**Logiscope Code Reducer**
**Finding code similarities**

Before using this information, be sure to read the general information under “Notices” section, on page **43**.

.

# Table of Contents

# About this manual

## *Audience*

This reference manual in intended for **Kalimetrix® 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*

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

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

Chapter 3 details each parameter and its impact on the results.

# Contacting Kalimetrix Software Support

If the self-help resources have not provided a resolution to your problem, you can contact Kalimetirx Software Support for assistance in resolving product issues.

**Note.** If you are a heritage Kalimetrix customer, you can go to http://support.Kalimetrix.com/toolbar and download the Kalimetrix Kalimetrix Software Support browser toolbar. This toolbar helps simplify the transition to the Kalimetrix Kalimetrix product online resources. Also, a single reference site for all Kalimetrix Kalimetrix support resources is located at: http://www.ibm.com/software/rational/support/Kalimetrix/

## *Prequisites*

To submit your problem to Kalimetrix Software Support, you must have an active Passport Advantage® software maintenance agreement. Passport Advantage is the Kalimetrix comprehensive software licensing and software maintenance (product upgrades and technical support) offering.

To submit your problem online (from the Kalimetrix Web site) to Kalimetrix Software Support, you must additionally:

- Be a registered user on the Kalimetrix Software Support Web site. For details about registering, go to
  http://support.kalimetrix.com
- Be listed as an authorized caller in the service request tool.

## *Submitting problems*

To submit your problem to Kalimetrix Software Support:

1) Determine the business impact of your problem. When you report a problem to Kalimetrix, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem that you are reporting.

   Use the following table to determine the severity level.

| Severity | Description |
|---|---|
| **1** | The problem has a *critical* business impact. You are unable to use the program, resulting in a critical impact on operation. This condition requires an immediate solution. |
| **2** | The problem has a *significant* business impact. The program is usable, but it is severely limited |
| **3** | The problem has a *some* business impact. The program is usable, but less significant features (not critical to operation) are unavailable. |
| **4** | The problem has a *minimal* business impact. The problem causes little impact on operations or a reasonable circumvention to the problem was implemented. |

2) Describe your problem and gather background information, When describing a problem to IBM, be as specific as possible. Include all relevant background information so that Kalimetrix Software Support specialists 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?

     To determine the exact product name and version, use the option applicable to you:

     - Start the Kalimetrix Logiscope and check Help About

- What is your operating system and version number (including any service packs or patches)?
- Do you have logs, traces, and messages that are related to the problem symptoms?
- Can you recreate the problem? If so, what steps do you perform to recreate the problem?
- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, or other system components?
- Are you currently using a workaround for the problem? If so, be prepared to describe the workaround when you report the problem.

3) Submit your problem to Kalimetrix Software Support. You can submit your problem to Kalimetrix Software Support in the following ways:

- **Online**: Go to the Kalimetrix Software Support Web site at http://support.kalimetrix.com

If the problem you submit is for a software defect or for missing or inaccurate documentation, Kalimetrix Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, Kalimetrix Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. Kalimetrix publishes resolved APARs on the Kalimetrix Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolutions.

# 1. Basic concepts

## *1.1. CodeReducer typical Use Cases*

**Real-life examples ... from software life-cycle**

When managing projects involving abundant source code, there are many 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 Logiscope CodeReducer.**

## 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.**

## *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 Engine*

*CodeReducer* uses a search engine based on an internal list of lexical items or "tokens" of the supported programming language.

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

## Language Tokens

A token is a structural element of the programming language: e.g. control structure, structure and instructions delimiters, assignments, operators.

**Important note**:

> Since *CodeReducer* is a similarity search tool, and not a difference tool, operands, identifiers (variable, function names, etc.) are not considered as tokens to evaluate code similarity.

For all languages, tokens are broken down into categories associating tokens to degrees of similarity. The higher the degree, the more tokens will be considered when looking for similarities. A given similarity degree considers all tokens for this degree and lower ones.

