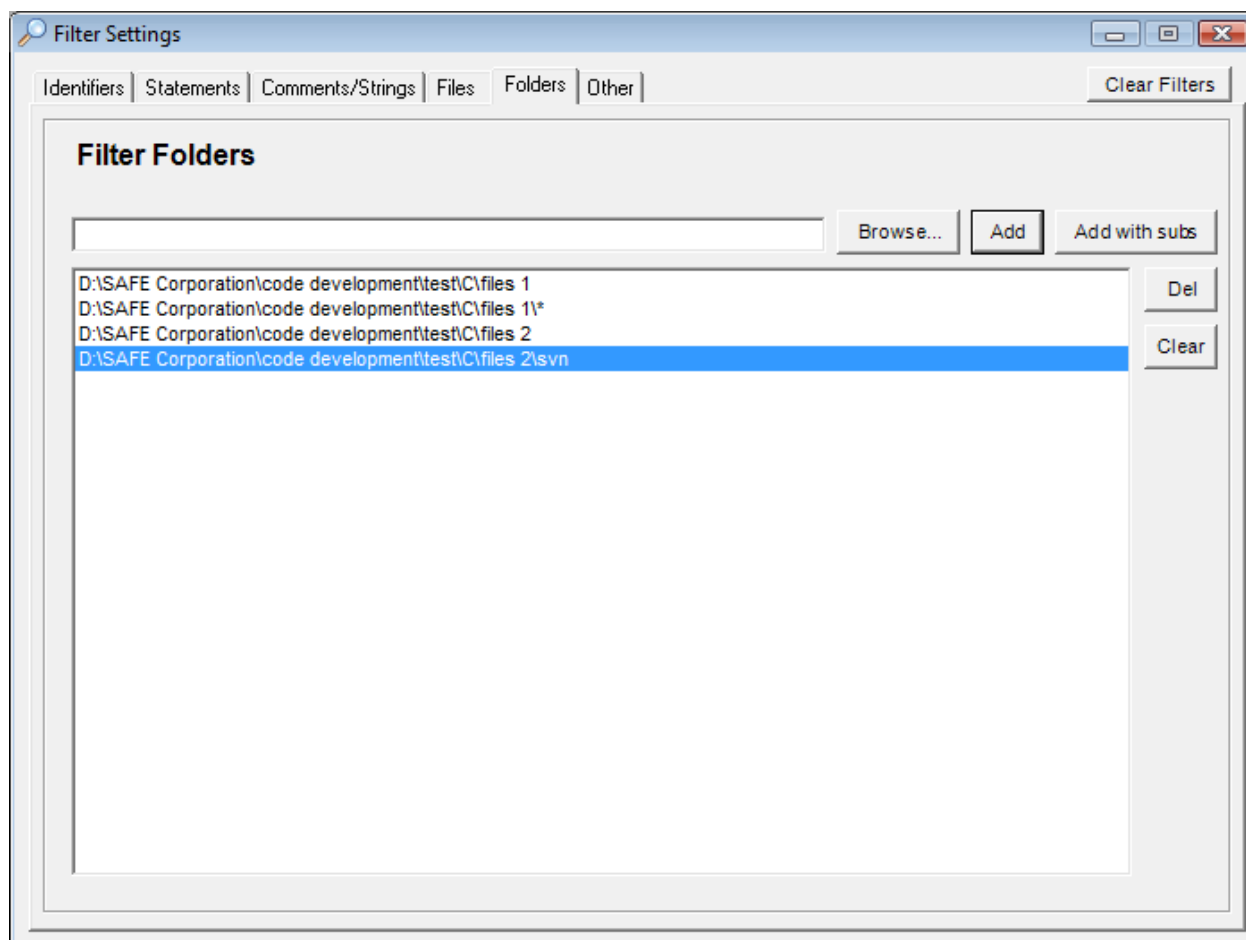
Filtering all files is used to filter out file names from a CodeSuite database. Enter a specific file name in the field at the top and click on the Add button to add the file to the list of files to be filtered. Select one or several files from the list and click on the Del button to remove the files from the list. Click on the Clear button to clear the list. When the database is filtered, the files in the list will be removed from the database, as if those files had never been compared. Note that you can use the * and ? wildcard characters.

Filter Specific Files

Filtering specific files is used to filter out specific files in specific paths from a CodeSuite database. Enter a specific file name and file path in the field at the top or click on the Browse button to select a specific file name and file path to be added to the filter. Click on the Add button to add the file to the list of files to be filtered. Select one or several files from the list and click on the Del button to remove the files from the list. Click on the Clear button to clear the list. When the database is filtered, the specific files in the list will be removed from the database, as if those files had never been compared.

Folder Filters

The folder filter form is shown below.



The folder filter is used to filter out specific folders from a CodeSuite database. Enter a specific folder in the field at the top or click on the Browse button to select a specific folder to be added to the filter. Click on the Add button to select the folder to be added to the list of folders to filter. Click on the Add with subs button to select a folder and all subfolders (specified by the trailing *) to be added to the list of folders to filter. Select one or several folders from the list and click on the Del button to remove the folders from the list. Click on the Clear button to clear the list.

Other Filters

The other filter form is shown below.

Filter Settings

Identifiers | Statements | Comments/Strings | **Files** | Folders | Other

Clear Filters

Filter Thresholds

Minimum Score: Maximum Files:

Maximum Score:

Filter Hits

Minimum Hits:

Maximum Hits:

Filter Statements ☐ Yes ☒ No

Filter Comments/Strings ☒ Yes ☐ No

Filter Sequences ☐ Yes ☒ No

Filter Identifiers ☒ Yes ☐ No

Filter Partial Identifiers ☒ Yes ☐ No

Filter Thresholds

The other filter form allows a reduction of the number of file pairs in the database. When the database is filtered, all file pairs that fall outside the maximum and minimum scores will be removed from the database. Also, only the maximum number of file pairs with the highest scores will remain in the database. Leaving a field blank will result in no threshold filter.

Filter Hits

The other filter form allows elements to be filtered out based on the number of hits found on the Internet using SourceDetective. When the database is filtered, all matching elements with hit values that fall outside the maximum and minimum hit values will be removed from the database and the scores will be modified appropriately. Leaving a field blank will result in no hit filter.

Filter Statements

The other filter form allows statements to be filtered out of the database. Selecting yes will eliminate statements from the database and modify the scores appropriately. For CodeDiff and CodeCLOC, this will filter out lines in the database.

Filter Comments/Strings

The other filter form allows comments and strings to be filtered out of the database. Selecting yes will eliminate comments and strings from the database and modify the scores appropriately.

Filter Sequences

The other filter form allows instruction sequences to be filtered out of the database. Selecting yes will eliminate instruction sequence from the database and modify the scores appropriately.

Filter Identifiers

The other filter form allows identifiers to be filtered out of the database. Selecting yes will eliminate identifiers from the database and modify the scores appropriately.

Filter Partial Identifiers

The other filter form allows all partial identifiers to be filtered out of the database. Selecting yes will eliminate all partial identifiers from the database and modify the scores appropriately.

Clear Filters

Pressing this button clears all filters on each filter form.

Sorting Databases

A CodeSuite database can be automatically sorted so that the files pairs with the highest scores from among all file pairs that were analyzed will appear at the top of the database file. This also means that the HTML reports generated from these databases will be sorted, making it easier to start examining file pairs starting with those that have the highest overall scores.

HTML Reports

A CodeSuite database can be automatically turned into HTML reports from the File menu for easy reading and presentation of results.

A basic report is generated that shows file pairs and their scores. By clicking on the score, a detailed HTML report will come up. These detailed reports are kept in subfolders. The detailed reports give more information about how the score was determined, showing specific similarities or differences between the files. The file names are given at the top of the report and include a hyperlink that, when clicked, allows the file to be brought up in a viewer or editor. The back and next buttons on the detailed reports allow you to navigate the detailed reports without going back to the basic report.

For information on BitMatch reports, see the sections entitled BitMatch Basic Report and BitMatch Detailed Report.

For information on CodeCLOC reports, see the sections entitled CodeCLOC Basic Report and CodeCLOC Detailed Report.

For information on CodeCross reports, see the sections entitled CodeCross Basic Report and CodeCross Detailed Report.

For information on CodeDiff reports, see the sections entitled CodeDiff Basic Report and CodeDiff Detailed Report.

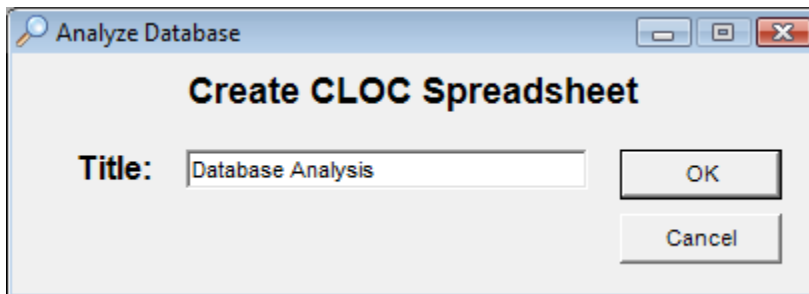
For information on CodeMatch reports, see the sections entitled CodeMatch Basic Report and CodeMatch Detailed Report.

For information on CodeSplit reports, see the sections entitled CodeSplit Basic Report and CodeSplit Detailed Report.

CLOC Spreadsheets

A CLOC spreadsheet shows the Changing Lines Of Code (CLOC) measurement produced by CodeCLOC) among the two sets of files belonging to two versions of a program. A CodeCLOC comparison is essentially a CodeDiff comparison of files with the same name such that each file is used at most once and the scores are optimized such that the overall percentage of similarity between the two sets of files is maximized. Note that perfect optimization would require compute time that grows unacceptably long for even small groups of files, so a local optimization is performed in a reasonable amount of time. In some cases, CLOC measurements on the same sets of files may result in very slightly different numbers.

Below is the form you use to create a CLOC spreadsheet from the File menu.



The screenshot shows a Windows-style dialog box titled "Analyze Database". Inside the dialog, the main heading is "Create CLOC Spreadsheet". Below this, there is a label "Title:" followed by a text input field containing the text "Database Analysis". To the right of the input field are two buttons: "OK" and "Cancel".

The title that you enter will appear at the top of spreadsheet. Press the OK button and you will be asked to name the spreadsheet file.

Example CLOC Spreadsheet

Below is an example of a spreadsheet created from a CodeCLOC database.

