

Telelogic Logiscope

CodeReducer
- Identifying Code Similarities

Version 6.5

Before using this information, be sure to read the general information under “Notices” section, on page 42.

This edition applies to **VERSION 6.5, TELELOGIC LOGISCOPE** (product number 5724V81) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 2008

US Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Table of Contents

1. Basic concepts.....	3
1.1. Why using Logiscope CodeReducer?.....	3
1.2. Differences vs. similarities.....	5
1.3. Search mechanism.....	6
2. Getting started with Logiscope CodeReducer.....	9
2.1. Use Case 1: Reducing code redundancies within a project.....	9
2.2. Use Case 2: Tracking new code in the new version of a project.....	20
2.3. Use Case 3: Searching a Reference Code.....	28
3. Command Line Mode.....	34
3.1. Logiscope create.....	34
3.2. Logiscope batch.....	37
4. Reference Guide.....	39
4.1. General settings.....	39
4.2. Display settings.....	41

About this manual

Audience

This reference manual is intended for **Telelogic® Logiscope™ CodeReducer** users such as software developers, project managers or quality engineers who want to identify source code similarities in order to factorize them.

Overview

Section 1 explains the basic concepts of code similarities, and presents several situations where enhances control over the source code.

Section 2 focuses on several real life use cases, detailing how the projects are created, and the results analyzed.

Section 3 details each parameter and its impact on the search results.

Contacting IBM Rational Software Support

Support and information for Telelogic products is currently being transitioned from the Telelogic Support site to the IBM Rational Software Support site. During this transition phase, your product support location depends on your customer history.

Product support

- If you are a heritage customer, meaning you were a Telelogic customer prior to November 1, 2008, please visit the [Logiscope Support Web site](#).

Telelogic customers will be redirected automatically to the IBM Rational Software Support site after the product information has been migrated.

- If you are a new Rational customer, meaning you did not have Telelogic-licensed products prior to November 1, 2008, please visit the [IBM Rational Software Support site](#).

Before you contact Support, gather the background information that you will need to describe your problem. When describing a problem to an IBM software support specialist, be as specific as possible and include all relevant background information so that the specialist can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, or messages that are related to the problem?
- Can you reproduce the problem? If so, what steps do you take to reproduce it?
- Is there a workaround for the problem? If so, be prepared to describe the workaround.

Telelogic Logiscope

Other information

For Rational software product news, events, and other information, visit the [IBM Rational Software Web site](#).

1. Basic concepts

1.1. Why using Logiscope CodeReducer?

A few real-life examples ...

When managing projects involving abundant source code, there are situations where an advanced comparison tool is required:

- A new version for your software has been developed, based on the previous version.

What proportion of older code has been reused as-is, which one has been modified, and to what extent?

- Your development team has to follow coding rules, but also use certain preferred algorithms, and avoid others considered non-optimal or unsafe.

How can you ensure that the preferred algorithms have been used (and where), and the forbidden ones have not been used?

- A subcontractor just delivered the latest version of a development package.

How can you verify what changes were made since the last version, excluding indentation, commentaries, identifiers names and functions order?

How can you detect if some functions have the exact same algorithm as old ones, you already paid for?

- A critical defect in a big project has been discovered, and the faulty code is being fixed.

How can you verify if the faulty code appears somewhere else in the project's thousands of files, in the same form or a similar one? Unless you fix the problem thoroughly, it is bound to appear again and generate another critical defect.

- A project was branched into two entities some years ago, and has now to be merged into a single project again.

How can you evaluate the overlapping of the two projects, and plan the merging, factorization and modularization?

All these situations can be addressed by <i>Logiscope CodeReducer</i> .

Logiscope CodeReducer's rationale

CodeReducer is a code similarity search tool, that can satisfy different needs:

- Search for all similar pieces of code in a given set of files,
- Search for code similar to a reference code in a given set of files,
- Comparison of source code files,
- Search for differences between two versions of a set of source code files.

In all these situations, CodeReducer identifies similar constructions, and provides you with information that can help you factorize and reduce your source code size.

Immediate benefits of this factorization are:

- Reduction of maintenance effort and cost on a smaller source code,
- Reduction of bugs possibilities,
- Improvement of code understandability when similar parts have been factorized,
- Ease of implantation of factorized code in other developments.

In addition, similarities searching allow powerful control on the source code:

- Find what the real modifications are between two versions,
- Look for already developed and tested code, saving effort and money.

1.2. Differences vs. similarities

Difference tools

Average difference tools are used to list what was removed, added, or left unchanged between two files.

This simple function can be useful when comparing two relatively close versions of a source file, but it quickly fails to give useful information on drastically different files, or even files where simple structural modifications were performed.

```
File Header

Block 1

Instruction 1

Block 2
```

Source file, version 1

```
File Header

Block 1

Instruction 3

Block 2
```

Source file, version 2

For example, when comparing version 1 and version 2 of this file, a difference tool will provide the following results:

- Both files are identical until the end of Block 1
- Instruction 1 was removed
- Instruction 3 was added
- Block 2 section is unchanged

```
File Header

Block 1

Instruction 3

Block 2
```

Source file, version 2

```
File Header

Comments

Block 1

Rewritten Instruction 3

Block 2
```

Source file, version 5

Now suppose that in version 5 of the file from previous example, several modifications were made:

- comments have been added throughout the source code,
- several variables have been renamed,
- “Instruction 3” has been reformatted to improve its readability.

In this situation, a difference tool will list all those modifications, and declare version 2 and version 5 as different, whereas the code is functionally the same.

Difference tools work well as long as the files are not too different.

Logiscope CodeReducer and similarities

Logiscope CodeReducer is able to compare source code basing its search on similarities, meaning code “with the same basis”, but not necessarily identical.

Consider the following source codes:

```
// Here is a comment
for (i=0;i<5;++i) {
    j = j+1;
}
```

A source code extract

```
for (j=5;
j<10;
++j) {
// Here is another kind of comment
    k = k+1;
}
```

A similar source code extract

Similarity comparison is not based on variable names, comments, indentation or code presentation, which means that:

- A classical difference tool will find that **all lines are different** between those two code snippets
- *CodeReducer* will find that they **are fully similar**

Logiscope CodeReducer does not look for identical, but similar code.

1.3. Search mechanism

For each supported language, CodeReducer uses a search mechanism based on an internal list of tokens.

These tokens are not only the elementary elements, but also the way to manage the search precision.

Tokens in Logiscope CodeReducer

A token is a structural element (control structure, structure and instructions delimiters, assignments, operators).

It is used to build an internal visualization of the source code, which is the basis for similarities search.

For all languages, tokens are broken down into categories:

- Category 1: Control structures (if, else, loops, switch, procedures, packages, classes),
- Category 2: Blocks (begin, end, {}),
- Category 3: Assignments,
- Category 4: Operators (+, -, *, ...),
- Category 5: Parenthesis and instructions terminators.

Important note:

Since CodeReducer is a similarities search tool, and not a difference tool, identifiers (variable, function names) are not considered as tokens.

Tokens and precisions

CodeReducer associates tokens to a precision:

- The higher the precision, the more tokens will be considered when looking for similarities.
- A given precision considers all tokens for this precision and lower ones too.

The tokens categories are naturally associated to search precisions.

Category 1 is associated to precision 1, Category 2 to precision 2, etc ...

The search results granularity is linked to the precision:

- Precision 1: Detection of similar algorithms,
- Precision 2: Precision 1 + detection of similar code structure,
- Precision 3: Precision 2 + detection of same number of variables assignments in code blocks,
- Precision 4: Precision 3 + detection of similar expressions,
- Precision 5: Precision 4 + detection of same number of instructions with similar contents.

Notes:

- For all precisions, a code portion can only qualify as similar to another one if it contains at least three “Category 1” (or “Precision 1”) tokens.
- Whatever the precision, identifiers are never use to evaluate code similarity.
- For precision 3: the number of assignments are detected, but the associated expressions can

Telelogic Logiscope

differ.

- For precision 4: similar expressions are detected, but other instructions (like sub-procedure call) can appear in the similar code.
- Always remember that raising the precision can reveal similarities that were not visible with a lower precision, because the set of tokens associated to different precisions are not subsets one of another.

2. Getting started with Logiscope CodeReducer

The situations described in this section refer to real-life problems to which *Logiscope CodeReducer* provides a solution.

For each problem, a step-by-step explanation will detail how to setup and use *Logiscope CodeReducer*, and how to analyze its results.

Before you start

In this session, you will use examples of source code files provided in the **samples** folder of the Logiscope installation directory.

As a precaution to keep original files safe, it is recommended to copy the samples subdirectory into a working directory of your own.

In addition, you will create Logiscope projects and associated repositories: i.e. sets of files containing internal data used by Logiscope. It is recommended to create a dedicated directory to store these data: e.g. a folder named **Logiscope\CodeReducer**.

The examples provided are considered to be on a Windows platform, but they can easily be adapted to a UNIX (Solaris or Linux) one.

2.1. Use Case 1: Reducing code redundancies within a project

The project development team has been renewed, and some critical knowledge about the source code has been lost in the process.

To avoid starting from scratch, and avoid unnecessary work, the first task is to identify the project's redundancies, and factorize them when possible.

In this example, you will use the C language “Mastermind” sample provided with the Logiscope standard distribution, and go through all the steps necessary to detect code redundancies.