Three degrees are made available to the user:

- **"Minimum" degree:**

  The following types of tokens are considered :
  - component delimiters: functions, classes, packages, ...
  - control structures : if, else, loops, switch, ...,
  - blocks delimiters: begin, end, {, },
  - assignments.

  The "Minimum" degree allows the detection of similar code structure with same number of variables assignments in code blocks but the associated expressions can differ.

- **"Medium" Degree:**

  In addition to the tokens of the "Minimum" degree, the operators: e.g.: +, -, *, ..., are considered.

  The "Medium" degree adds the detection of similar expressions.
  But other instructions (e.g.: function call) can appear in the similar code.

- **"Maximum" Degree:**

  In addition to the tokens of the "Medium" degree, the parenthesis and instructions terminators are considered.
  The "Maximum" degree adds the detection of same number of instructions with similar contents.

**Important note:**

> Always remember that increasing the similarity degree can reveal similarities that were not visible with a lower degree, because the set of tokens associated to different similarity degrees are not subsets one of another.

**CodeReducer_limitations:**

CodeReducer's engine searches for executable code, looking for programming structural elements. If a code contains no such elements, no similarities can be found.
For example, similar C++ class search only works if classes contain inline methods, and will produce incorrect results if some classes don't contain executable code (i.e containing language tokens).

# 2. Getting started with 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 standard 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 a create a dedicated directory to store these data: e.g. a folder named: `LogiscopeProjects`**.**

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 rework, 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. The source files for this Use Case are located in the `samples\C\Mastermind` folder of your Logiscope installation directory.

## Step 1: Starting a Logiscope Studio Session

To begin a Logiscope *Studio* session:

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

- launch the **vcs** binary .

On Windows:

- click the **Start** button and select the **Kalimetrix Logiscope 6.6** item in the **Kalimetrix** Programs Group.

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

For more details on Logiscope **Studio**, please refer to *Kalimetrix Logiscope Studio Reference Manual.*

## Step 2: Creating a new Logiscope project

First of all, you shall create a Logiscope project which mainly consists in the list of source files to be analyzed.

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

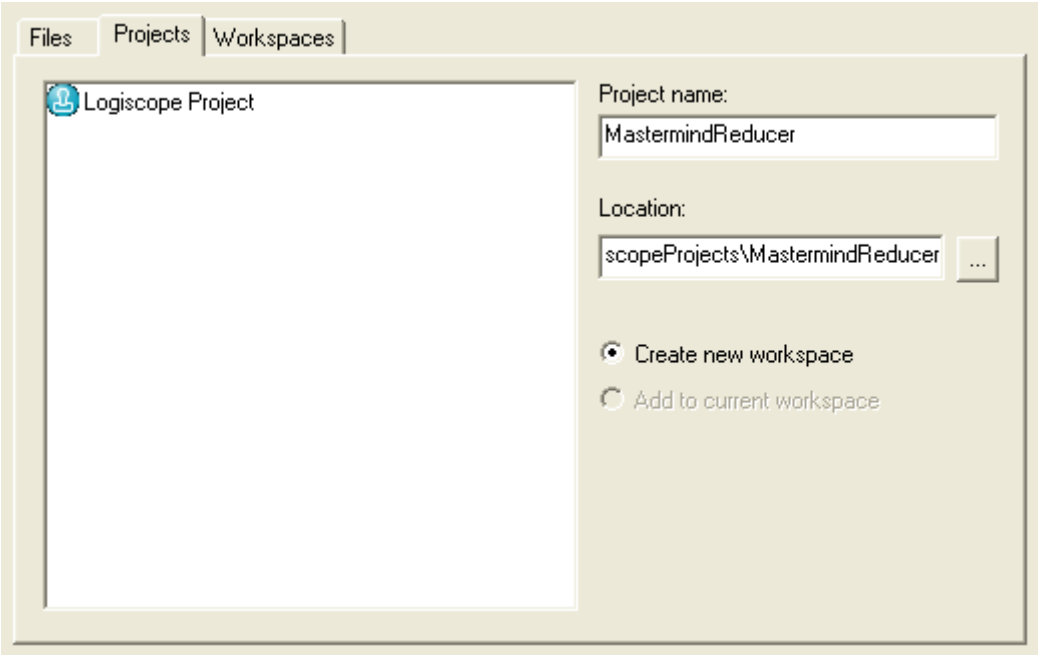
The project creation wizard is invoked, and guides you through the process of creating a new Logiscope project.