	A	B
1	CodeCLOC CLOC Spreadsheet	
2		
3	Database:	C:\SAFE Corporation\code development\test\C\results\CodeCLOC.cdb
4	Database Analysis	
5	Analysis date:	4/27/2010
6		
7	Version 1 directory:	C:\SAFE Corporation\code development\test\C\files1
8		including subdirectories
9	Version 2 directory:	C:\SAFE Corporation\code development\test\C\files2
10		

11		
12	File types:	*.cpp;*.c;*.h
13		
14	Total files in first set [TF0]	150
15	Total lines of code in the first set [LOC0]	64829
16	Total bytes in the first set [TB0]	2246044
17	Total files in the second set [TF1]	809
18	Total lines of code in the second set [LOC1]	291038
19	Total bytes in the second set [TB1]	10741835
20		
21	Change Summary	
22	Files	
23	New files [NF]	739
24	Modified continuing files [MCF]	70
25	Lines	
26	New and modified lines of code [CLOC]	275635
27		
28	Continuation Summary	
29	Files	
30	Continuing files [CF]	70
31	Unchanged continuing files [UCF]	0
32	Lines	
33	LOC in continuing files [LOCinCF]	49313
34	CLOC in continuing files [CLOCinCF]	33910
35	Unchanged continuing lines of code [ULOC]	15403
36		
37	Summary Statistics	
38	Files	
39	Percent of new and modified files $((NF+MCF)/TF1)$	100
40	Percent of new and modified files relative to base version $((NF+MCF)/TF0)$	539
41	Percent of continuing files $(CF/TF1)$	9
42	Percent of unchanged continuing files $(UCF/TF1)$	0
43	Lines	
44	Percent CLOC change $(CLOC/LOC1)$	95
45	Percent CLOC change relative to base version $(CLOC/LOC0)$	425
46	Percent LOC growth $(LOC/LOC0)$	449
47	Percent unchanged continuing lines of code $(ULOC/LOC1)$	5

The top line shows that the spreadsheet was created from a CodeSuite database generated by CodeCLOC. The third line gives the name of the database file. The fourth line is the title that the user placed in the spreadsheet form shown above. The analysis date shows the date that the spreadsheet was created. The folders are the ones that were compared to create the database including subdirectories if noted.

The quantities in the spreadsheet are described below. The two versions being compared consist of the initial *base* version of code and a subsequent *examined* version that is being compared to it.

- **Total files [TF0]:** The total number of files in the base version.
- **Total lines of code [LOC0]:** The total lines of code (blank lines ignored) in the base version.
- **Total bytes [TB0]:** The total bytes of code in the base version.
- **Total files [TF1]:** The total number of files in the examined version.
- **Total lines of code [LOC1]:** The total lines of code (blank lines ignored) in the examined version.
- **Total bytes [TB1]:** The total bytes of code in the examined version.
- **New files [NF]:** The number of files that exist in the examined version but did not exist in the base version.
- **Modified continuing files [MCF]:** The number of continuing files (files that exist in both the examined version and the base version) with new lines of code.
- **New and modified lines of code [CLOC]:** The number of lines of code in the examined version that are new or have been modified from the base version.
- **Continuing files [CF]:** The number of files in the examined version that were also in the base version.
- **Unchanged continuing files [UCF]:** The number of continuing files that have not changed from the base version to the examined version.
- **Lines of code in continuing files [LOCinCF]:** The number of lines of code in the continuing files.
- **New and modified lines of code in continuing files [CLOCinCF]:** The number of lines of code in the continuing files that are new or have been modified from the base version.
- **Unchanged continuing lines of code [ULOC]:** The number of lines of code in continuing files that have not changed.
- **Percent of new and modified files:** $((NF+MCF)/TF1)$
- **Percent of new and modified files relative to base version:** $((NF+MCF)/TF0)$

- **Percent of continuing files:** $(CF/TF1)$
- **Percent of unchanged continuing files:** $(UCF/TF1)$
- **Percent CLOC change:** $(CLOC/LOC1)$
- **Percent CLOC change relative to base version:** $(CLOC/LOC0)$
- **Percent LOC growth:** $(LOC1/LOC0)$
- **Percent unchanged continuing lines of code:** $(ULOC/LOC1)$

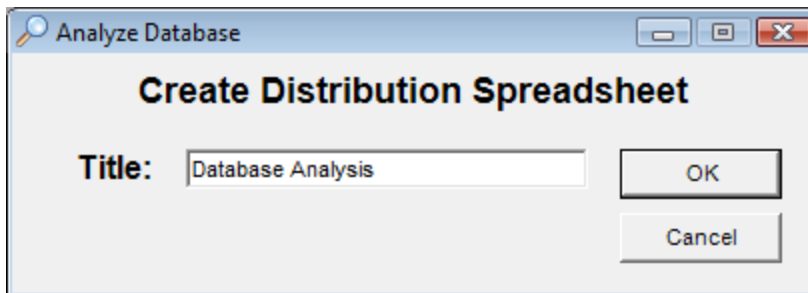
Distribution Spreadsheets

A distribution spreadsheet shows the distribution of scores (typically similarity scores from CodeDiff) among the two sets of files compared. It is important to note that for each file in the first set of compared files, a distribution spreadsheet only considers the similarity scores for the most similar file in the second set of files compared. So for each file in the first set of files, the spreadsheet will only consider the most similar file in the second set, even other less similar files are noted in the database. Also note that while each file in the first set of files is considered exactly once, some files in the second set of files may be considered multiple times or not at all.

Consider set A consisting of files A1 and A2. Consider set B consisting of files B1 and B2. Now consider the table below of similarity scores. The file with the most similarity to file A1 is file B1 and the file with the most similarity to file A2 is also file B1. So file B1 will be counted twice in the distribution spreadsheet.

Set A	Set B	Similarity Score
File A1	File B1	90
File A1	File B2	20
File A2	File B1	67
File A2	File B2	12

Below is the form you use to create a distribution spreadsheet from the File menu.



The screenshot shows a dialog box titled 'Analyze Database'. Inside, there is a section titled 'Create Distribution Spreadsheet'. Below this, there is a 'Title:' label followed by a text input field containing 'Database Analysis'. To the right of the input field are two buttons: 'OK' and 'Cancel'.

The title that you enter will appear at the top of spreadsheet. Press the OK button and you will be asked to name the spreadsheet file.

Example Distribution Spreadsheet

Below is an example of a spreadsheet created from a CodeDiff database.

	A	B	C	D	E	F
1	CodeDiff Results Distribution					
2	Database Analysis					

CodeSuite User's Guide

3	Database	C:\SAFE Corporation\code development\test\C\results\CodeMatch1.cdb				
4	Run date	12/1/2007				
5	Analysis date	12/2/2007				
6						
7	Folder 1	C:\SAFE Corporation\code development\test\C\files1				
8		including subdirectories				
9	Folder 2	C:\SAFE Corporation\code development\test\C\files2				
10						
11						
12	Algorithm	Ignoring case				
13	Algorithm	Ignoring whitespace				
14	Algorithm	Percentage of file pairs				
15	File threshold	8				
16	Score threshold	1				
17						
18	Total files in folder 1	16				
19						
20	File pair comparisons					
21			Folder 1		Folder 2	
22	Match percentage	Number of files	Number of lines	Number of bytes	Number of lines	Number of bytes
23	20	1	5	252	5	230
24	28	1	291	7725	227	5010
25	75	2	8	224	8	228
26	99	1	56	1997	52	1890
27	100	11	960	27288	960	27282
28						
29	Totals	16	1320	37486	1252	34640
30	Total changed	5	216		170	
31	Percent changed	31.25	16.37		13.58	

The top line shows that the spreadsheet was created from a CodeSuite database generated by CodeDiff. The second line is the title that the user placed in the spreadsheet form shown above. The run date shows the date that CodeDiff was run while the analysis date shows the date that the spreadsheet was created. The folders are the ones that were compared to create the database.

The algorithms show all algorithms that were selected for the CodeDiff run. In this case, letter case and whitespace were ignored, and the percentage given is the percentage of lines in the first file that were matched in the second file. The file threshold shows that

the top 8 files were reported. The score threshold shows that files needed at least 1% similarity to be reported. Note that the database may have been subsequently filtered and that this fact is not reflected in the spreadsheet.

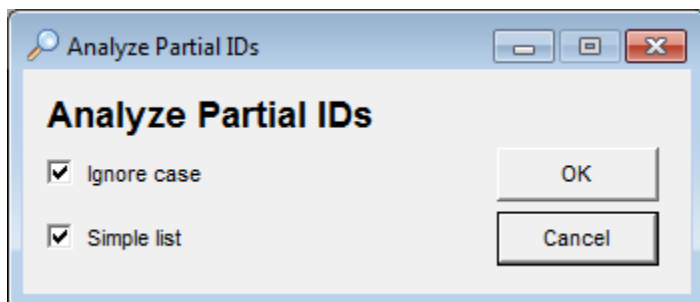
The total number of files in the first folder (including all subfolders if that was selected for the CodeDiff run) is shown as 16.

Rows 21 and 22 are the table header. Column A shows a percentage score while column B shows the total number of files. Columns C and D show the total number of lines and bytes in files in folder 1 that had this particular score. Columns E and F show the total number of lines and bytes in files in folder 2 that had this particular score. Note that only the highest percentage score is considered for this analysis. In other words, if file A in folder 1 is matched 97% by file X in folder 2, 93% by file Y in folder 2, and 37% by file Z in folder 2, only file X in folder 2 is considered in the analysis.

Creating PID Spreadsheets

A PID spreadsheet gives a list of all of the partially matching identifiers from a CodeMatch comparison in the currently selected CodeMatch database.

Below is the form you use to create a PID spreadsheet from the File menu.



Check the Ignore case check box to ignore the case of the partial IDs.

Check the Simple list check box to produce a spreadsheet that only shows the partially matching identifiers. Uncheck the check box to produce a spreadsheet that shows the partially matching identifiers and the original identifiers that partially matched. This will be a much longer spreadsheet.

Example Simple PID Spreadsheet

Below is an example of a simple PID spreadsheet created from a CodeMatch database.

	A	B	C
1	CodeMatch Partial Identifiers		
2	Ignore Case		
3	Simple List		
4	Database	Z:\SAFE Corporation\code development\test\C\results\CodeMatch1.cdb	
5	Run date	3/11/2009	
6	Analysis date	12/16/2012	
7			
8	Folder 1	C:\SAFE Corporation\code development\CodeSuite\test\C\files 1	
9			
10	Folder 2	C:\SAFE Corporation\code development\CodeSuite\test\C\files 2	
11			

12			
13	Partial Identifier		
14	bpf_		
15	bpf_d		
16	bpf_i		
17	bpf_im		
18	f_i		
19	f_in		
20	f_len		
21	ins		

The top line shows that the spreadsheet was created from a CodeSuite database generated by CodeMatch. The second line shows that letter case was ignored. The third line indicates that a simple list of partial identifiers was specified. The fourth line is the path to the database that was analyzed. Line 5 shows the run date that CodeMatch was run while the analysis date on line 6 shows the date that the spreadsheet was created. The folders on lines 8 and 9 are the ones that were compared to create the database.