Step 1: Start a Logiscope Studio Session

To begin a Logiscope *Studio* session:

On UNIX (i.e. Solaris or Linux):

- > launch the **vcs** binary .
- Start .

On Windows:

- > click the **Start** button and select the **Telelogic Logiscope <version>** item in **Telelogic Programs Group**.

The Logiscope splash screen is first displayed and then the Logiscope **Studio** main window appears.

Step 2: Create a Logiscope project

First, you shall define a Logiscope project which mainly consists in:

- the list of source files to be analysed,
 - applicable source code parsing options according to the compilation environment,
 - the verification modules to be activated on the source code files and the associated controls.
- In the **File** menu, select the **New...** command.

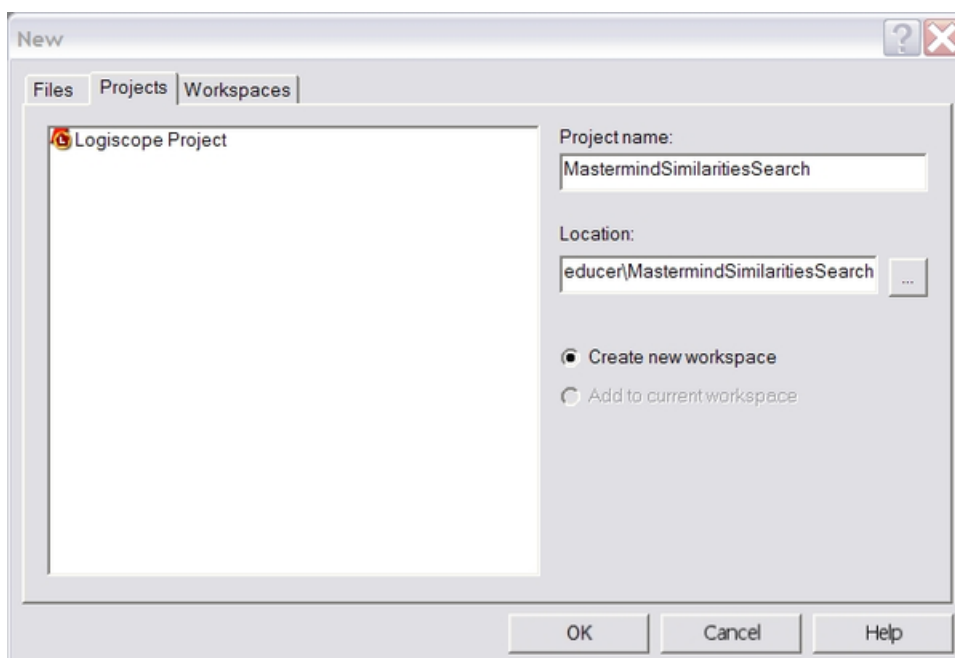
The Logiscope project creation wizard is invoked, and guides you through the process of defining a project.

The first dialog box prompts you to define the Project name and location.

- In the **Project name:** pane, enter the name for the new Logiscope project to be created. In the context of the guided tour, type “MastermindSimilaritiesSearch”.
- Then select its **Location:** i.e. the directory where the Logiscope project (i.e. a “.ttp” file) and the associated Logiscope repository will be created; the Logiscope repository is a folder in which Logiscope internal analysis result files are generated.

You can either keep the proposed default location or enter a the directory you lay have prepared as recommended in the Before You Start section: c:\Logiscope\CodeReducer

Note: By default, the project name is automatically added to the specified location. This implies that a subdirectory named <ProjectName> is automatically created.



- Validate the project name and location by pressing “OK”.

The next screen lets you decide what modules are associated to the project, and the language of the source files.

- Select the **Project Language:** i.e. the programming language in which are written the source code files to be analysed.
For the Mastermind project, select C.
- Select the **Project Modules:** i.e. the verification modules to be activated on the source files of the project .
For the guided tour, select only **CodeReducer**

Notes: At least one module should be selected. The *TestChecker* module cannot be selected with another module.



- Validate the project type definition by pressing “Next >”.
- For more details on the *QualityChecker* and *RuleChecker* modules, please refer to *Telelogic Logiscope RuleChecker and QualityChecker Getting Started*.
- For more details on the *TestChecker* module, please refer to *Telelogic Logiscope - TestChecker Getting Started*.

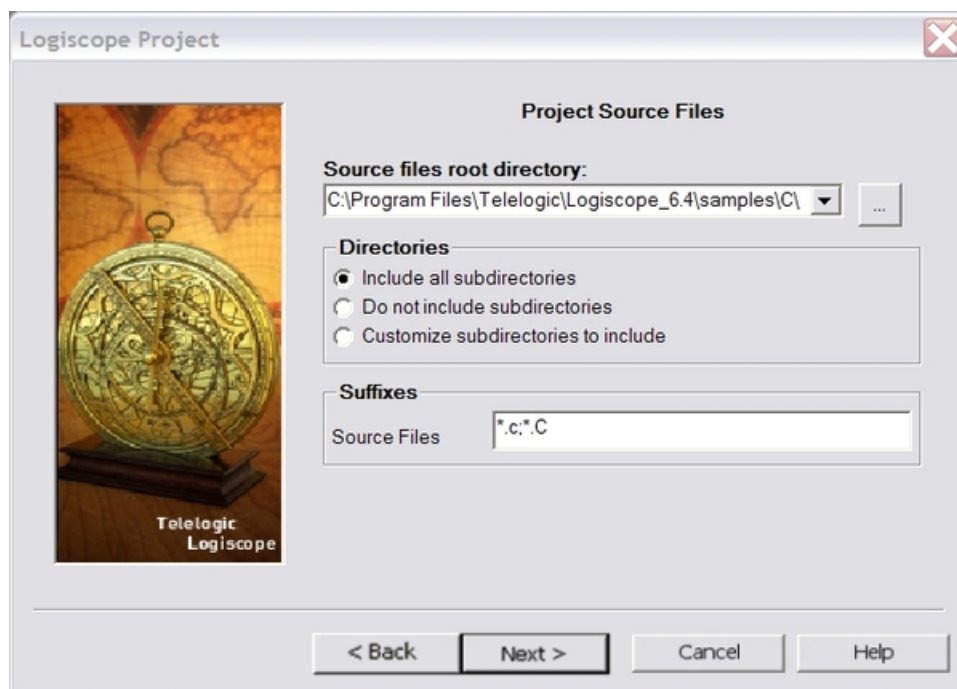
The **Project Source Files** dialog box allows to specify what source files are to be analysed and where they are located.

The sources for this Use Case are located in the samples\C\Mastermind folder of your Logiscope installation directory.

- **Source files root directory:** Browse to select the directory where the “Mastermind” source files are located: C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind.
- The **Directories** choice allows to select the list of repertories covering the application source files.
 - **Include all subdirectories** means that selected files will be searched for in every sub-directory of the source file root directory.
 - **Do not include subdirectories** means that only files included in the application directory will be selected.
 - **Customize subdirectories to include** allows the user to select the list of directories that include application files through a new page.

Keep the default setting.

- **Suffixes** choices allow to specify applicable source, header and inline file extensions needed in the above selected directories. Extensions shall be separated with a semi-colon. Keep the default values.



- Validate the project name and location by pressing “Next >”.

The following dialog box allows you to select some key settings of CodeReducer. They significantly impact the nature and number of the similarities found.

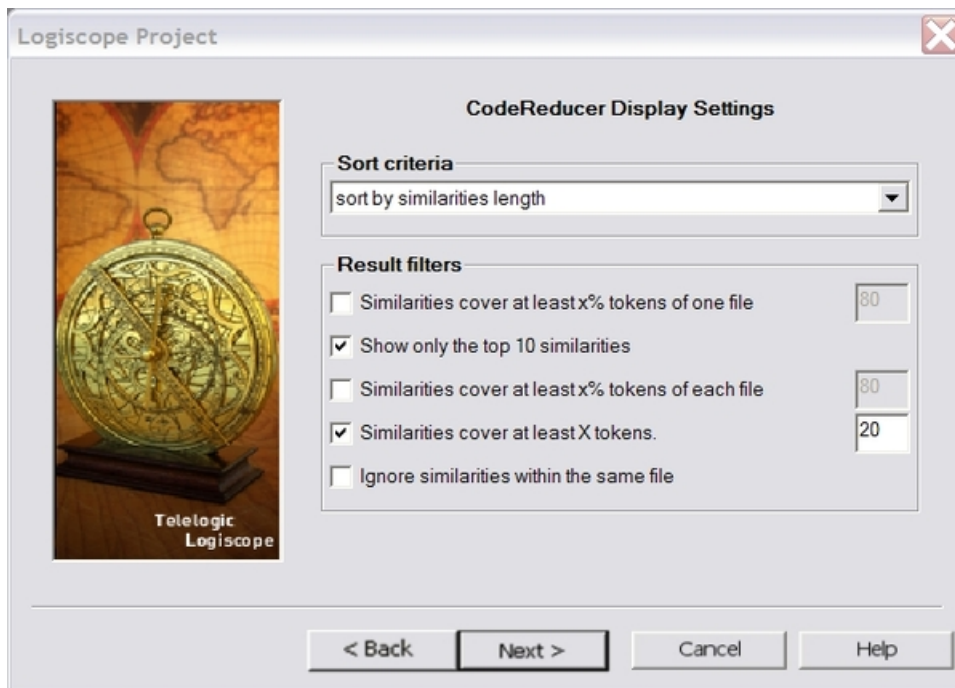
- **Search configuration:** Several type of scenarios are proposed according to the type of language.
Keep the default choice: i.e. C similar functions
- **Search options:** This section allows to specify the **precision:** i.e the set of tokens that will be considered for identifying similarities in the code.
Keep all default settings.