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 "`MastermindReducer`".

- 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 may have prepared as recommended in the Before You Start section: LogiscopeProjects

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



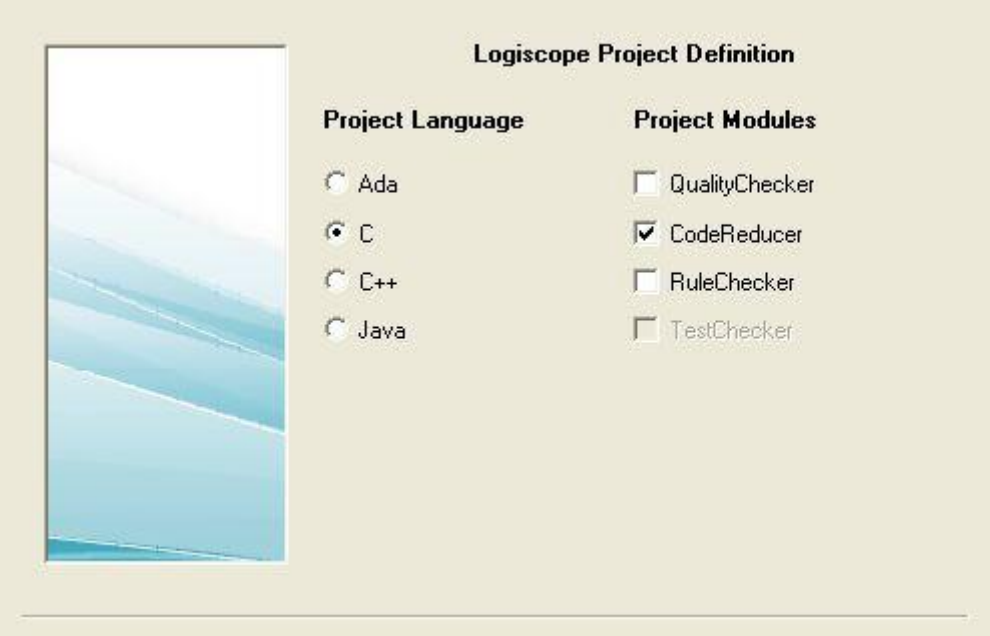

- Validate the project name and location by pressing "OK".

The next screen lets you select the programming language of the source files and the Logiscope verification modules associated to the project.

- 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 >”.