Rows 14 through 21 are the partial identifiers that were specified in the database. In other words, for all identifiers that could did not have complete matches in both sets of code, these are the parts of identifiers that could be found within the identifiers in both sets of code.

Example Non-simple PID Spreadsheet

Below is an example of a non-simple PID spreadsheet created from a CodeMatch database.

	A	B	C
1	CodeMatch Partial Identifiers		
2	Ignore Case		
3	Simple List		
4	Database	Z:\SAFE Corporation\code development\test\C\results\CodeMatch1.cdb	
5	Run date	3/11/2009	
6	Analysis date	12/16/2012	
7			
8	Folder 1	C:\SAFE Corporation\code development\CodeSuite\test\C\files 1	
9			

10	Folder 2	C:\SAFE Corporation\code development\CodeSuite\test\C\files 2	
11			
12			
13	Partial Identifier	Identifier1	Identifier2
14	bpf_	bpf_a	bpf_dump
15	bpf_	bpf_a	bpf_image
16	bpf_	bpf_a	bpf_program
17	bpf_	bpf_div	bpf_image
18	bpf_	bpf_div	bpf_program
19	bpf_	bpf_imm	bpf_dump
20	bpf_	bpf_imm	bpf_program
21	bpf_	bpf_ind	bpf_dump
22	bpf_	bpf_ind	bpf_program
23	bpf_	bpf_len	bpf_dump
24	bpf_	bpf_len	bpf_image
25	bpf_	bpf_len	bpf_program
26	bpf_d	bpf_div	bpf_dump
27	bpf_i	bpf_ind	bpf_image
28	bpf_im	bpf_imm	bpf_image
29	f_i	bpf_imm	bf_insns
30	f_in	bpf_ind	bf_insns
31	f_len	bpf_len	bf_len
32	ins	winsock	bf_insns
33	ins	winsock	insn
34	ins	winsock	bf_insns
35	ins	winsock	insn

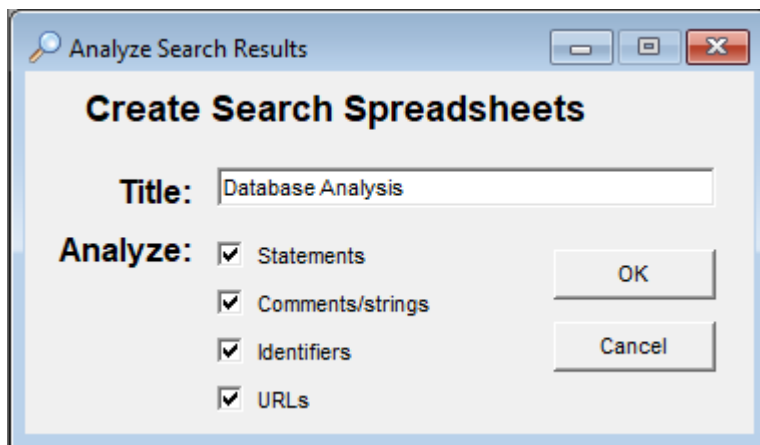
The top line shows that the spreadsheet was created from a CodeSuite database generated by CodeMatch. The second line shows that letter case was ignored. The third line indicates that a simple list of partial identifiers was specified. The fourth line is the path to the database that was analyzed. Line 5 shows the run date that CodeMatch was run while the analysis date on line 6 shows the date that the spreadsheet was created. The folders on lines 8 and 9 are the ones that were compared to create the database.

Column A of rows 14 through 35 shows the partial identifiers that were specified in the database. In other words, for all identifiers that could did not have complete matches in both sets of code, these are the parts of identifiers that could be found within the identifiers in both sets of code. Columns B and C show the complete identifier names that partially matched to form the associated partially matching identifier name.

Search Spreadsheets

A search spreadsheet shows the number of times a code element (i.e., statement, comment, string, or identifier) can be found when searching the Internet using SourceDetective.

Below is the form you use to create a search spreadsheet from the File menu. Before a search spreadsheet can be created, SourceDetective must be run on the database to search the Internet for code elements such as statements, comments, and identifiers. There is also an option to create a spreadsheet of URLs where all of the elements were found on the Internet.



The title that you enter will appear at the top of spreadsheet. Press the OK button and you will be asked to name the spreadsheet files.

Example Search Spreadsheet

Below is an example of a search spreadsheet created from a CodeMatch database.

	A	B
1	CodeMatch Internet Search Results	
2	Database Analysis	
3	Database	C:\SAFE\code development\test\C\results\CodeMatch1.cdb
4	Run date	12/1/2007
5	Analysis date	12/2/2007
6		
7	Folder 1	C:\SAFE Corporation\code development\test\C\files1
8		including subdirectories
9	Folder 2	C:\SAFE Corporation\code development\test\C\files2

10		
11		
12	Statements	Search Score
13	#include <Assert.h>	2147483647
14	#include <time.h>	2147483647
15	&hKeyAdapter	5
16	(void)sprintf(image	29
17	bpf_dump(struct bpf_program *p, int option)	37
18	case BPF_ALU BPF_LSH BPF_K:	100
19	case BPF_LD BPF_B BPF_ABS:	155
20	char xchar	398
21	hKeyClassNet,	1
22	if(nPlatformVersion == 0x0004)	0
23	int number222	0
24	NULL,	226000000
25	switch (p->code) {	57
26	xyz abc	110000

The top line shows that the spreadsheet was created from a CodeSuite database generated by CodeMatch. The second line is the title that the user placed in the spreadsheet form shown above. The run date shows the date that CodeMatch was run while the analysis date shows the date that the spreadsheet was created. The folders are the ones that were compared to create the database.

The next line shows that this is a listing of statements. The left column lists statements in alphabetical order. The right column gives the number of hits on the search engine for this statement.

Example URL Spreadsheet

Below is an example of a URL spreadsheet created from a CodeMatch database.

	A		B
1	CodeMatch Internet Search Results		
2	Database Analysis		
3	Database	C:\SAFE\code development\test\C\results\CodeMatch1.cdb	
4	Run date	12/1/2007	
5	Analysis date	12/2/2007	
6			

7	Folder 1	C:\SAFE Corporation\code development\test\C\files1	
8		including subdirectories	
9			
10	Website	URL	Instances
11	01-17-12MAC-Agenda	http://alamore.org/alamomacfolder/01-17-12MAC-Agenda.pdf	37
12	111 and other triple numbers Explained	http://angelscribe.com/1111.html	62914
13	Aspen Elementary	http://aes.uinta1.com/	557
14	AUTOLAB: UC Berkeley Laboratory for Automation Science and Engineering	http://autolab.berkeley.edu/	290
15	AVR Assembler User Guide	http://academy.cba.mit.edu/classes/embedded_programming/doc1022.pdf	317
16	Biro Pengawasan Farmaseutikal Kebangsaan	http://acronyms.thefreedictionary.com/Biro+Pengawasan+Farmaseutikal+Kebangsaan	100
17	BPF syntax	http://biot.com/capstats/bpf.html	155
18	SAFE Corporation - Software Analysis and Forensic Engineering	http://www.SAFE-corp.com	398
19	The Meaning of the Number 1111	http://awakenlight.org/the-meaning-of-the-number-1111	1
20	What is BPF and why is it taking over Linux Performance Analysis?	http://blog.memsql.com/bpf-linux-performance/	64

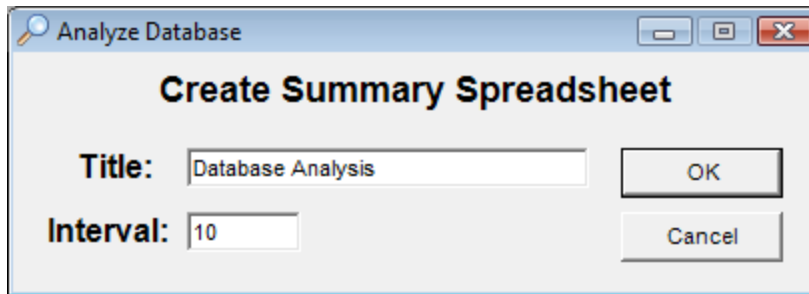
The top line shows that the spreadsheet was created from a CodeSuite database generated by CodeMatch. The second line is the title that the user placed in the spreadsheet form shown above. The run date shows the date that CodeMatch was run while the analysis date shows the date that the spreadsheet was created. The folder is the ones containing source code files that were split to create the database.

The next line shows that this is a listing of URLs. The left column lists website names in alphabetical order. The middle column lists the URLs for those websites. The right

column gives the number of times that an element in the database could be found at that URL.

Summary Spreadsheet

Below is the form you use to create a summary spreadsheet from the File menu.



Analyze Database

Create Summary Spreadsheet

Title: Database Analysis OK

Interval: 10 Cancel

The title that you enter will appear at the top of spreadsheet. The interval is used to divide up file pair scores for the spreadsheet. Press the OK button and you will be asked to name the spreadsheet file.

Example CodeMatch Spreadsheet

Below is an example of a spreadsheet created from a CodeMatch database.

	A	B	C	D	E	F	G	H
1	CodeMatch Results Summary							
2	Database Analysis							
3	Run date	7/4/2006						
4	Analysis date	8/1/2006						
5								
6	Algorithm	Statement Matching						
7	Algorithm	Comment/String Matching						
8	Algorithm	Identifier Matching						
9	Algorithm	Instruction Sequence Matching						
10	File threshold	4						
11								
12	Total files in folder 1	22						
13	Total file pairs	83						
14								
15	Scores		(0-9)	(10-19)	(20-29)	(30-39)	(40-49)	(50-59)
16	Numbers		17	15	26	21	0	4
17	Percentage		20	18	31	25	0	5

The top line shows that the spreadsheet was created from a CodeSuite database generated by CodeMatch. The second line is the title that the user placed in the spreadsheet form shown above. The run date shows the date that CodeMatch was run while the analysis date shows the date that the spreadsheet was created.

The algorithms show all algorithms that were selected for the CodeMatch run. In this case, the statement matching, comment matching, identifier matching, and instruction

sequence matching algorithms were all selected. The threshold shows that the top four files that were correlated to the files in folder 1 were reported. Note that the database may have been subsequently filtered and that this fact is not reflected in the spreadsheet.