- Validate the CodeReducer Settings by pressing “Next >”.

The following dialog box completes the CodeReducer setting by defining how the similarities results should be filtered and displayed.

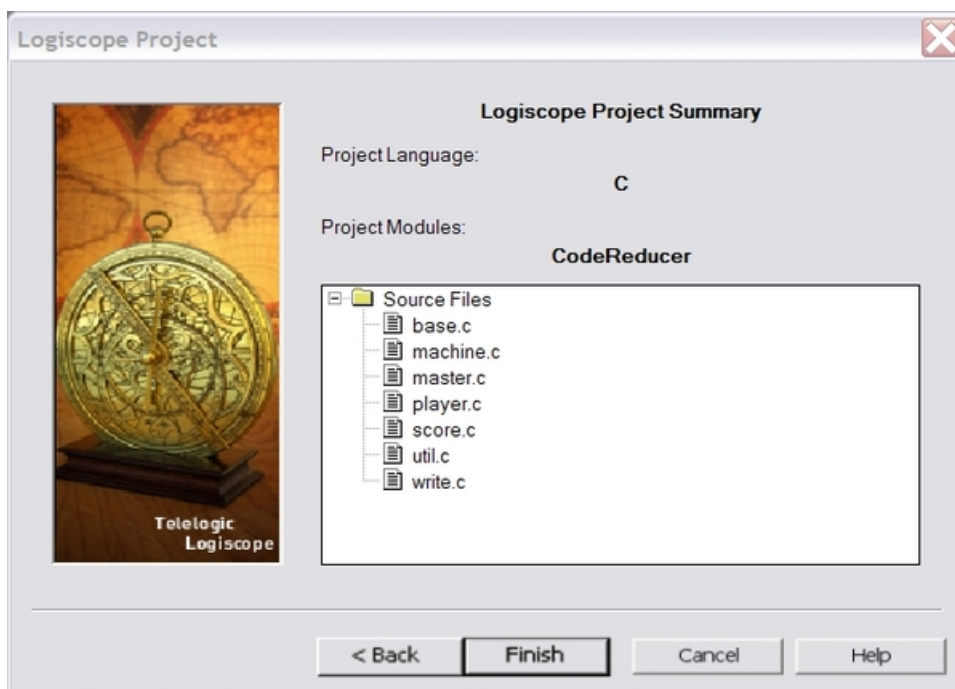
- **Sort criteria:** Keep the default value: i.e. Sort by similarities length.
- In the **Result filters:**
 - Check “Show only the top 10 similarities” in order to limit this first investigation to the most significant similarities found.
 - Check “Similarities cover at least X tokens” and type the value “20” assuming that you only want to display similarities with at least 20 tokens, discarding small similarities that would not qualify for a function factorization



- Validate the CodeReducer Display Settings by pressing “Next >”.

The final screen summarizes the project main attributes.

- Just click on “Finish” to finalize the creation process.



Step 3: Build the Logiscope project

Now that the project is fully defined, it can be “built”: i.e. parsing the project source files and extract all necessary information to identify code similarities.

- To build the project, simply use the appropriate menu item: “**Project – Build**”.

A new **Build** tab is added in the **Output Window** next to **Messages**. Several messages are displayed while parsing the source files and then loading the data showing that the build process is in progress.

As soon as the **Project [...] loaded.** message is displayed in the **Messages** tab, the project is built i.e. all the source files have been analysed and associated results generated and loaded.

Step 4: Analyse the code similarities found

After the build is completed, results can be displayed.

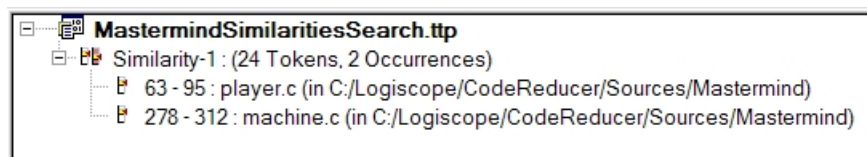
- Click on the “Displays CodeReducer Similarities Tree” icon, or choose the “**Browse – Reducer – Similarities Tree**” menu item.

The results are displayed as a dynamic tree, which nodes are associated to all similarities found in the project.

Each similarity node contains leaves associated to the location of the occurrences of similarities.

In our example, one similarity has been found, hence producing one node in the tree. This similarity (called “Similarity-1”):

- is 24 tokens long (which is greater than the 20 tokens limit we specified in the project settings),
- has two occurrences, located in the “player.c” and “machine.c” file.



- In the Similarity Tree tab, double-click on the “Similarity-1” node.

A new window is displayed showing the two files where a similarity occurrence was found in two distinct panes, one for each occurrences involved. The similar code is highlighted in blue.

Here two observations can be done:

- The two similar codes have different comments and line count. This is normal, since similarities are not based on presentation attributes.
- Apart from the presentation attributes, the code is very similar, meaning a factorization is possible. Factorizing the code between these two functions means twice as few possible defects and maintenance effort.

```

C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/player.c
63 {
64     char c[4][7];
65     int bad_in, code_valid;
66     int x;
67
68     code_valid = bad_in = FALSE;
69     kode->blacks = kode->whites = 0;
70     while (!code_valid)
71     {
72         code_valid = TRUE;
73         if (bad_in)
74             format_output("Previous code contains invalid color.\n",1);
75
76         format_output("Please enter guess",1);
77         printf("%d ", x);
78     }
79 }

C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/machine.c
284 {
285     extern int cur;
286     char c[4][7];
287     int bad_in, code_valid;
288     int x;
289
290     code_valid = bad_in = FALSE;
291     kode->blacks = kode->whites = 0;
292     while (!code_valid) /* get a player code */
293     {
294         code_valid = TRUE;
295         if (bad_in) /* Invalid player code */
296             format_output("Previous code contains invalid color.\n",1);
297         format_output("Please enter the correct code -> ",1);
298         printf("%d ", x);
299     }
300 }

```

- Click on the <ESC> key to close the window and be back to Logiscope Studio.

This simple example shows how you can easily look for similarities in a source code in order to find factorizable code, and improve the maintainability of the project source files.

Step 5: Change the CodeReducer Settings

The previous results have been obtained with the default settings of CodeReducer. They have been tuned for providing quick results. However, the most impressive results on this application are obtained with an other search configuration and a different precision level.

- Click on the “**Project – Settings...**” menu item.

The Logiscope Settings dialog box is open.

- Select the “CodeReducer” tab

You can now change the similarities search configuration to find different type of similar code in the application.

- In the “**Search configuration**” sections, select the “C similarities everywhere” scenario,
- In the “**Search options**” sections, select the higher precision level: **5** to get very similar

code.

General | Modules | Rule Set | Rules | Advanced
CodeReducer | Output | QualityChecker | Analysis

CodeReducer Settings

Search configuration

C similarities everywhere

This configuration is used to search for C similar code inside the whole application. Precision has to be set between 1 (low similarities) and 5 (strong similarities).

Search options

Precision: 1 2 3 4 5

Computes data for project comparison.

80 % of similarities at least between files.

- Select now the “Output” tab.
- In the “**Sort criteria**” section, select “sort by number of occurrences”,
- Keep the “**Result filters**” to focus on the most interesting similarities.

General | Modules | Rule Set | Rules | Advanced
CodeReducer | Output | QualityChecker | Analysis

CodeReducer Display Settings

Sort criteria

sort by number of occurrences

Result filters

Similarities cover at least x% tokens of one file 80

Show only the top 10 similarities

Similarities cover at least x% tokens of each file 80

Similarities cover at least X tokens. 20

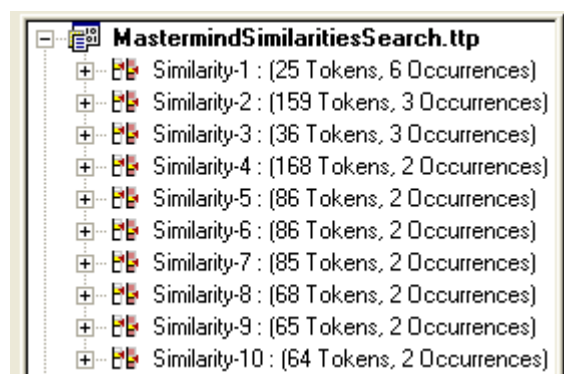
Ignore similarities within the same file

- click on the “OK” button to save the new CodeReducer settings.

As the project settings have been changed, the project shall be (re-)built to take into account these new configuration.

- Simply use the appropriate menu item: **“Project – Build”** or click on the corresponding icon.
- As soon as the build process is finished, refresh the **“Similarities”** tab by clicking on the **“Similarities Tree”** icon or activate the **“Browse – Reducer - Similarities Tree”** menu item.

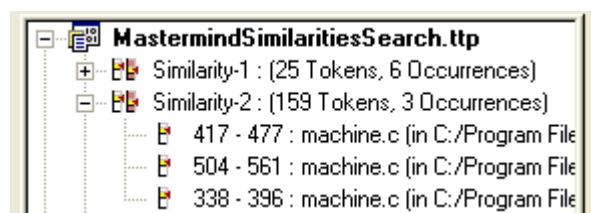
You can now discover much more similarities in the code than at the function level as shown in the previous step .