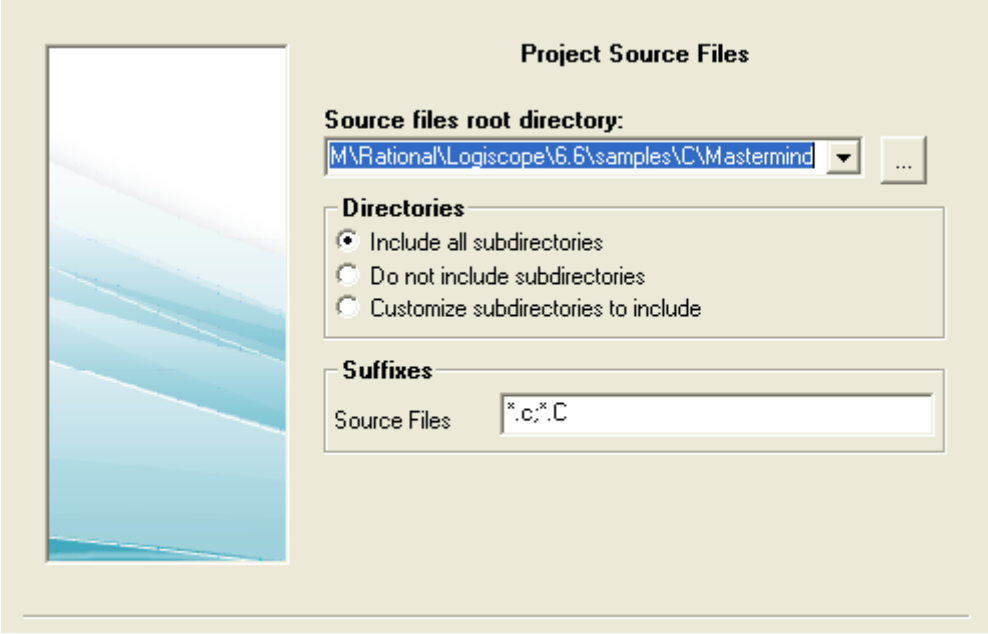
- For more details on the *QualityChecker* and *RuleChecker* modules, please refer to *Kalimetrix Logiscope RuleChecker and QualityChecker Getting Started.*
- For more details on the *TestChecker* module, please refer to *Kalimetrix Logiscope - TestChecker Getting Started.*

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

- **Source files root directory:** Browse to select the directory where the "Mastermind" source files are located: i.e. in the `samples\C\Mastermind` folder in the standard Logiscope installation directory: e.g. `C:\Program Files\IBM\Rational\Logiscope\6.6\`
- 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.

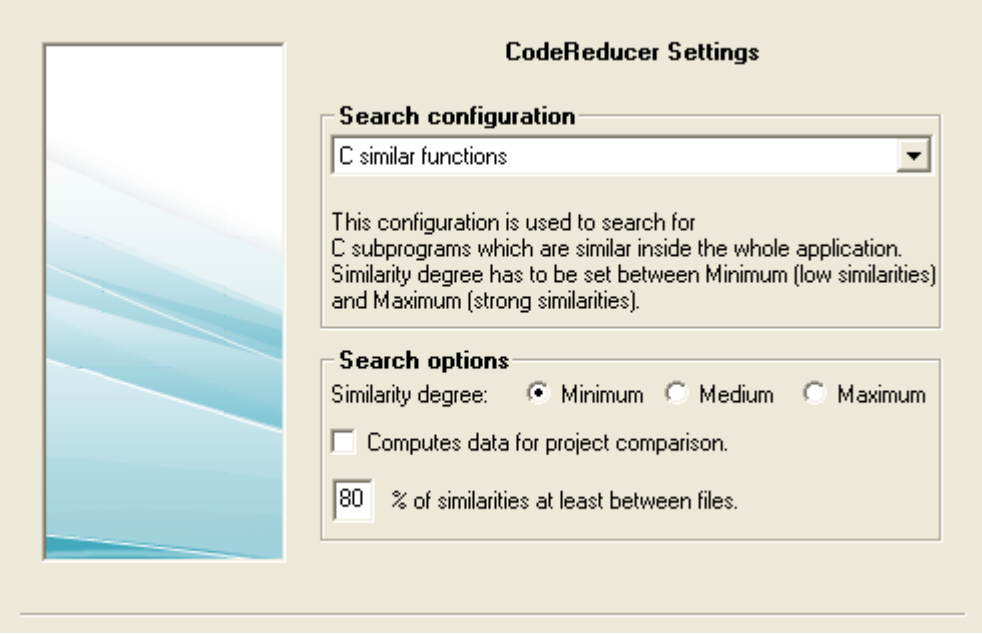



- Validate the project name and location by pressing "Next >".

The following dialog box allows you to select some key settings of *CodeReducer*.