The total number of files in the first folder (including all subfolders if that was selected for the CodeMatch run) is shown as 22. The total number of files pairs that are reported in the database is 83.

Row 15 is the table header. Columns C through H show intervals of match scores (correlation scores). For example, column C represents match scores between 0 and 9 inclusively.

Row 16 shows the number of file pairs in each match score interval while row 17 shows the percentage of file pairs in each match score interval.

Creating XML Databases

A CodeSuite database can be automatically turned into an XML file from the File menu for analysis by off-the-shelf XML analysis tools. For detailed information on the specification of a CodeSuite XML file, see the section on XML file format.

Languages

Languages Supported

The following programming languages are currently supported:

ABAP	ASM-6502	ASM-65C02	ASM-65816	ASM-C55x
ASM-C67x	ASM-M68k	BASIC	C	C++
C#	COBOL	D	Delphi	DRI ASM
Flash ActionScript	Fortran	FoxPro	Go	Java
JavaScript	Kotlin	LISP	LotusScript	Lua
MASM	MATLAB	MPE/iX	Objective-C	OpenEdge
Pascal	Perl	PHP	PL/M	PL/SQL
PowerBuilder	PowerHouse	PowerShell	Progress	Prolog
Python	RealBasic	Ruby	Scala	SQL
Structured Text	Swift	TAL	TCL	TypeScript
Verilog	VHDL	Visual Basic		

Check the SAFE Corporation website for new language modules, available at no charge, as they become available. If the language you need is not available, contact SAFE Corporation about creating it for a nominal fee.

Advanced Topics

CodeSuite Database Format

SAFE Corporation wants third party developers to create software to analyze CodeSuite database files and generate reports and statistics from them. In light of that desire, below is an example of a CodeSuite database file with all sections explained by a comment. Comments are lines beginning with a # symbol.

Tag Definitions

The CodeSuite database tags and their meanings are given in the table below.

Tag	Definition
#	First character in a comment
<Program>	Program name (e.g., CodeMatch or CodeCLOC)
<Version>	Program version
<Date>	Date file was created
<Time>	Time file was created
<LComment>	Left-justified comment to be placed in the output file
<Describe>	Insert program description
<CComment>	Right-justified comment to be placed in the output file
<Filters>	Beginning of filters used to create this database (see Filters section)
</Filters>	End of filters used to create this database
<Folder1>	Input folder 1
<Subs1>	Compare subdirectories for folder 1? T or F
<Folder2>	Input folder 2
<Subs2>	Compare subdirectories for folder 2? T or F
<FolderT>	Templates folder
<SubsT>	Include subdirectories of templates folder? T or F
<Language>	Programming language
<Encoding1>	Binary encoding for folder 1
<Encoding2>	Binary encoding for folder 2
<FileType>	Filetype
<SameName>	Compare files of same name only? T or F
<Dirs>	Compare directories? T or F
<URLThresh>	URL threshold
<Algorithm>	Algorithm name
<FileThresh>	File number threshold to report
<ScoreThresh>	Score threshold to report
<Filter>	Filter name
<Dir1>	Input file 1 path relative to InFolder1
<Dir2>	Input file 2 path relative to InFolder2
<File1>	Input file 1 name
<NumWords1>	Number of words in file 1 (for DocMate)
<NumLines1>	Number of lines in file 1
<Size1>	Number of bytes in file 1
<File2>	Input file 2 name

<NumWords2>	Number of words in file 2 (for DocMate)
<NumLines2>	Number of lines in file 2
<Size2>	Number of bytes in file 2
<StatementScore>	Statement match score
<CommentScore>	Comment match score
<SequenceScore>	Sequence match score
<IdentifierScore>	Identifier match score
<SubIdentifierScore>	Partial identifier match score
<Score>	Match score
<NoScore>	File was not compared -- no score
<Differences>	Beginning of line differences
</Differences>	End of line differences
<Statements>	Beginning of statement comparison
</Statements>	End of statements comparison
<Comments>	Beginning of comment comparison
</Comments>	End of comment comparison
<Lines1>	Lines in file 1
<Lines2>	Lines in file 2
<Line>	Instruction/comment line
<Sequences>	Beginning of instruction sequences
<InSeq>	Instruction sequence
<Instr>	Instruction in sequence
</Sequences>	End of instruction sequences
<IDs>	Beginning of identifiers
<ID>	Identifiers
</IDs>	End of identifiers
<PIDs>	Beginning of partial identifiers
<PID1>	Partial identifiers in file 1
<PID2>	Partial identifiers in file 2
</PIDs>	End of partial identifiers
<BingHits>	Number of hits on Bing search engine for preceding elements
<YahooHits>	Number of hits on Yahoo search engine for preceding elements
<URL>	URL where the preceding element can be found
<Folder1Bytes>	Total bytes in all files examined in folder 1
<Folder1Lines>	Total lines in all files examined in folder 1
<Folder1Files>	Total number of files examined in folder 1
<Folder2Bytes>	Total bytes in all files examined in folder 2
<Folder2Lines>	Total lines in all files examined in folder 2
<Folder2Files>	Total number of files examined in folder 2
<ExTime>	Time to execute program

Example CodeSuite database file

#Each line begins with a tag, ends with a newline
 #Comments begin with #, end with a newline

```
<Program>CodeMatch
<Version>version
<Date>date
<Time>time
```

```
<CComment>Centered comment
```

```

<LComment>Left justified comment

<Folder1>input_folder
<Subs1>T
<Folder2>compare_folder
<Subs2>F
<Language>programming_language
<CaseSensitive>T
<FileType>filetypes
<SameName>T
<Dirs>T
#Note: <Dirs> tag specifies that comparison is being done on a directory
basis
<Algorithm>algorithm_name
<Algorithm>algorithm_name
<Algorithm>algorithm_name
<FileThresh>number
<ScoreThresh>number

#The following filters (from the filter file) were used to create this
database
<Filters>
<Filter>filter
<Filter>filter
</Filters>

#For each file in the first set of files being compared:
<Dir1>input_file_1_path_relative_to_Folder1
<Dir2>input_file_2_path_relative_to_Folder2
<File1>filename
<NumLines1>number_of_lines
<Size1>size_in_bytes
<File2>filename
<NumLines2>number_of_lines
<Size2>size_in_bytes
#Note: <Dir1> and <Dir2> tags may or may not be repeated if they are
unchanged from subsequent files

#Show differences for CodeDiff
<Differences>
<Line>line
<BingHits>129375
<Lines1>line_number line_number
<Lines1>line_number line_number
<Line>line
<BingHits>673
<Lines2>line_number line_number line_number
</Differences>

<Statements>
<Line>instruction
<BingHits>75
<Lines1>line_number line_number
<Lines1>line_number line_number
<Lines2>line_number line_number line_number
<Lines2>line_number
<Line>instruction

```

CodeSuite User's Guide

```
<BingHits>12
<Lines1>line_number line_number
<Lines2>line_number line_number line_number
<Line>instruction
<BingHits>9348753
<Lines1>line_number line_number line_number
<Lines2>line_number line_number line_number
</Statements>
<StatementScore>statement_correlation_score
# Note: If there is no statement score
# there were no statements to compare or
# statements were not compared in this run.

<Comments>
<Line>comment
<BingHits>2341
<Lines1>line_number line_number line_number
<Lines2>line_number line_number line_number
<Line>comment
<BingHits>444
<Lines1>line_number line_number
<Lines2>line_number line_number line_number line_number line_number
<Lines2>line_number
</Comments>
<CommentScore>comment_correlation_score
# Note: If there is no comment score
# there were no comments to compare or
# comments were not compared in this run.

#Instruction listing using <Instr> tag is optional
<Sequences>
<InSeq>line_number line_number sequence_length_number
<Instr>Instruction 1
<Instr>Instruction 2
<Instr>Instruction 3
<Instr>Instruction n<InSeq>line_number line_number sequence_length_number
<InSeq>line_number line_number sequence_length_number
<InSeq>line_number line_number sequence_length_number
</Sequences>
<SequenceScore>sequence_correlation_score
# Note: If there is no sequence score
# there were no sequences to compare or
# sequences were not compared in this run.

<IDs>
<ID>identifier_string identifier_string identifier_string identifier_string
<BingHits>12 888 3 90023
<ID>identifier_string
<BingHits>129
</IDs>
<PIDs>
<PID1>partial_identifier_string partial_identifier_string
<PID1>partial_identifier_string partial_identifier_string
<PID1>partial_identifier_string
<PID2>partial_identifier_string partial_identifier_string
<PID2>partial_identifier_string partial_identifier_string
<PID2>partial_identifier_string
```

```

</PIDs>
<IdentifierScore>identifier_correlation_score
# Note: If there is no identifier score
# there were no identifiers to compare or
# identifier matching was not selected for this run.

<Score>file_pair_correlation_score

#For each uncomparing file (a file that was never compared to anything):
<Dir1>input_file_1_path_relative_to_Folder1
<File1>filename
<NumLines1>number_of_lines
<Size1>size_in_bytes
<NoScore>

#Summary
<Folder1Bytes>number
<Folder1Lines>number
<Folder1Files>number
<Folder2Bytes>number
<Folder2Lines>number
<Folder2Files>number
<ExTime>execution time

```

CodeSuite Filter Format

SAFE Corporation wants third party developers to create software to filter CodeSuite database files. In light of that desire, below is an example of a CodeSuite filter file with all sections explained by a comment. Comments are lines beginning with a # symbol.

Tag definitions

The CodeSuite filter tags and their meanings are given in the table below.

Tag	Definition
#	Filter file comment character
<Statement>	Statements to filter
<Comment>	Comments to filter
<Identifier>	Identifiers to filter
<File>	Files to filter
<Folder>	Folders to filter
<Threshold>	Filter threshold (possible parameters are given below)
MinScore	Minimum score threshold value
MaxScore	Maximum score threshold value
MaxFile	Maximum file number threshold value
<AllStatements>	Filter all statements
<AllComments>	Filter all comments
<AllSequences>	Filter all sequences
<AllIdentifiers>	Filter all identifiers
<PartialIDs>	Filter all partial identifiers

Example CodeSuite filter file

```
#Comments begin with #, end with a newline
#Note that all filters are case sensitive
```

```
#List of statements to filter out
<Statement>Statement1
<Statement>Statement2
<Statement>Statement3
```

```
#List of comments to filter out
<Comment>Comment1
<Comment>Comment2
<Comment>Comment3
<Comment>Comment4
```

```
#Filter out all sequences
<AllSequences>
```

```
#List of identifiers to filter out
<Identifier>Identifier1
<Identifier>Identifier2
<Identifier>Identifier3
```

```
<Identifier>Identifier4
<Identifier>Identifier5

#Filter out all partial identifiers
<PartialIDs>

#List of files to filter out
<File>*\file1
<File>*\file2
<File>*\file3
<File>D:\CodeSuite\Code Development\CodeSuite UI\CodeDiff.frx
<File>D:\CodeSuite\Code Development\CodeSuite UI\CodeSuite.vbp
<File>D:\CodeSuite\Code Development\CodeSuite UI\PENS04.ICO

#List of folders to filter out
<Folder>D:\CodeSuite\documents
<Folder>D:\Congregation Beth David\Security

#Threshold filters
<Threshold>MinScore 10
<Threshold>MaxScore 98
<Threshold>FileNum 20
```

Command Line Interface

Running CodeSuite

CodeSuite can be invoked from the command line. This allows easy integration with other programs and also allows multiple comparisons to be run one after the other in a batch mode.