Indeed, the Similarity-1 corresponds to some code that occurs 6 times in the overall Mastermind application. Factorizing this section of code would significantly improve the level of maintainability of the source code.

But the most impressive finding relates to Similarity-2 node where a section of 159 identical tokens has been found 3 times in the code by CodeReducer.

- Expand the Similarity-2 node to locate these 3 occurrences.



The similarities are all located in the machine.c file and correspond to pieces of code of about 60 lines of code.

If you double-click on the **“Similarity-2”** node to open the corresponding code, you will discover that such a similarity may result from a large **“Copy -Paste”** action ... That is definitively a bad coding practice regarding code maintainability requirements.

CodeReducer helps you to find the code in just a few seconds. However, note that this search configuration may require a significant time to complete the analysis for a consequent volume of code.

Step 6: Generate a report

- Click on the “Displays CodeReducer Similarities HTML Report” icon, or choose the “**Browse – Reducer – Similarities Report**” menu item.

An HTML format is automatically generated where the similarities data are presented in tabular form as shown on the next page. You can browse within to get access to all results.

Date: 02 Oct 2008

This document contains information concerning the similarities analysis of the project [MastermindSimilaritiesSearch](#) made with Logiscope CodeReducer which is part of **Telelogic Logiscope™**.

Code Similarities

In the following table you will find for each code similarity found in the application:

- The name of the file.
- The line where the similarity starts in the file.
- The line where the similarity ends in the file.
- The length of the similarity in number of lines.

File	First line	Last line	Nb lines
Similarity 1 (6 occurrences - 25 tokens)			
C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\machine.c	353	360	8
C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\machine.c	365	372	8
C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\machine.c	434	441	8
C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\machine.c	446	453	8
C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\machine.c	518	525	8
C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\machine.c	530	537	8

The report has been saved in the reports.dir folder in the Logiscope Repository.

2.2. Use Case 2: Tracking new code in the new version of a project

A sub-contractor has delivered a new version for a development package, announcing that the whole source code had to be reworked in order to meet the requirements.

A series of functional tests validated the package, but you would like to verify what the actual changes brought by this new version are.

In this second Use Case, you will use both the C language “Mastermind” and “Mastermind_v2” samples provided with Logiscope, and compare these two versions.

The Mastermind_v2 sample is a copy of the Mastermind project, where the following modifications have been applied:

- The master.c and master.h files have been renamed newmaster.c and newmaster.h. The “#include master.h” directives have also been replaced by “#include newmaster.h”
- Likewise, the machine.c and machine.h files have been renamed engine.c and engine.h. The “#include machine.h” directives have also been replaced by “#include engine.h”
- In the file engine.c:
 - the “machine_read_file” function has been renamed “machineReadFile”, and its location in the file have been changed,
 - the “machine_plays” function has been renamed “machinePlays”, and its references in all the project are updated accordingly.
- In file newMaster.c, all internal variables have now different names
- In file newMaster.h, the constants MAX_TRY, MAX_TRY_MAC, BLACK, WHITE have been renamed, impacting several files.
- In file player.c, all comments have been removed, indentation is now different.
- In file util.c, functions are declared in a new order.
- In file score.c, internal variables are now declared in a new order
- In file base.c and base.h, function “format_output” has been renamed “to_screen”, impacting 227 calls in the project.
-

Step 1: Create a new Logiscope project in an existing Workspace.

When creating a project, Logiscope always associates it to a workspace, with the same name. So, the previous use case led to the creation of the “MastermindSimilaritiesSearch” workspace, which contains the “MastermindSimilaritiesSearch” project.

If you have closed the Logiscope Studio session started in the previous use case.

- Start a new session as in Section 2.1 Step 1
- Open the “MastermindSimilaritiesSearch” workspace created while performing the Use Case 1 by using either the “File > Open ...” command or the “File > Recent Worspaces ...” command.

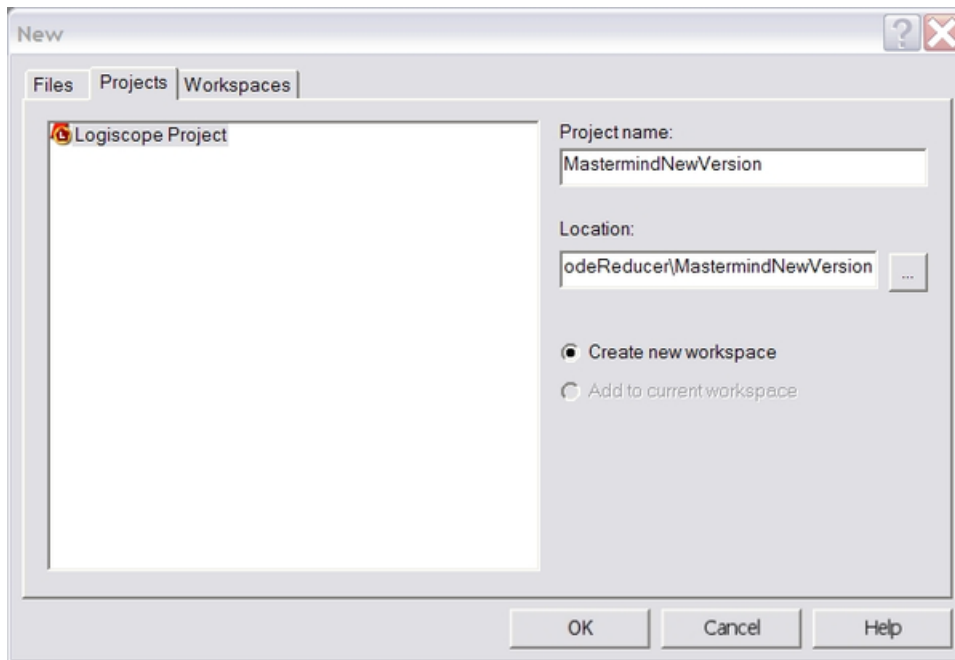
Once the workspace is created:

- In the **File** menu, select the **New...** command.

The Logiscope project creation wizard starts as already seen in the previous Use Case (see §2.1).

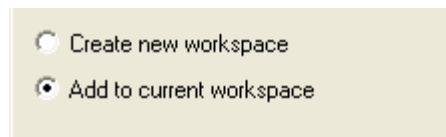
The first dialog box prompts you to define the Project name and location.

- In the **Project name:** type MastermindNewVersion.
- Then select its **Location:** : c:\Logiscope\CodeReducer



But in this context:

- Tick the **Add to current workspace** option. Indeed to allow comparing two projects, they shall be in the same Logiscope workspace.



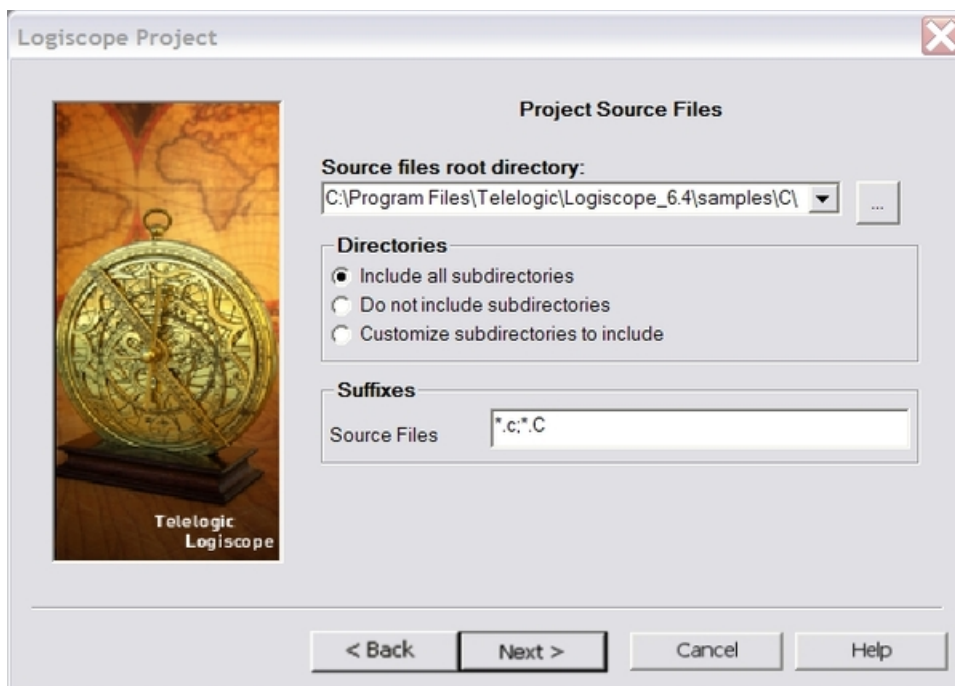
- Validate the project name and location by pressing “OK”.

As explained in section 2.1:

- In the **Logiscope Project Definition** dialog box, select C as the Project Language and CodeReducer as the project Module and Click “Next >”.



- Validate the project definition by pressing “Next >”.
- In the **Project Source Files dialog box**, browse to select **Source files root directory**: i.e. the directory where the version 2 of the “Mastermind” source files are located as the C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind_v2.