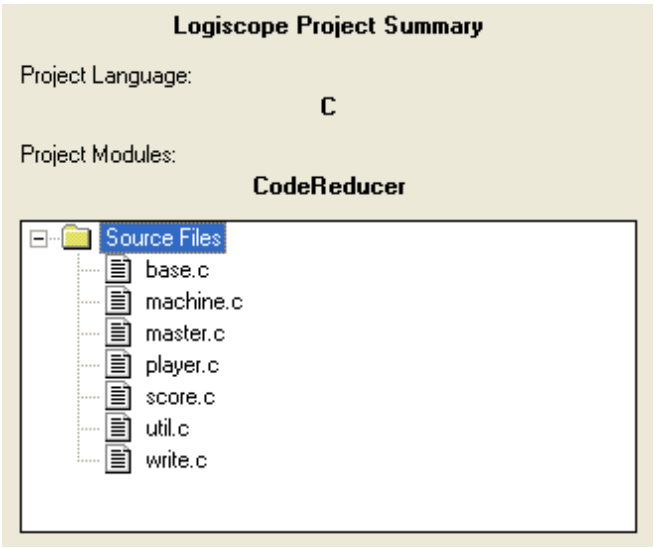
**<u>These settings significantly impact the nature and number of the similarities found.</u>**

- **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 **similarity degree:** i.e. the set of tokens that will be considered for identifying similarities in the code (cf. §1.3).




- For this first use case, keep all default settings. Just click "Next >".

  The final screen summarizes the project main attributes.
- Expand the "Source Files" folder .



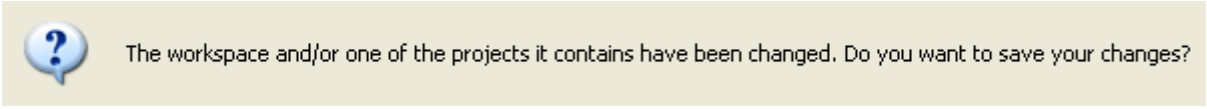

- Click on "Finish" to achieve the Logiscope project creation process.

## Step 3: "Building" 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**" or click on the corresponding icon.

In case a message window is displayed related to changing the workspace ...




select "Yes"

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 analyzed and associated results generated and loaded.

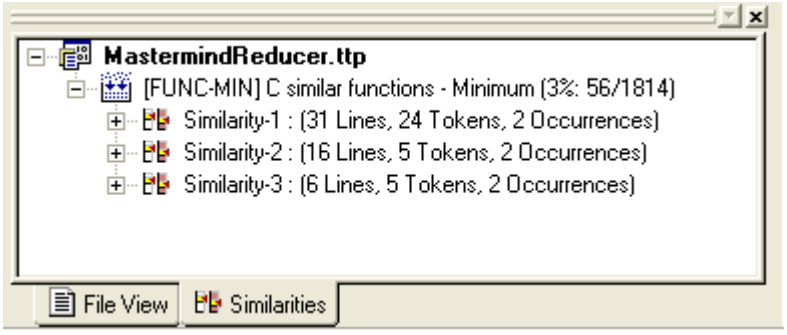
## Step 4: Analyse the code similarities found

After the build is completed, you are ready to analyse and state on the similarities found by *CodeReducer* for this scenario i.e. the search configuration and the similarity degree.

*CodeReducer* results icons are now available :