In order to run CodeSuite from the command line, your PATH environment must point to the CodeSuite location (see below for instructions).

To run a series of CodeSuite functions from a command file, type

```
cscl cfile
```

where:

cfile a command file listing one or more lines of arguments and options

To run a CodeSuite comparison, type

```
cscl -PGn [-option] [-option] dbase dir1 dir2 patt1 patt2 [lang1 lang2]
```

where:

dbase	name of the output database file
dir1	the first directory of files to compare
dir2	the second directory of files to compare
patt1	file patterns in dir1
patt2	file patterns in dir2
lang1	language of files in dir1 (BitMatch, CodeCross, CodeMatch only)
lang2	language of files in dir2 (BitMatch, CodeCross, CodeMatch only)

options:

h	print this description
PGn	which function to run (must be the first option)
	n = 0 for CodeMatch
	n = 1 for CodeDiff
	n = 2 for BitMatch
	n = 3 for CodeCross
	n = 4 for CodeCLOC
	n = 5 for DocMate
	n = 6 for CodeSplit

RSn	recursively examine subdirectories n = 0 for neither n = 1 for directory1 only n = 2 for directory2 only n = 3 for both (default)
SCn	directory contains source code (BitMatch, CodeCross, CodeMatch only) n = 0 for neither directory (BitMatch default) n = 1 for directory1 only n = 2 for directory2 only n = 3 for both (CodeMatch default)
FTn	n = number of files to report (default = 8)
SNn	compare files with the same name n = 0 for full comparison (default) n = 1 for same name comparison
CSn	compare statements (CodeMatch, CodeSplit only) n = 0 for no comparison n = 1 for comparison (default)
CCn	compare comments (CodeMatch, CodeSplit only) n = 0 for no comparison n = 1 for comparison (default)
CI n	compare identifiers (CodeMatch, CodeSplit, DocMate only) n = 0 for no comparison n = 1 for comparison (default)
CQn	compare instruction sequences (CodeMatch, DocMate only) n = 0 for no comparison n = 1 for comparison without listing sequences in the database (default) n = 2 for comparison with listing sequences in the database
QTn	n = minimum sequence to report (CodeMatch, DocMate only, default = 10)
IWn	ignore whitespace (CodeDiff only) n = 0 to consider n = 1 to ignore (default)
ICn	ignore case (CodeDiff only) n = 0 to consider n = 1 to ignore (default)
PMn	percentage calculation (CodeDiff only) n = 0 for percent of total lines of both files (default) n = 1 for percent of lines of first file n = 2 for percent of lines of second file
CDn	compare directories (CodeDiff only)

	n = 0 to not compare directories (default)
	n = 1 to compare directories
PTn	n = minimum percent to report (CodeDiff only, default = 0)
RMn	report matching lines in the database (CodeCLOC, CodeDiff only)
	n = 0 to report lines in both files that do not match (default)
	n = 1 to report lines in both files that do match
	n = 2 to not report any lines
FEb	b = binary number for encoding combinations, zero extended. First instance on the command line applies to folder 1 files. Second instance on the command line applies to folder 2 files. (BitMatch, DocMate only, default = 1)
	bit 0 = 1 for ASCII encoding
	bit 1 = 1 for UTF-8 encoding
	bit 2 = 1 for UTF-16LE encoding
	bit 3 = 1 for UTF-16BE encoding
	bit 4 = 1 for UTF-32LE encoding
	bit 5 = 1 for UTF-32BE encoding

To filter a CodeSuite database, type

```
cscl -PG10 dbase_in filter dbase_out
```

where:

dbase_in	input database file to filter
filter	filter file
dbase_out	output filtered database file

To export a CodeSuite database to XML, type

```
cscl -PG11 dbase_in xml_out
```

where:

dbase_in	input database file to export
xml_out	output XML file

To sort a CodeSuite database, type

```
cscl -PG12 dbase_in dbase_out
```

where:

dbase_in	input database file to export
dbase_out	output sorted database file

To create HTML reports from a CodeSuite database, type

```
cscl -PGn dbase report
```

where:

dbase	database file
report	HTML report file to generate

options:

PGn	format of HTML report to generate
	n = 20 for basic report and detailed reports
	n = 21 for basic report only
	n = 22 for basic report for online use

To create spreadsheets from a CodeSuite database, type

```
cscl -PGn [-option] [-option] dbase spreadsheet
```

where:

dbase	input database file to filter
spreadsheet	spreadsheet file(s) to generate

options:

PGn	which spreadsheet(s) to generate (must be the first option)
	n = 30 for distribution spreadsheet
	n = 31 for search spreadsheets
	n = 32 for summary spreadsheet
INn	interval (summary spreadsheet only, default = 10)
CSn	consider statements (search spreadsheet only)
	n = 0 for no consideration
	n = 1 for consideration (default)
CCn	consider comments (search spreadsheet only)
	n = 0 for no consideration
	n = 1 for consideration (default)
CLn	consider identifiers or words (search spreadsheet only)
	n = 0 for no consideration
	n = 1 for consideration (default)
CUn	consider URLs (search spreadsheet only)
	n = 0 for no consideration (default)

	n = 1 for consideration
ICn	ignore case (PID spreadsheet only) n = 0 to consider n = 1 to ignore (default)
SLn	simple list (PID spreadsheet only) n = 0 to show matching partial IDs and contributing IDs n = 1 to show only matching partial IDs (default)

To run SourceDetective, type

```
cscl -PG40 [-option] dbase_in dbase_out
```

where:

dbase_in	input database file
dbaseout	output database with search results

options:

STn	search for statements n = 0 do not search for statements n = 1 search for statements (default)
CO n	search for comments n = 0 do not search for comments n = 1 search for comments (default)
IDn	search for identifiers n = 0 do not search for identifiers n = 1 search for identifiers (default)
URn	search for URLs n = 0 do not search for URLs (default) n = search for n URLs
QSn	require quotes around search strings n = 0 do not require quotes around search strings n = 1 require quotes around search strings (default)

To restart a CodeSuite analysis from an interrupted analysis, type

```
cscl -PG50 dbase
```

where:

dbase	database from interrupted analysis to be restarted
-------	--

To create a spreadsheet of file types in a folder, type

```
cscl -PG60 [-option] dir spreadsheet
```

where:

dir	directory to analyze
spreadsheet	spreadsheet file to generate

options:

RSn	recursively examine subdirectories
n = 0	non-recursive
n = 1	recursive (default)

To preprocess HTML files, type

```
cscl -PG70 [-option] input_dir output_dir pattern
```

where:

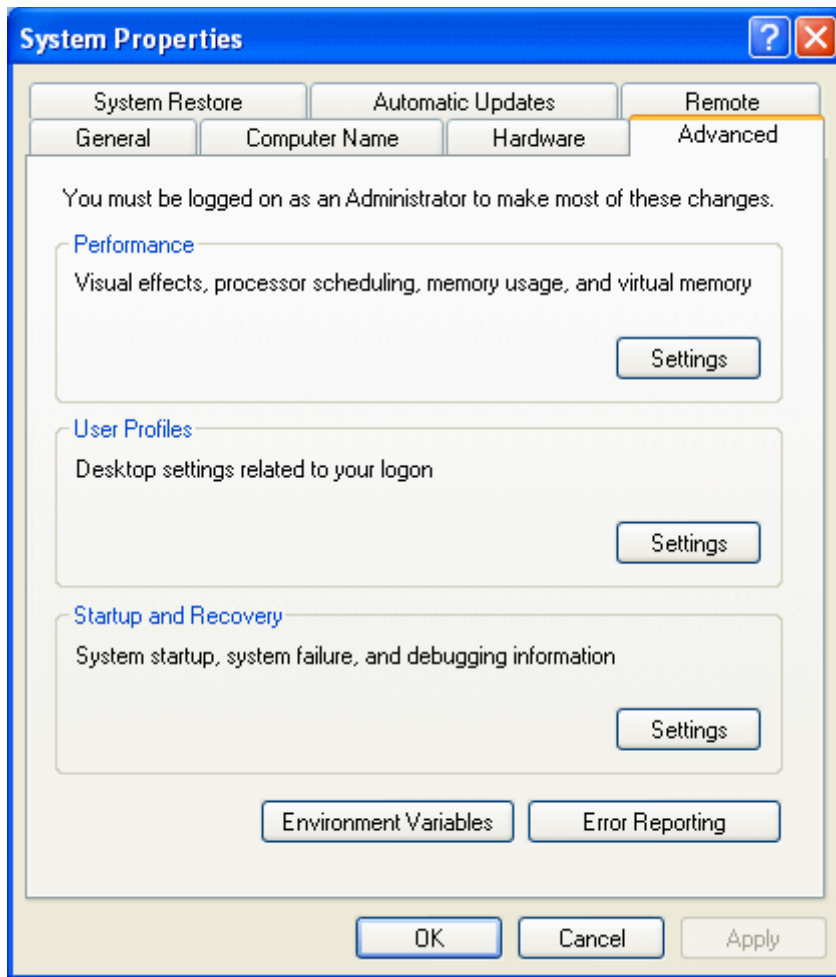
input_dir	directory containing the HTML files to preprocess
output_dir	directory to save the output files
pattern	file pattern(s)

options:

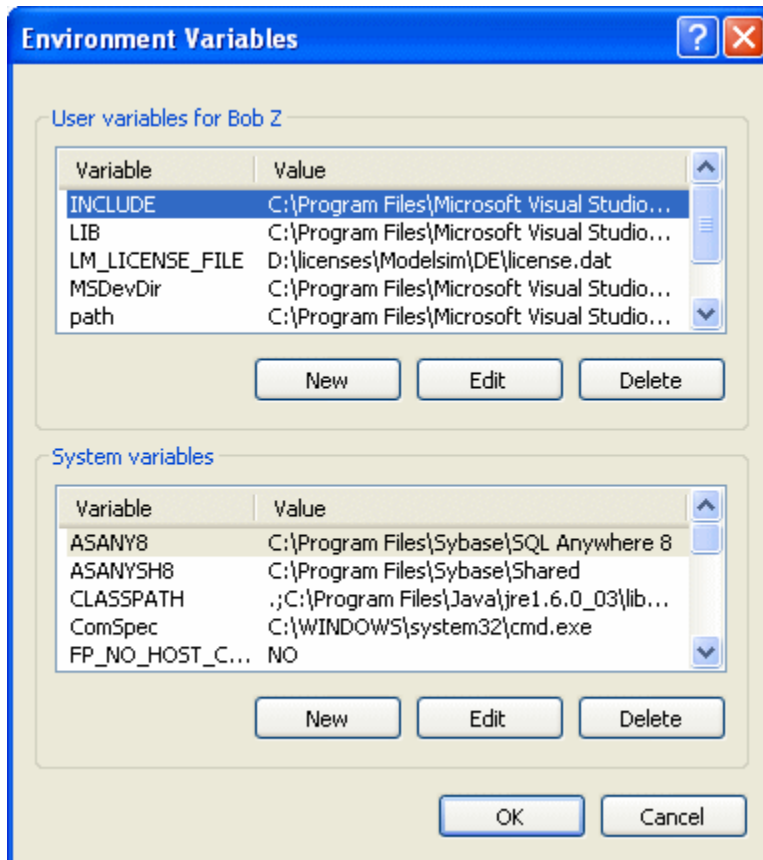
RSn	recursively examine subdirectories
n = 0	non-recursive
n = 1	recursive (default)

Setting up the PATH environment

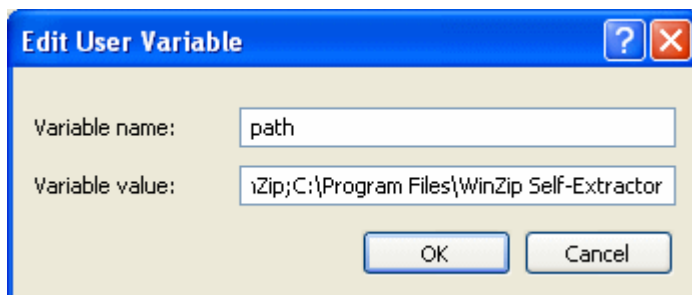
On your computer open Control Panel->Performance and Maintenance->System or right-click on My Computer and choose Properties. In the box that opens, click the Advanced tab to bring up the dialog box shown below.



Next, click the Environment Variables button in the lower left corner to bring up the dialog box shown below.



In the top window, highlight the path variable and click on the Edit button below the window to bring up the dialog box shown below.



At the end of the path value, type a semicolon followed by the full path to CodeSuite on your computer. If you installed to the default directory, the path is "C:\Program Files\SAFE\CodeSuite" (without the quotes). Click OK on this dialog box and the previous two dialog boxes and you are done.

XML File Format

XML is a standard text file format for containing data. An XML file contains hierarchical structures. A browser, such as Microsoft Internet Explorer, Mozilla Firefox, or Google Chrome, can open these files and display their contents for purposes of reviewing. The following sections describe how CodeSuite databases (CDB files) are transformed into XML.

Comments and messages

Comments within CDB files are converted directly into comments for XML. Comments in CDB files begin with # symbol while comments in XML enclose comments in comment tags. Messages are the XML form of permanent comments. An XML message consists of an arbitrary string enclosed within a pair of XML <Message> tags. Each XML element may have multiple messages which can occur anywhere between elements. Within a CDB file, <LComment> tags are converted into XML <message> tags.

CDB File

```
#This is a comment
<LComment>This is a message
```

XML File

```
<!-- This is a comment -->
<Message>This is a message</Message>
```

Database header

In a CDB file, at the start of the file, each application in CodeSuite places parameters and other information about the comparison.

CDB File

```
<Program>CodeMatch
<Version>5.4.0
<Date>03/17/11
<Time>15:20:42
<CComment>CodeSuite copyright 2003-2011 by Software Analysis and
Forensic Engineering Corporation
<LComment>SETTINGS
<Folder1>input\SourceDetective\x
<Subs1>F
<Folder2>input\SourceDetective\x
<Subs2>F
<Language>C
<FileType>*.c;*.h;*.txt
<SameName>F
```

```

<URLThresh>12
<Algorithm>Statement Matching
<Algorithm>Comment Matching
<Algorithm>Identifier Matching
<Algorithm>Instruction Sequence Matching
<FileThresh>1

```

XML File

```

<Application_Header>
  <Program>CodeMatch</Program>
  <Version>5.4.0</Version>
  <Date>03/17/11</Date>
  <Time>15:20:42</Time>
  <CComment>CodeSuite copyright 2003-2011 by Software Analysis
  and Forensic Engineering Corporation</CComment>
  <Message>SETTINGS</Message>
  <Folder1>input\SourceDetective\x</Folder1>
  <Subs1>F</Subs1>
  <Folder2>input\SourceDetective\x</Folder2>
  <Subs2>F</Subs2>
  <Language>C</Language>
  <FileType>*.c;*.h;*.txt</FileType>
  <SameName>F</SameName>
  <URLThresh>12</URLThresh>
  <Algorithm>Statement Matching</Algorithm>
  <Algorithm>Comment Matching</Algorithm>
  <Algorithm>Identifier Matching</Algorithm>
  <Algorithm>Instruction Sequence Matching</Algorithm>
  <FileThresh>1</FileThresh>
</Application_Header>

```

Filters

Filters from a CDB file are simply placed between <Raw_Filter> tags of the XML file as shown below.

CDB File

```

<Filters>
  <Filter><Statement>i = 0;
  <Filter><Comment>This is just a comment
  <Filter><Identifier>idl
</Filters>

```

XML File

```

<Raw_Filters>
  <Raw_Filter><Statement>i = 0;</Raw_Filter>
  <Raw_Filter><Comment>This is just a comment</Raw_Filter>

```

```
<Raw_Filter><Identifier>id1</Raw_Filter>
</Raw_Filters>
```

Post Application Transform List

The list of raw filter lines includes character codes which must be converted into special HTML representations as shown below.

CDB File

```
# CodeSuite filter file
<Filter><Threshold>MinScore 1
<LComment>Before Threshold MaxScore
<Filter><Threshold>MaxScore 100
# Before Threshold MinHits
<Filter><Threshold>MinHits 10
<Filter><Threshold>MaxHits 100
<Filter><Threshold>MaxFile 10
<Filter><AllStatements>
<Filter><AllSequences>
<Filter><PartialIDs>
```

XML File

```
<Post_Application_Transform_List>
  <Raw_Filters>
    <Raw_Filter>&lt;Filter&gt;&lt;Threshold&gt;MinScore
1</Raw_Filter>
    <Message>Before Threshold MaxScore</Message>
    <Raw_Filter>&lt;Filter&gt;&lt;Threshold&gt;MaxScore
100</Raw_Filter>
    <!--Before Threshold MinHits-->
    <Raw_Filter>&lt;Filter&gt;&lt;Threshold&gt;MinHits
10</Raw_Filter>
    <Raw_Filter>&lt;Filter&gt;&lt;Threshold&gt;MaxHits
100</Raw_Filter>
    <Raw_Filter>&lt;Filter&gt;&lt;Threshold&gt;MaxFile
10</Raw_Filter>
    <Raw_Filter>&lt;Filter&gt;&lt;AllStatements&gt;</Raw_Filte
r>
    <Raw_Filter>&lt;Filter&gt;&lt;AllSequences&gt;</Raw_Filter
>
    <Raw_Filter>&lt;Filter&gt;&lt;PartialIDs&gt;</Raw_Filter>
  </Raw_Filters>
</Post_Application_Transform_List>
```

Line Data Attributes

This XML `<Line_Data_Attributes>` type specifies an arbitrary string, together with optional `SourceDetective` data. The optional `SourceDetective` data is stored as one of two attributes (but not both): `Search_Hits` or `Search_Error`.

CDB File

```
<Line>assert (daemon_passive_states_.empty())
<YahooHits>0
```

XML File

```
<Line Search_Hits="0">assert
(daemon_passive_states_.empty())</Line>
```

File2

A `<File2>` structure specifies file information about a file in `folder2`. The file path and size of the file in bytes are always specified. If the file is processed as a text file, then the `<NumLines2>` tag specifies the number of lines in the file. The structure ends with a `<Data>` element.

CDB File

```
<File2>race.h
<Num_Lines2>134
<Size2>4989
...
```

XML File

```
<File2>
  <Name2>race.h</Name2>
  <Num_Lines2>134</Num_Lines2>
  <Size2>4989</Size2>
  <Data>
    ...
  </Data>
</File2>
```

Dir2

A `<Dir2>` structure contains a non-empty list of `<File2>` structures (previously described). A CDB file has a similar hierarchy, but sometimes the tags are not always in the expected order. In the following example CDB file, a `<Dir2>` tag appears directly after a `<Dir1>` tag, without an intervening `<File1>` tag. In effect, the `<Dir2>` and `<File1>` tags set a state, and any following `<File2>` tags use the most previously specified `<Dir2>` and `<File1>` (or even `<Dir1>`) tags.

CDB File

```
<Dir1>C:\SafeRuns\CSamps\files 1
```

```
<Dir2>C:\SafeRuns\CSamps\files 2
<File1>aaa.c
<File2>aaa.c
```

XML File

```
<Dir1>
  <Path1>C:\SafeRuns\CSamps\files 2</Path1>
  <File1>
    <Dir2>
      <Path2>C:\SafeRuns\CSamps\files 2</Path2>
      <File2>
        [...]
      </File2>
    </Dir2>
  </File1>
</Dir1>
```

File1

A `<File1>` structure specifies file information about a file in folder1. It contains a list of `<Dir2>` objects. The file path and size of the file in bytes are always specified. If the file is processed as a text file, the number of lines is also specified with the `<Num_Lines1>` tag. If no files are actually compared against `<File1>`, then the `<File1>` structure has a `<No_Comparees>` element; otherwise a minimum of at least one `<Dir2>` tag with at least one `<File2>` tag will be present.