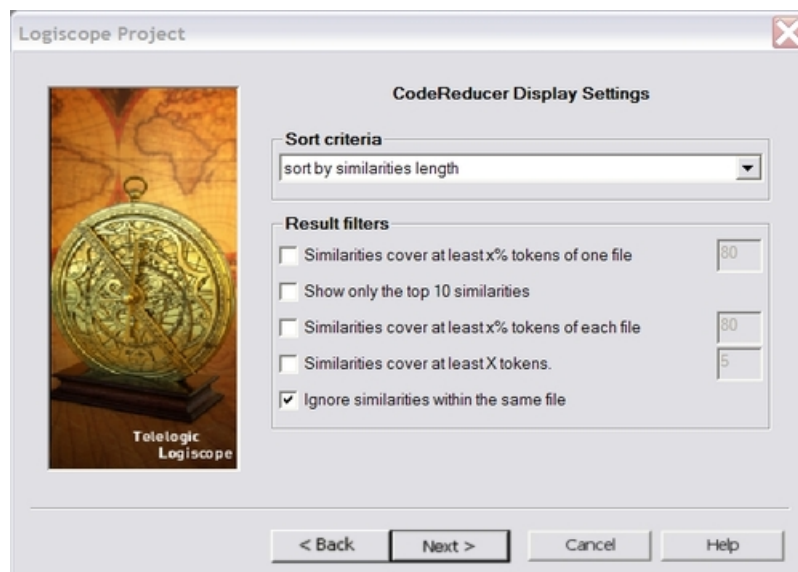
- Validate by pressing “Next >”.

In the **CodeReducer Settings** dialog box:

- **Search configuration:** Select “C similarities everywhere” as you will search for similarities every where in the code ... not only similar functions as seen in the first Use Case;
- Search options: keep all default options but check the box **Compute data for project comparison** as you definitively want to compare the two project.

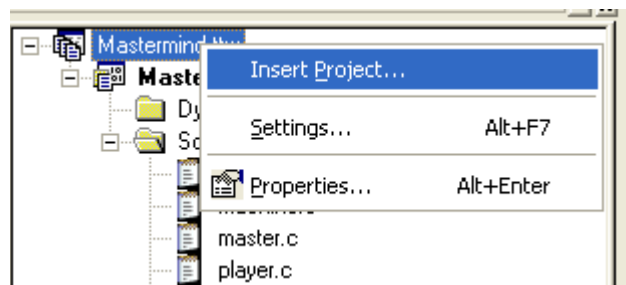


- Validate by pressing “Next >”.
- In the next dialog box, just check the display filter **Ignore similarities within the same file**



- Validate by pressing “Next >”.
- In the **Logiscope Project Summary** screen, just click on “Finish” to finalize the creation process.

Note: You also can add a previously created project into the current workspace by selecting it and using the contextual menu command “**Insert Project ...**”.



Step 2: Build the project

As done in the Use Case 1, you shall now extract the information from the source code files of the active project: i.e. the project in bold in the workspace.

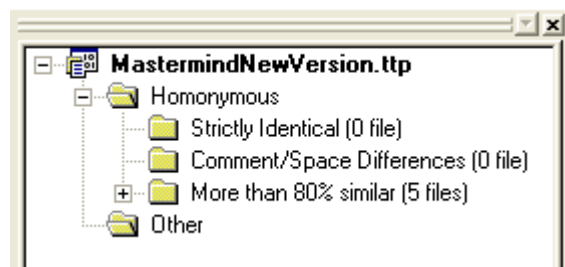
- Activate the command “**Project – Build**”.

Step 3: Display the results

After the build is completed, the projects comparison results can be displayed and interpreted.

- Click on the “Displays CodeReducer Project Comparison Tree”, or choose the “Browse – Reducer – Projects Comparison” menu item.

Results are displayed as a dynamic tree, project comparison creates two kinds of nodes:

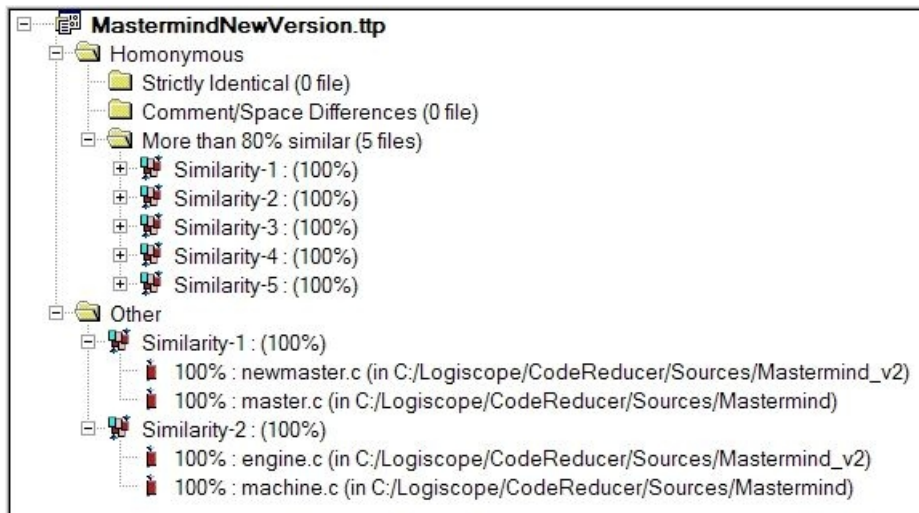


- Homonymous:
These nodes are associated to files having the same name in both projects. They are broken down into three categories:
 - Strictly identical files,
 - Files only different because of additional comments or spaces,
 - Similar files, within a given similarity ratio,

- Other:
These are files that do not have the same name, but contain similar code.

In the example, there are no more fully identical files between the two versions, nor any different because of different comments and spaces.

All files are just similar.



As shown in the above result window, even if some files have been modified, *Logiscope CodeReducer* indicates a similarity ratio of 100% for homonymous files, as well as for some files with different names. This means that:

- All homonymous files are structurally identical, no functional modification was actually made,
- The “Other”: i.e. not homonymous files result show that “newmaster.c” and “master.c” are identical, as well as “machine.c” and “engine.c”

With a single build on the project, you reach the conclusion that the new version is exactly similar to the previous one.

Logiscope CodeReducer found that
the new version has the **same functional source code** as the previous version.

As a matter of fact, if you had compared all files in pairs with a traditional “difference tool”, you would have (after some manual processing time) reached a rather different conclusion.

A “difference tool” found that
all files from the new version have **different source code** from the previous one.

Note: In the Tree tab, double-clicking on any Similarity node will display an overview of the similarity occurrences, as shown in Use Case 1

Step 5: Generate a report

- Click on the “Displays CodeReducer Projects Comparison HTML Report” icon, or choose the “**Browse – Reducer – Projects Comparison Report**” menu item.

The result is displayed in HTML format, and the project comparison summary is presented in tabular form.

IBM

Telelogic Logiscope Projects Comparison Report

Date: 02 Oct 2008

This document contains information concerning the comprison of the project C:\Program Files\Telelogic\Logiscope_6.5\samples\C\LogiscopeProjects\MastermindSimilaritiesSearch.ttp and the project C:\Program Files\Telelogic\Logiscope_6.5\samples\C\LogiscopeProjects\MastermindNewVersion.ttp made with Logiscope CodeReducer which is part of **Telelogic Logiscope™**.

Homonymous Files

Strictly Identical (0 file)

In the following table you will find for each homonymous file present in both projects and strictly identical:

- ▲ **Homonymous Files**
 - Strictly Identical
 - Comment/Space Diffs
 - More than 80% similar
- ▼ **Non Homonymous files**

- On the left frame, click on “**More than 80% similar**” menu item.

The list of corresponding files is now displayed as shown on the next page.

This example shows the capabilities of Logiscope CodeReducer when looking for functional differences between two versions of the same project.

More than 80% similar (5 files)

In the following table you will find for each homonymous file present in both projects and showing more than 80% similarities

- The pathname of the file in the first project.
- The percentage of line of this file common to the one from the second project
- The pathname of the file in the second project.
- The percentage of line of this file common to the one from the first project

File	Common
File: base.c	
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind_v2\base.c	100%
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind\base.c	100%
File: player.c	
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind_v2\player.c	100%
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind\player.c	100%
File: score.c	
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind_v2\score.c	100%
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind\score.c	100%
File: util.c	
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind_v2\util.c	100%
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind\util.c	100%
File: write.c	
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind_v2\write.c	100%
C:\Program Files\Telelogic\Logiscope_6.5\samples\C.Mastermind\write.c	100%

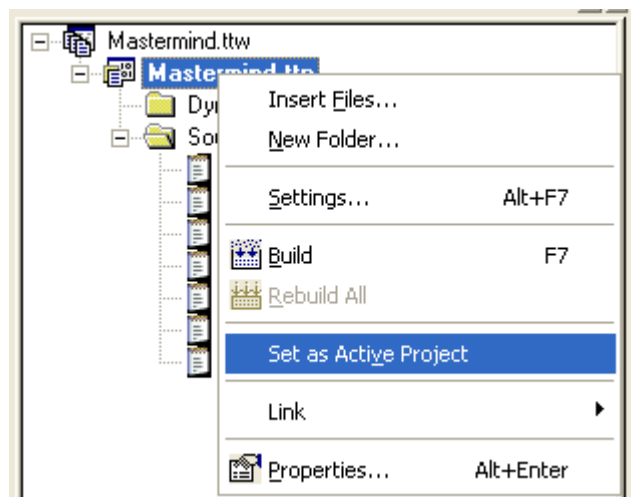
2.3. Use Case 3: Searching a Reference Code

A particular algorithm used in your product has been rewritten. You would like to check if other parts of the product are using the same old algorithm, and replace them by the new version.