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

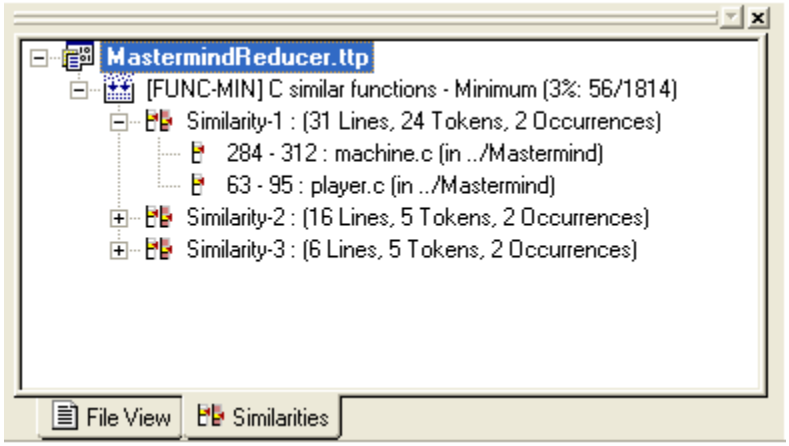
The results are displayed as a dynamic tree, which nodes are associated to all similarities found in the project for this given scenario : i.e. FUNC: "C similar functions" configuration and MIN: "Minimum" similarity degree.




For this scenario, 3 similarities have been found, hence producing one node per similarity in the tree under the corresponding scenario node.

By default, the similarities found are sorted according to the descending total number of lines of code found in the occurrences of similar code. The sort criterion can be changed using the Project Settings command.

- Expand the node “Similarity-1” .




Each similarity node contains leaves associated to the location of the occurrences of similarities. The “similarity-1” :

- is 31 lines long (this is an average value, as the occurrences may not have the exact same number of lines),
- is 24 tokens long,
- has 2 occurrences, located in
  - the file `machine.c` from line 284 to line 312, and
  - the file `player.c` from line 63 to line 95.

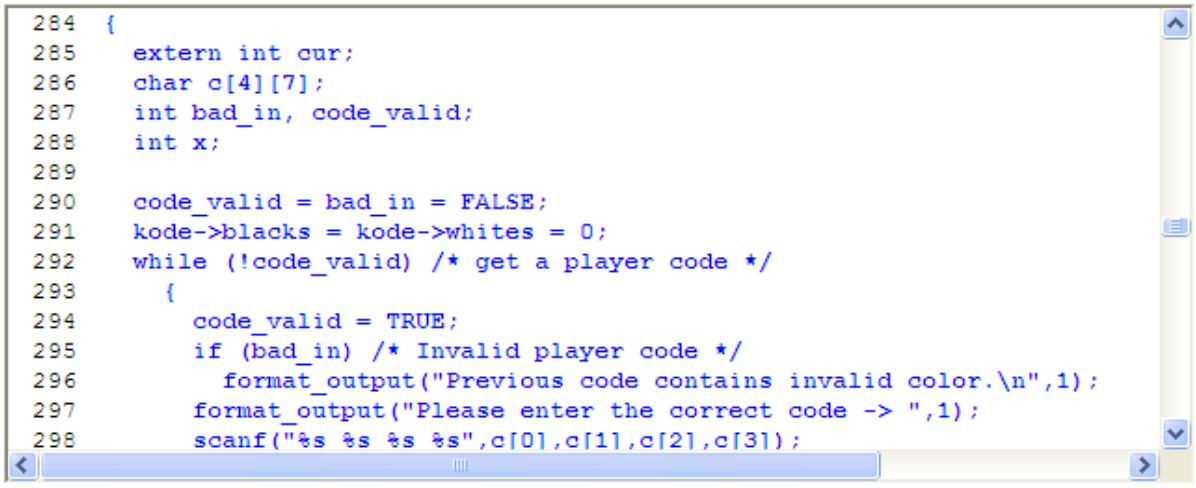
- In the Similarity Tree tab, double-click on the “Similarity-1” node.

A new window is displayed “full-screen” showing the two files where similar occurrences of code were found in two distinct panes, one for each occurrence involved. The similar code is highlighted in blue.