CDB File

```
<File1>aaa.c
<Num_Lines1>24
<Size1>410
<Dir2>Z:\SAFE Corporation\code development\test\C\files2
<File2>bbb_case.c
...
<File1>bbb.c
<Num_Lines1>67
<Size1>741
<NoScore>
...
```

XML File

```
<File1>
  <Name1>aaa.c</Name1>
  <Num_Lines1>24</Num_Lines1>
  <Size1>410</Size1>
  <Dir2>
    ...
```

```

    </Dir2>
</File1>
<File1>
  <Name1>bbb.c</Name1>
  <Num_Lines1>67</Num_Lines1>
  <Size1>741</Size1>
  <No_Compare1>true</No_Compare1>
</File1>
...

```

Dir1

A `<Dir1>` structure contains a path to a directory and a non-empty list of `<File1>` structures (previously described). A CDB file has a similar hierarchy, but sometimes the tags are not always in the expected order. In the following example CDB file `<Dir2>` appears directly after a `<Dir1>` tag without an intervening `<File1>` tag.

CDB File

```

<Dir1>C:\SafeRuns\CSamps\files 1\A
<Dir2>C:\SafeRuns\CSamps\files 2\A
<File1>aaa.c
...

```

XML File

```

<Dir1>
  <Path1>C:\SafeRuns\CSamps\files 1\A</Path1>
  <File1>
    ...
  </File1>
  <File1>
    ...
  </File1>
</Dir1>

```

Data

The XML `<Data>` structure is used to encompass all results of the comparison data between two files. The ordering is critical. Any one of the items inside the `<Data>` tags may be absent, except the final score. In a CDB file, the scores for each comparison type are usually stored just after the comparison except for `<Identifiers>` where it is stored after the partial identifiers. There are no ordering requirements in the XML.

XML File

```

<Data>
  ...
</Data>

```

Statements

The XML `<Statements>` tag contains a list of `<Statement>` tags followed by an optional `<Statement_Score>` tag. If there is no `<Statement_Score>` tag, there were no statements to compare or the statements were not compared in this CodeSuite run. In a CDB file, the score occurs outside the `<Statements>` block while in an XML file the score occurs inside the `<Statements>` block. The XML `<Statement>` tag specifies the occurrences of a specific matching statement in the files. A string is specified, together with the line numbers where that statement occurs in both the first file and the second file. Note that only one number appears for each `<Line1>` and `<Line2>` tag whereas the corresponding CDB `<Lines1>` and `<Lines2>` tags may contain multiple line numbers. If SourceDetective has been run on comments, then the single hit count is stored as an attribute of the tag.

CDB File

```
<Statements>
<Line>assert (daemon_passive_states_.empty())
<YahooHits>0
<Lines1>85
<Lines2>124
<Line>to_string (__LINE__)
<BingHits>4
<Lines1>122
<Lines2>468 619
</Statements>
<StatementScore>1.000000
```

XML File

```
<Statements>
  <Statement>
    <Line Search_Hits="0">assert
      (daemon_passive_states_.empty())</Line>
    <Line1>85</Line1>
    <Line2>124</Line2>
  </Statement>
  <Statement>
    <Line Search_Hits="4">to_string (__LINE__)</Line>
    <Line1>122</Line1>
    <Line2>468</Line2>
    <Line2>619</Line2>
  </Statement>
  <Statement_Score>1</Statement_Score>
</Statements>
```

Comments

The XML `<Comments>` tag corresponds to a CDB `<Comments>` tag. The `<Comment_Score>` tag is optional. If there is no comment score, there were no comments to compare or the comments were not compared in this CodeSuite run. The XML `<Comment>` tag specifies the occurrences of a specific matching comment in the files. A string is specified, together with the line numbers where it occurs in both the first comparison file and the second file. Note that only one number appears for each `<Line1>` and `<Line2>` tag whereas the corresponding CDB `<Lines1>` and `<Lines2>` tags may contain multiple lines. If SourceDetective has been run on comments, then the single hit count is stored as an attribute of the tag.

CDB File

```
<Comments>
<Line>copyright notice in version.h
<BingHits>0
<Lines1>6
<Lines2>6
<Line>C++ Simulator
<YahooHits>13000
<Lines1>14 20
<Lines2>14
</Comments>
<CommentScore>0.926174
```

XML File

```
<Comments>
  <Comment>
    <Line Search_Hits="0">copyright notice in
    version.h</Line>
    <Line1>6</Line1>
    <Line2>6</Line2>
  </Comment>
  <Comment>
    <Line Search_Hits="13000">C++ Simulator</Line>
    <Line1>14</Line1>
    <Line1>20</Line1>
    <Line2>14</Line2>
  </Comment>
  <Comment_Score>0.926174</Comment_Score>
</Comments>
```

Identifiers

The XML `<Identifiers>` tag corresponds to a CDB `<IDs>` tag. The XML sub-element `<Identifier>` tag holds a single matching identifier in the files. The CDB `<ID>` tag may hold multiple identifiers separated by spaces. Therefore, a CDB `<ID>` tag may translate to many XML `<Identifier>` tags.

CDB File

```
<IDs>
<ID>belongs_to CurSituation
<YahooHits>344000 8
<URL>http://www.abc.com
<URL>http://www.safe-corp.com/index.htm
<URL>https://zeidmanconsulting.com/company.htm
<URL>https://www.amazon.com/Software-Detectives-Handbook-
Measurement-Infringement/dp/0137035330
<URL>http://www.samanna.biz/products.html
<URL>http://www.z-enterprises.com
<ID>x_to
<YahooHits>83
<URL>http://www.firtiva.com
<URL>http://zeidman.biz/blog
</IDs>
```

XML File

```
<Identifiers>
  <Identifier Search_Hits="344000">belongs_to</Identifier>
  <URL>http://www.abc.com</URL>
  <URL>http://www.safe-corp.com/index.htm</URL>
  <URL>https://zeidmanconsulting.com/company.htm</URL>
  <URL>https://www.amazon.com/Software-Detectives-Handbook-
Measurement-Infringement/dp/0137035330</URL>
  <URL>http://www.samanna.biz/products.html</URL>
  <URL>http://www.z-enterprises.com</URL>
  <Identifier Search_Hits="8">CurSituation</Identifier>
  <Identifier Search_Hits="83">x_to</Identifier>
  <URL>http://www.firtiva.com</URL>
  <URL>http://zeidman.biz/blog</URL>
</Identifiers>
```

Partial Identifiers

The XML<Partial_Identifier> tag corresponds to the CDB <PIDs> tag and represents all the partial identifier tags. In a CDB file the <PID1> and <PID2> tags contain partial identifiers separated by spaces. However, in the XML format, only one partial identifier (either <Partial_Identifier1> or <Partial_Identifier2>) appears per element. Therefore each <PID1> or <PID2> may translate to many XML tags, as shown in the example, below. The <Partial_Identifier> tags cannot have hits associated with them.

CDB File

```
<PIDs>
```

```

<PID1>(PCHAR (PUCHAR )
<PID1>W32N_REGSTR_PATH_
<PID2>_
<PID2>W32N_OSGetPlatformVersion
</PIDs>

```

XML File

```

<Partial_Identifiers>
  <Partial_Identifier1>(PCHAR</Partial_Identifier1>
  <Partial_Identifier1>(PUCHAR</Partial_Identifier1>
  <Partial_Identifier1>)</Partial_Identifier1>
  <Partial_Identifier1>W32N_REGSTR_PATH_</Partial_Identifier1>
  <Partial_Identifier2>_</Partial_Identifier2>
  <Partial_Identifier2>_OSGetPlatformVersion
  </Partial_Identifier2>
</Partial_Identifiers>

```

Sequences

The `<Sequences>` tag (for CDB and XML) specifies a list of all the matching instruction sequences, together with an optional sequence score value. If there is no `<Sequence_Score>` tag, then there were no sequences to compare or the sequences were not compared in this CodeSuite run. The XML `<Seq>` tag specifies the occurrences of specific matching instruction sequences in the files. A `<Seq>` tag uses one and only one `<InSeq>` element, followed by optional `<Instr>` elements. The `<Instr>` elements only show up when the CodeMatch option to "List sequences" is specified. The `<Instr>` tag is a list of strings. However, the instructions do not include whitespace. The XML `<In_Seq>` tag corresponds to the CDB `<InSeq>` tag and specifies a sequence of instructions matched in two files. Both XML and CDB specify the following sequence of three integers.

- `<In1>`: this tag specifies the starting line number from the first file.
- `<In2>`: this tag specifies the starting line number from the second file.
- `<Length>`: this tag specifies the number of instructions which matched.

CDB File

```

<Sequences>
<InSeq>118 615 3
<Instr>to_string
<Instr>for i
<InSeq>34 426 2
<Instr>y gantt
</Sequences>
<SequenceScore>0.368421

```

XML File

```

<Sequences>
  <Seq>
    <In_Seq>
      <In1>118</In1>
      <In2>615</In2>
      <Length>3</Length>
    </In_Seq>
    <Instr>to_string</Instr>
    <Instr>for</Instr>
    <Instr>i</Instr>
  </Seq>
  <Seq>
    <In_Seq>
      <In1>34</In1>
      <In2>426</In2>
      <Length>2</Length>
    </In_Seq>
    <Instr>y</Instr>
    <Instr>gantt</Instr>
  </Seq>
  <Sequence_Score>1</Sequence_Score>
</Sequences>

```

Lines

The XML `<Lines>` tag only appears in a database created by CodeDiff. The tag encloses either (not both) a list of `<Match>` tags or `<Diff>` tags followed by an optional XML `<Differences_Score>`. If there is no `<Differences_Score>` then there were no statements in the list of either `<Match>` or `<Diff>`. For examples, refer to the descriptions below for Match and Diff.

Match

An XML `<Match>` tag is used to denote statement matching data in a CDB file. This element will only occur in a CDB file resulting from execution of a CodeDiff application, and only where the "Report matched lines" option is specified. The first sub-element is an XML `<Line>` tag and specifies an entire line of text which appears in both files of a comparison. A non-empty series of both XML `<Line1>` tags and then `<Line2>` tags follow the `<Line>` tag and indicate line numbers where the matching lines occur. Each XML `<Line1>` tag or `<Line2>` tag contains one line number indicating the occurrence of the specified line in the corresponding file. A CDB `<Lines1>` tag may contain multiple line numbers; each line number is transformed into a separate XML `<Line1>` tag. The same is true for transforming the CDB `<Lines2>` tag into multiple XML `<Line2>` tags.

The following example illustrates a portion of output from CodeDiff, when the "Report matched lines" option is set. In the XML file, the XML `<Lines>` tag correlates to the CDB `<Statements>` tag. Also, the XML `<Match>` tag is used to enclose the data for each `<Line>` tag. The `<Match>` tag does not appear in the CDB file but is needed in the XML file to enclose the `<Line>`, `<Line1>`, and `<Line2>` tags.