In this example, you will reuse the project defined in Use Case 1.

Step 1: Modify project settings

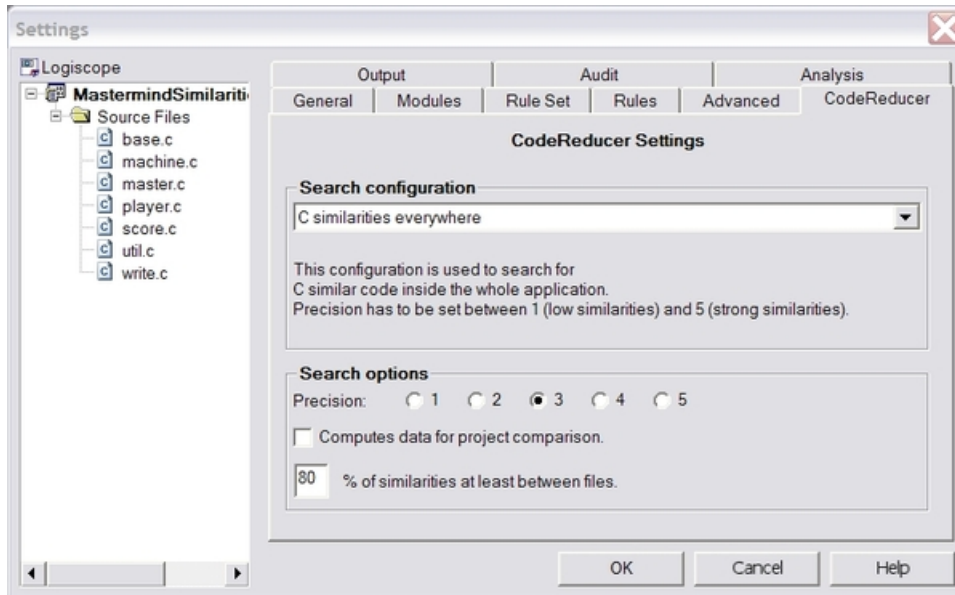
- In the File View tab select the project MastermindSimilaritiesSearch and activate the contextual menu command “**Set as Active project**”.



- Activate the command “**Project – Settings**”.
- Click on the “CodeReducer” tab.

All CodeReducer Settings can be modified for a new search for instance using a different precision level or a new scenario.

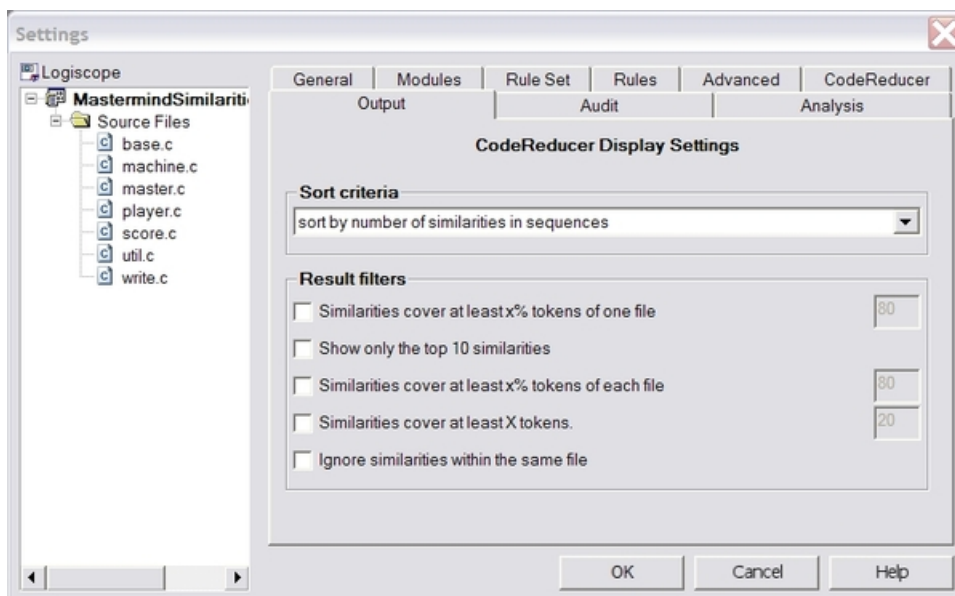
- In Search configuration; select the “C similarities everywhere” scenario.



- Click now on the “Output” tab.

The CodeReducer Display Settings can also be modified:

- In Sort criteria: select the “sort by number of similarities in sequences” option.
- Uncheck all Result filters, if any.



- Click “Ok” to validate the modifications.

Step 2: Rebuild the project

Now that the settings have been altered, the project has to be built again, so that these new parameters values are taken into account.

- Activate the command “**Project – Build**”.

Step 3: Look for reference code

After the build is completed, looking for a reference source code is fairly easy.

Suppose you have to modify the code in file “player.c”, at line 22, so that the 'if' structure nested in the 'for' loop is followed by an 'else' statement.

```
for (x = 0; x < 4; x++)
  if (guesses[cur].pegs[x].color == code.pegs[x].color)
  {
    /* guess color = code color */
    code.pegs[x].used = guesses[cur].pegs[x].used = BLACK;
    guesses[cur].blacks++;
    PDEBUG(0, guesses[cur].blacks);
    printf("b");
  }
```

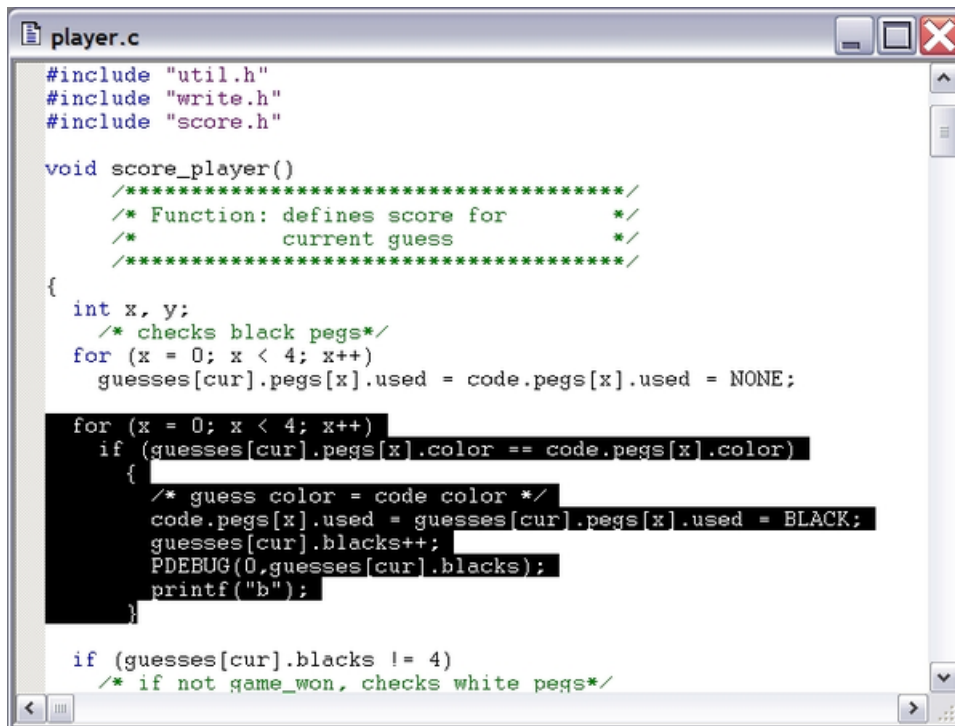
Original code

```
for (x = 0; x < 4; x++)
  {
    if (guesses[cur].pegs[x].color == code.pegs[x].color)
    {
      /* guess color = code color */
      code.pegs[x].used = guesses[cur].pegs[x].used = BLACK;
      PDEBUG(0, guesses[cur].blacks);
      guesses[cur].blacks++;
      printf("b");
    } else
    {
      printf("'if' condition failed for loop index %d", x);
    }
  }
```

Code after modification (added code is in red boldface)

The problem here is to look for other similar “if nested in a for loop” structures in the project. This search is achieved in two steps:

1. Identify the reference code
 2. Look for similar code
- To **identify the reference code**: simply select the code in file player.c, from line 22 to 30.



```

player.c
#include "util.h"
#include "write.h"
#include "score.h"

void score_player()
/******
/* Function: defines score for
/* current guess
/******
{
int x, y;
/* checks black pegs*/
for (x = 0; x < 4; x++)
guesses[cur].pegs[x].used = code.pegs[x].used = NONE;

for (x = 0; x < 4; x++)
if (guesses[cur].pegs[x].color == code.pegs[x].color)
{
/* guess color = code color */
code.pegs[x].used = guesses[cur].pegs[x].used = BLACK;
guesses[cur].blacks++;
PDEBUG(0,guesses[cur].blacks);
printf("b");
}

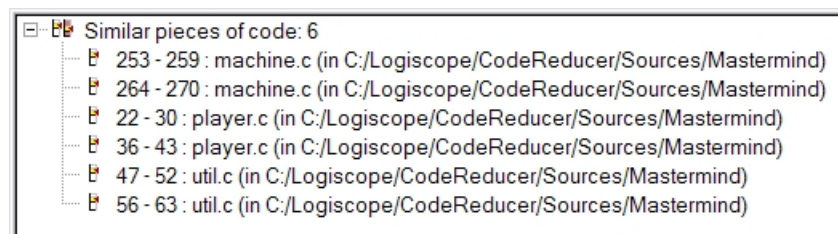
if (guesses[cur].blacks != 4)
/* if not game_won, checks white pegs*/

```

- To **look for similar code**, click on the “Searching code similar to selected one”, or choose the “Browse – Reducer – Searching Code” menu item.