Here two observations can be done:

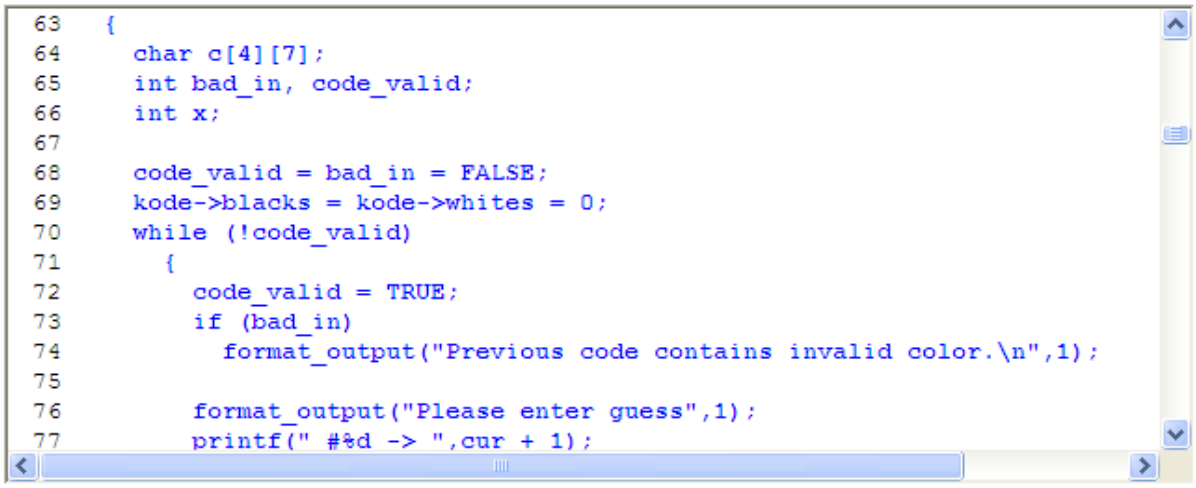
- The two similar codes have different comments and line count. This is normal, since similarities are not based on presentation attributes.
- Apart from the presentation attributes, the code is very similar, meaning a factorization is possible.
  Factorizing the code between these two functions means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

C:/Program Files/IBM/Rational/Logiscope/6.6/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        scanf("%s %s %s %s",c[0],c[1],c[2],c[3]);
```

C:/Program Files/IBM/Rational/Logiscope/6.6/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 -> ",cur + 1);
```

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

  This simple example shows how *CodeReducer* make easy and fast looking for similarities in a source code in order to find factorizable code, and improve the maintainability of the project source files.

## Step 5: Changing the CodeReducer Settings

The previous results have been obtained with the default settings of *CodeReducer*. They have been tuned to provide quick results. However, the most impressive results on this application are obtained with an other search configuration and a different similarity degree.

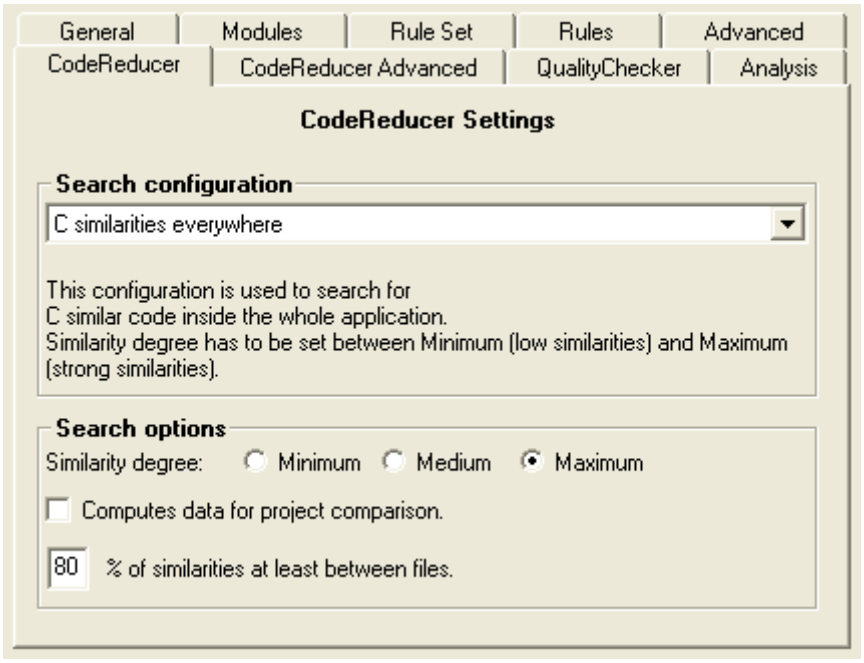
- Click on the "**Project – Settings...** " menu item.