CDB File

```
<Statements>
<Line>"*
<YahooHits>20010001
<Lines1>4
<Lines2>2 4
<Line>*/
<YahooHits>4300
<Lines1>6
<Lines2>8
</Statements>
```

XML File

```
<Lines>
  <Match>
    <Line Search_Hits="20010001">"*</Line>
    <Line1>4</Line1>
    <Line2>2</Line2>
    <Line2>4</Line2>
  </Match>
  <Match>
    <Line Search_Hits="4300">*/</Line>
    <Line1>6</Line2>
    <Line2>8</Line2>
  </Match>
</Lines>
```

Diff

An XML `<Diff>` tag is used to denote line matching data in a CDB file. This element will only occur in a CDB file generated by a CodeDiff run, and only where the "Report unmatched lines" option was specified. The first sub-element is an XML `<Line>` tag that specifies an entire line of text that appears in only one of the two comparison files. The occurrence of the entire line in only one of the two compared files, differentiates the XML `<Diff>` tag from the previously described XML `<Match>` tag, which required the statement to appear at least once in both compared files.

A non-empty series, of either (never both) XML `<Line1>` tags or `<Line2>` tags, follows the `<Line>` tag, and indicates line numbers where the line appears. Each XML `<Line1>` tag or `<Line2>` tag contains one line number. The numbers indicate the

occurrence of the specified statement line in the corresponding file. For example, an XML `<Line1>` tag specifies the line number in the first folder group. A CDB `<Lines1>` tag may contain multiple line numbers; each line number is transformed into a separate XML `<Line1>` tag. The same is true for transforming the CDB `<Lines2>` tag into multiple XML `<Line2>` tags.

The following example illustrates a portion of output from CodeDiff, when the "Report unmatched lines" option is specified. In the CDB file the CDB `<Differences>` tag surrounds the output. In the XML file, the XML `<Lines>` tag correlates to the CDB `<Differences>` tag. Also, the XML `<Diff>` tag is used to enclose the data for each `<Line>` tag. The `<Diff>` tag does not appear in the CDB file but is needed in the XML file to enclose the `<Line>`, `<Line1>`, and `<Line2>` tags.

CDB File

```
<Differences>
<Line>"*
<BingHits>20010001
<Lines1>4
<Line>*/
<BingHits>4300
<Lines2>23 34
</Differences>
```

XML File

```
<Lines>
  <Diff>
    <Line Search_Hits="20010001">"*</Line>
    <Line1>4 </Line1>
  </Diff>
  <Diff>
    <Line Search_Hits="4300">*/</Line>
    <URL>http://www.abc.com</URL>
    <URL>http://www.safe-corp.com/index.htm</URL>
    <URL>https://zeidmanconsulting.com/company.htm</URL>
    <Line2>23</Line2>
    <Line2>34</Line2>
  </Diff>
</Lines>
```

Data Comparison Hierarchy

The full data consists of a list, possibly empty, of all the `<Dir1>` structures.

XML File

```
<Data_Comparison_Hierarchy>
  <Dir1>
```

```

        ...
    </Dir1>
    <Dir1>
        ...
    </Dir1>
    ...
</Data_Comparison_Hierarchy>

```

Trailer

The XML `<Trailer>` and `<Summary>` tags specify summary information available only after the comparison is complete, such as the total number of bytes processed, including all files from folder 1 and folder 2, and the execution time. A completely processed file will end with up to seven tags. Although the XML file can utilize any ordering, the CDB summary information must always begin with `<Bytes1>` tag or `<Folder1Bytes>` tag. Also, the `<Bytes2>` tag or `<Folder2Bytes>` tag and `<ExTime>` tag must be present in any ordering. The tags `<Bytes1>` and `<Bytes2>` are equivalent to `<Folder1Bytes>` and `<Folder2Bytes>`, respectively. The tags `<Bytes1>` and `<Bytes2>` are deprecated.

CDB File

```

<Folder1Bytes>76723
<Folder2Bytes>68480
<ExTime>3 Seconds
# and optionally, the following tags...
<Folder1Lines>300
<Folder1Files>2
<Folder2Lines>1000
<Folder2Files>10

```

XML File

```

<Trailer>
  <Summary>
    <Bytes_Folder1>76723</Bytes_Folder1>
    <Bytes_Folder2>68480</Bytes_Folder2>
    <Time_Execution>3 Seconds </Time_Execution>
    <Lines_Folder1>300</Lines_Folder1>
    <Files_Folder1>2</Files_Folder1>
    <Lines_Folder2>1000</Lines_Folder2>
    <Files_Folder2>10</Files_Folder2>
  </Summary>
</Trailer>

```

The optional XML tag `<User_Abort>` is normally absent or "false" but is set to "true" if the user aborted before the comparison was completed.

CDB File

```
#### USER ABORTED ####
<Bytes1>76723
<Bytes2>68480
<ExTime>3 Seconds
# [and optionally, the following tags...
<Folder1Lines>300
<Folder1Files>2
<Folder2Lines>1000
<Folder2Files>10
```

XML File

```
<Trailer>
  <User_Abort>true</User_Abort>
  <Summary>
    <Bytes_Folder1>76723</Bytes_Folder1>
    <Bytes_Folder2>68480</Bytes_Folder2>
    <Time_Execution>3 Seconds </Time_Execution>
    <Lines_Folder1>300</Lines_Folder1>
    <Files_Folder1>2</Files_Folder1>
    <Lines_Folder2>1000</Lines_Folder2>
    <Files_Folder2>10</Files_Folder2>
  </Summary>
</Trailer>
```

CDB

The CDB structure defines the entire XML database file, which includes the previously defined structures. This is the full description of the contents of an XML file representing a CDB database.

XML File

```
<CDB>
  <Application_Header>
    ...
  </Application_Header>
  <Post_Application_Transform_List>
    ...
  </Post_Application_Transform_List>
  <Data_Comparison_Hierarchy>
    ...
  </Data_Comparison_Hierarchy>
  <Trailer>
    ...
  </Trailer>
</CDB>
```

Restarts

A file that was aborted and restarted, includes a string specifying the last restart time and date as shown below.

XML File

```
<Restarted>Restarted on 03/18/10 at 13:11:44</Restarted>
```

Contacting SAFE Corporation

Contacting SAFE Corporation



Software Analysis and Forensic Engineering Corporation

Web: www.SAFE-corp.com

Email: Support@SAFE-corp.com

Index

A

ABAP 99
ASCII 17
ASM-6502 99
ASM-65816 99
ASM-65C02 99
ASM-C55x 99
ASM-C67x 99
ASM-M68k 99
Authorization Key 7

B

BASIC 99
BitMatch 1, 12, 16
BitMatch Algorithms 19
BitMatch Basic Report 20
BitMatch Detailed Report 22

C

C programming language 99
C# 99
C++ 99
Case 41
CLOC 24, 26
CLOC spreadsheet 10, 82
COBOL 99
CodeCLOC 1, 12, 24
CodeCLOC Algorithm 26
CodeCLOC Basic Report 28
CodeCLOC Detailed Report 30
CodeCross 2, 13, 31
CodeCross Algorithm 33
CodeCross Basic Report 34
CodeCross Detailed Report 36
CodeCross Score 33
CodeDiff 2, 13, 38
CodeDiff Algorithm 41
CodeDiff Basic Report 42
CodeDiff Detailed Report 44
CodeGrid 18, 32, 40, 47, 57
CodeMatch 3, 13, 45
CodeMatch Algorithms 48
CodeMatch Basic Report 50

CodeMatch Detailed Report 53
CodeSplit 3, 13, 56
CodeSplit Basic Report 58
CodeSplit Detailed Report 60
CodeSuite Database 100
CodeSuite database format 100
CodeSuite filter format 105
CodeSuite-MP 18, 32, 40, 47, 57
Command line interface 107
Comment/String Filters 75, 79
Comment/string matching 19
Comment/String Matching 46, 48
Copyrights 5
Correlation Score 19, 49

D

D programming language 99
Database 100
Delphi 99
Distribution Spreadsheet 10, 86
DRI ASM 99

F

File Filters 75
File Menu 9
FileCount 3, 14, 62
FileIdentify 4, 14, 63
FileIsolate 4, 14, 65
Filter database 9
Filters 15, 73
Flash ActionScript 99
Folder Filters 77
Fortran 99
FoxPro 99

G

Go 99

H

Help Menu 15
Hit Filters 78
HTML Preprocessor 4, 14, 68
HTML report 10
HTML reports 81

I

- Identifier Filters 73, 79
- Identifier matching 19
- Identifier Matching 46, 49
- Instruction Sequence Matching 46, 48
- Internet 71

J

- Java 99
- JavaScript 99

K

- Kotlin 99

L

- Languages Enabled 8
- Languages Supported 99
- License Type 7
 - File size based 8
 - Time based 8
 - Unlimited 8
- Licenses 7, 15
 - Allocated 8
 - Remaining 8
- LISP 99
- LOC 26
- LotusScript 99
- Lua 99

M

- MASM 99
- MATLAB 99
- Menu 9
- MPE/iX 99

O

- Objective-C 99
- OpenEdge 99

P

- Partial Identifier Filters 79
- Pascal 99
- Patents 5
- Percentage 41
- Percentage options
 - Percentage of file pair 39
 - Percentage of first file 39
 - Percentage of second file 39
- Perl 99
- PHP 99
- PID Spreadsheet 10, 89
- PL/M 99

- PL/SQL 99

- PowerBuilder 99

- PowerHouse 99

- PowerShell 99

- Progress 99

- Prolog 99

- Python 99

R

- RealBasic 99

- Restart 13

- Ruby 99

S

- SAFE Corporation 5, 131

- Scala 99

- Search Spreadsheet 11, 92

- Sequence Filters 79

- Similarity Score 41

- Software Analysis and Forensic Engineering 131

- Sorting databases 10, 80

- SourceDetective 4, 14, 71

- SQL 99

- Statement Filters 74, 79

- Statement Matching 46, 48

- Statistics 14, 70

- Status Bar 15

- Structured Text 99

- Summary Spreadsheet 11, 96

- Swift 99

- System Requirements 6

T

- TAL 99

- TCL 99

- Threshold Filters 78

- Toolbar 9

- Tools Menu 12

- Trademarks 5

- TypeScript 99

U

- URLs 71

- UTF-16 17

- UTF-32 17

- UTF-8 17

V

- Verilog 99

- VHDL 99

View Menu 12
Visual Basic 99

W

Whitespace 41, 48
Windows 10 6
Windows 11 6

Windows 7 6
Windows 8 6
Windows Vista 6
X
XML 11, 115
XML database 98