Results are displayed as a dynamic tree, which nodes are associated to all occurrences of similar code portions found in the project.

In our example six similar portions of code were found, hence producing six nodes in the tree:



```

Similar pieces of code: 6
├── 253 - 259 : machine.c (in C:/Logiscope/CodeReducer/Sources/Mastermind)
├── 264 - 270 : machine.c (in C:/Logiscope/CodeReducer/Sources/Mastermind)
├── 22 - 30 : player.c (in C:/Logiscope/CodeReducer/Sources/Mastermind)
├── 36 - 43 : player.c (in C:/Logiscope/CodeReducer/Sources/Mastermind)
├── 47 - 52 : util.c (in C:/Logiscope/CodeReducer/Sources/Mastermind)
└── 56 - 63 : util.c (in C:/Logiscope/CodeReducer/Sources/Mastermind)

```

- In the Tree tab, double-click on the “Similar pieces of code” root to display an overview of the similarity occurrences.

Telelogic Logiscope

This overview shows the first five code portions similar to the reference code.

C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/machine.c

```
253     for (x = 0; x < 4; x++) /* searching black pegs */
254         if (guesses[cur].pegs[x].color == code.pegs[x].color)
255             {
256                 /* guess color = code color */
257                 code.pegs[x].used = guesses[cur].pegs[x].used = BLACK;
258                 guesses[cur].blacks++;
259             }
```

C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/machine.c

```
264     for (y = 0; y < 4; y++) /*searching white pegs */
265         if (guesses[cur].pegs[x].color == code.pegs[y].color &&
266             !code.pegs[y].used && !guesses[cur].pegs[x].used)
267             {
268                 code.pegs[y].used = guesses[cur].pegs[x].used = WHITE;
269                 guesses[cur].whites++;
270             }
```

C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/player.c

```
22     for (x = 0; x < 4; x++)
23         if (guesses[cur].pegs[x].color == code.pegs[x].color)
24             {
25                 /* guess color = code color */
26                 code.pegs[x].used = guesses[cur].pegs[x].used = BLACK;
27                 guesses[cur].blacks++;
28                 PDEBUG(0,guesses[cur].blacks);
```

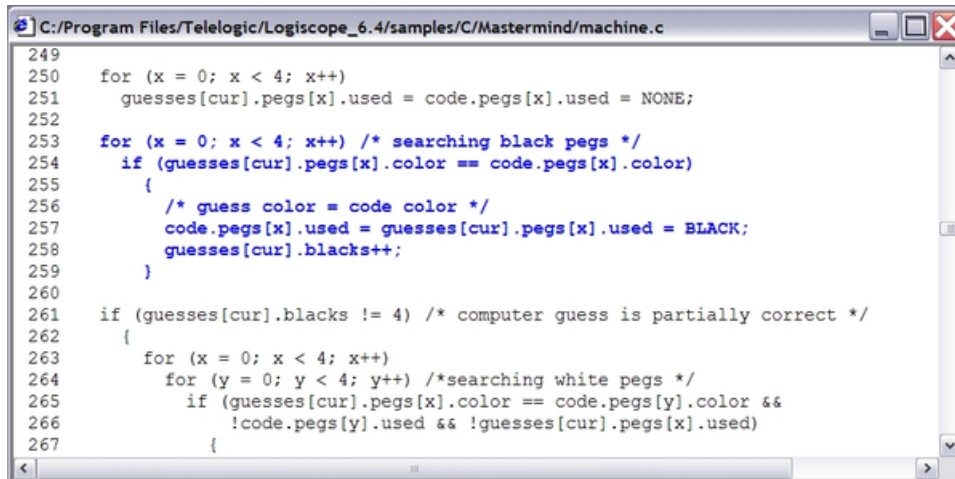
C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/player.c

```
36     for (y = 0; y < 4; y++)
37         if (guesses[cur].pegs[x].color == code.pegs[y].color &&
38             !code.pegs[y].used && !guesses[cur].pegs[x].used)
39             {
40                 code.pegs[y].used = guesses[cur].pegs[x].used = WHITE;
41                 guesses[cur].whites++;
42                 printf("w");
```

C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/util.c

```
47     for (x = 0; x < 4; x++)
48         if (guesses[cur].pegs[x].color == dummy[z].pegs[x].color)
49             {
50                 dummy[z].pegs[x].used = guesses[cur].pegs[x].used = BLACK;
51                 dummy[z].blacks++;
52             }
53     if (dummy[z].blacks != 4)
```

To access to each one of the similar code portions, just click on its associated node in the result tree. For example, clicking on the tree's first node displays the following window:



```
C:/Program Files/Telelogic/Logiscope_6.4/samples/C/Mastermind/machine.c
249
250     for (x = 0; x < 4; x++)
251         guesses[cur].pegs[x].used = code.pegs[x].used = NONE;
252
253     for (x = 0; x < 4; x++) /* searching black pegs */
254         if (guesses[cur].pegs[x].color == code.pegs[x].color)
255             {
256                 /* guess color = code color */
257                 code.pegs[x].used = guesses[cur].pegs[x].used = BLACK;
258                 guesses[cur].blacks++;
259             }
260
261     if (guesses[cur].blacks != 4) /* computer guess is partially correct */
262     {
263         for (x = 0; x < 4; x++)
264             for (y = 0; y < 4; y++) /*searching white pegs */
265                 if (guesses[cur].pegs[x].color == code.pegs[y].color &&
266                     !code.pegs[y].used && !guesses[cur].pegs[x].used)
267                     {
```

Now that all source code similar to the reference have been found, you can modify them as needed.

Conclusion

This example showed how one can pinpoint code similar to a given reference in two easy steps, allowing effortless propagation of an algorithm modification.

3. Command Line Mode

3.1. Logiscope create

Logiscope projects: i.e. “.ttp” file are usually built using Logiscope **Studio** as described in chapter *Project Settings* or in the *Logiscope RuleChecker & QualityChecker Getting Started* documentation.

The logiscope **create** tool builds Logiscope projects from a standalone command line or within makefiles (replacing the compiler command) .

Command Line Mode

When started from a standard command line, The **create** tool creates a new project file with the information provided on the command line.

For a complete description of the command line options, please refer to the Command Line Options paragraph.

When used in this mode, there are two different ways for providing the files to be included into the project:

Automatic search

This is the default mode where the tool automatically searches the files in the directories.

Key options having effect on this modes are:

-root <root_dir> : the root directory where the tool will start the search for source files. This option is not mandatory, and if omitted the default is to start the search in the current directory.

-recurse : if present indicates to the tool that the search for source files has to be recursive, meaning that the tool will also search the subdirectories of the root directory.

File list

In this mode, the tool will look for the **-list** option which has to be followed by a file name. This provided file contains a list of files to be included into the project. The file shall contain one filename per line.

Example: Assuming a file named `filelist.lst` containing the 3 following lines:

```
/users/logiscope/samples/C/mstrmind/master.c
/users/logiscope/samples/C/mstrmind/player.c
/users/logiscope/samples/C/mstrmind/machine.c
```

Using the command line:

```
create aProject.ttp -rule -reducer -lang c -list filelist.lst
```

will create a new Logiscope C project file named `aProject.ttp` containing 3 files: `master.c`, `player.c`

and `machine.c` on which *RuleChecker* and *CodeReducer* modules will be activated.

Makefile Mode

When launched from makefiles, **create** is designed to intercept the command line usually passed to the compiler and uses the arguments to build the Logiscope project.

The project makefiles must be modified in order to launch **create** instead of the compiler. In this mode, the name of the project file (".ttp" file) has to be an absolute path, otherwise the process will stop.

When used inside a Makefile, **create** uses the same options as in command line mode, except for:

`-root`, `-recurse`, `-list` : which are not available in this mode

`--` : which introduces the compiler command.

The following lines can be introduced in a Makefile to build a Logiscope project file :

```
CREATE=create /users/projects/myProject.ttp -rule -reducer -lang c
CC=$(CREATE) -- gcc
CPP=$(CC) -E
...
```

In this mode, the project file building process is as follows:

1. **create** is invoked for each file by the make utility, instead of the compiler.
2. When **create** is invoked for a file it adds the file to the project, with appropriate preprocessor options if any, then Create starts the normal compilation command which will ensure that the normal build process will continue.
3. At the end of the make process, the Logiscope project is completed and can be used either using Logiscope **Studio** or with the **batch** tool (see next section).

*Note: Before executing the makefile, first clean the environment in order to force a full rebuild and to ensure that the **create** will catch all files.*

Options

The create options are the following:

`create -lang c`

`<ttp_file>`

name of a Logiscope project to be created (with the .ttp extension).
Path has to be absolute if the option `--` is used.

`[-root <directory>]`

where `<directory>` is the starting point of the source search. Default is the current directory.
This option is exclusive with `-list` option.

Telelogic Logiscope

<code>[-recurse]</code>	if present the source file search is done recursively in subfolders.
<code>[-list <list_file>]</code>	where <code><list_file></code> is the name of a file containing the list of filenames to add to the project (one file per line). This option is exclusive with <code>-root</code> option.
<code>[-repository <directory>]</code>	where <code><directory></code> is the name of the directory where Logiscope internal files will be stored.
<code>[-source <suffixes>]</code>	where <code><suffixes></code> is the list of accepted suffixes for source file to be placed in project folder "Source Files" (default is <code>"*.c;*.C"</code>)
<code>[-no_compilation]</code>	avoid compiling the files if the <code>--</code> option is used
<code> [--]</code>	when used in a makefile, introduces the compilation command with its arguments.
<code>-reducer</code>	to select the CodeReducer verification module
<code>[-precision <value>]</code>	where <code><value></code> is the desired precision for similarities search. Accepted value is an integer between 1 and 5.
<code>[-percent <value>]</code>	where <code><value></code> is the desired percentage of similarities to retain files. Accepted value is an integer between 0 and 100.
<code>[-multi]</code>	if present, project comparison will be activated when the current project is part of a workspace containing two projects.
<code>[-config <config-id>]</code>	where <code><config-id></code> is the identifier of one of the configuration available for the current language. Possible values are - (c-function c-everywhere) for C language - (cpp-class cpp-function cpp-everywhere) for C++ language - (java-class java-package java-function java-everywhere) for Java language - (ada-package ada-function ada-everywhere) for Ada language
<code>[-sort <sort-id>]</code>	where <code><sort-id></code> is the identifier of one of the sort algorithm available. Possible values are (nb-occur length).
<code>[-filter <filter-id>]*</code>	where <code><filter-id></code> is the identifier of one of the filter algorithm available. Possible values are (percent-a-file top-10 percent-file length same-file).
<code>[-percent-a-file <filter-value>]</code>	where <code><filter-value></code> is the integer value for the 'percent-a-file' filter.
<code>[-percent-file <filter-value>]</code>	where <code><filter-value></code> is the integer value for the

'percent-file' filter.

[-length <filter-value>]

where <filter-value> is the integer value for the 'length' filter.

3.2. Logiscope batch

Logiscope **batch** is a tool designed to work with Logiscope in command line to:

- parse the source code files specified in a Logiscope project: i.e. “.ttp” file,
- generate reports in HTML and/or CSV format automatically.

Note that before using **batch**, a Logiscope project shall have been created:

- using Logiscope **Studio**, refer refer to Section 1 or to *Telelogic Logiscope RuleChecker & QualityChecker Getting Started* documentation,
- or using Logiscope **create**, refer to the previous section.

Once the Logiscope project is created, **batch** is ready to use.

Options

The batch command line options are the following:

batch

<ttp_file>	name of a Logiscope project.
[-tcl <tcl_file>]	name of a Tcl script to be used to generate the reports instead of the default Tcl scripts.
[-o <output_directory>]	directory where the all reports are generated.
[-external <violation_file>]*	name of the file to be added into the import project folder. This option can be repeated as many times as needed. This option is only significant for <i>RuleChecker</i> module for which the external violation importation mechanism is activated
[-nobuild]	generate reports without rebuilding the project. The project must have been built at least once previously.
[-clean]	before starting the build, the Logiscope build mechanism removes all intermediate files and empties the import project folder when the external violation importation mechanism is activated.
[-addin <addin> options]	where <i>addin</i> nis the name of the <i>addin</i> to be

	activated and options the associated options generating the reports.
<code>[-table]</code>	generate tables in predefined html reports instead of slices or charts. By default, slices or charts are generated (depending on the project type). This option is available only on Windows as on Unix there are no slices or charts, only tables are generated. This option is only significant for <i>RuleChecker</i> module
<code>[-noframe]</code>	generate reports with no left frame.
<code>[-v]</code>	display the version of the batch tool.
<code>[-h]</code>	display help and options for batch .
<code>[-err <log_err_folder>]</code>	directory where troubleshooting files batch.err and batch.out should be put. By default, messages are directed to standard output and error.

Examples of use

Considering a previously created Logiscope project named **MyProject.ttp** where:

- *RuleChecker*, *QualityChecker* and *CodeReducer* verification modules have been activated,
- the Logiscope Repository is located in the folder **MyProject/Logiscope**,

(Refer to the previous section or to the *RuleChecker & QualityChecker Getting Started* documentation to learn how creating a Logiscope project).

Executing the command on a command line or in a script:

```
batch MyProject.ttp
```

will:

- perform the parsing of all source files specified in the Logiscope project **MyProject.ttp**,
- run the standard TCL script **QualityReport.tcl** located in `<log_install_dir>/Scripts` to generate the standard *QualityChecker* HTML report named **MyProjectquality.html** in the default **MyProject/Logiscope/reports.dir** folder.
- run the standard TCL script **RuleReport.tcl** located in `<log_install_dir>/Scripts` to generate the standard *RuleChecker* HTML report named **MyProjectrule.html** in the default **MyProject/Logiscope/reports.dir** folder.
- run the standard TCL script **Reducer_report.tcl** located in `<log_install_dir>/Scripts` to generate the standard *CodeReducer* HTML reports named **testfilescomparison.html** and **testsimilarities.html** in the default **MyProject/Logiscope/reports.dir** folder.

4. Reference Guide

This section lists all CodeReducer parameters, with their values and effects.

Important note:

**Whenever a parameter is changed,
the project must be rebuilt to take it into account.**

This means that if you change a parameter, and don't rebuild the project, the displayed results will be those of the previous parameters values.

4.1. General settings

Search configuration

All project configurations are based on the same assumption:

To be even considered by the search engine, a code portion must contains at least three “Category 1” tokens, and additional tokens depending on the search precision level.

This parameter has an effect on the search algorithm's scope, as described below.

Search configuration	
Value	Effect on the search algorithm's scope
C similar functions	Scopes are associated to each function source code.
C similarities everywhere	This thorough (and thus longer) search mode bases its search on the whole source code.
Java similar packages	Scopes are associated to each package source code.
Java similar classes	Scopes are associated to each class source code.
Java similar functions	See “C similar functions”.
Java similarities everywhere	See “C similarities everywhere”.
C++ similar classes	See “Java similar classes”. <u>Note:</u> This search mode only applies to the classes definitions.
C++ similar functions	See “C similar functions”.
C++ similarities everywhere	See “C similarities everywhere”.
Ada similar packages	See “Java similar packages”.
Ada similar procedures	Scopes are associated to each procedure source code.
Ada similarities everywhere	See “C similarities everywhere”.

Search options

Precision	
Value	Effect
1 to 5 (default is 3)	The precision determines which kind of tokens the search algorithm must consider when looking for similarities. Choosing a lower or higher precision will produce a coarse-grained or fine-grained result, respectively.

Computes data for projects comparison	
Value	Effect
Checked or unchecked (default)	If checked, data necessary for project comparison will be produced when building the project.

Note about projects comparison:

- To activate the projects comparison feature, the workspace must contain two projects exactly,
- You should only check this parameter for one of the projects only, or the search will be performed twice,
- When building the project in which this parameter is checked, CodeReducer will compare both projects, and provide similarity results between projects.

% of similarities at least between files	
Value	Effect
Between 0 and 100 (default is 80)	This threshold is the percentage of similarities two files must reach to be considered similar. This parameter only has effect on the file and project comparison features, not on the “Display similarities” feature.

4.2. Display settings

Sort criteria

Sort criteria	
Value	Effect
Sort on number of similarities for sequences (default)	This sort option will order the similarities by counting the number of times the similar code portion has been found.
Sort on similarities length	This sort option orders the similarities by counting the number of tokens they contain.

Result filters

Result filters	
Value	Effect
Similarities are at least x% of one file	For a similarity to be added to the result, it must contain at least x% of tokens in at least one of the files it appears in.
Similarities are at least x% of each file	For a similarity to be added to the result, it must contain at least x% of tokens in each files it appears in.
Shows only the top 10	Restricts the number of results to the ten most relevant ones.
Similarities are at least X tokens	Similarities must be at least X tokens long to be added to the result. The higher this threshold is, the fewer similarities will be found.
Ignore similarities when limited to only one file	Discard similarities that were found within the same file.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for

this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Copyright license

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_.

Trademarks

IBM, the IBM logo, ibm.com, Telelogic, Telelogic Synergy, Telelogic Change, Telelogic DOORS, Telelogic Tau, Telelogic DocExpress, Telelogic Rhapsody, Telelogic Statemate, and Telelogic System Architect are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both, are trademarks of Telelogic, an IBM Company, in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at

www.ibm.com/legal/copytrade.html.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, FrameMaker, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

AIX and Informix are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

HP and HP-UX are registered trademarks of Hewlett-Packard Corporation.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Macrovision and FLEXnet are registered trademarks or trademarks of Macrovision Corporation.

Microsoft, Windows, Windows 2003, Windows XP, Windows Vista and/or other Microsoft products referenced herein are either trademarks or registered trademarks of

Microsoft Corporation.

Netscape and Netscape Enterprise Server are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Sun, Sun Microsystems, Solaris, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Pentium is a trademark of Intel Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.