The Logiscope Settings dialog box is open.

- Select the "CodeReducer "tab

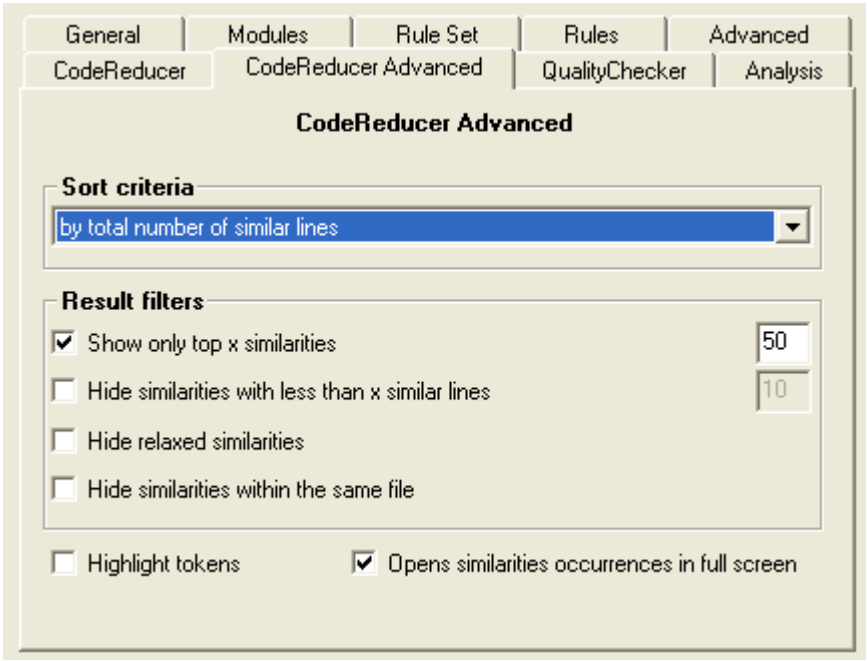
You can now change the similarities search configuration to find different types of similar code in the application.

- In the "**Search configuration**" sections, select the item "C similarities everywhere",
- In the "**Search options**" section, select the similarity degree: **"**Maximum**"** to get very similar code.

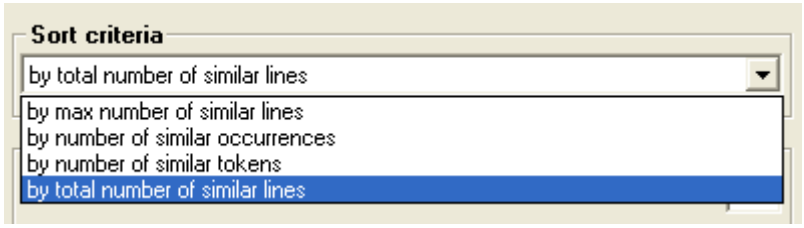



- Select now the "CodeReducer Advanced" tab.

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



- The **Sort criteria** section allows selecting the way the similarities are sorted when displayed. 4 choices are available. They are detailed in section 4.2.




- In the **Result filters**:
  - In order to limit this first investigation to the most significant similarities found, put 10 in area associated to the already checked filter "Show only the top x similarities".


  - Check "Highlight tokens" to easily identifier similar tokens in various occurrences of code when displaying the similarities.


- click on the "**OK**" button to save the new *CodeReducer* settings.

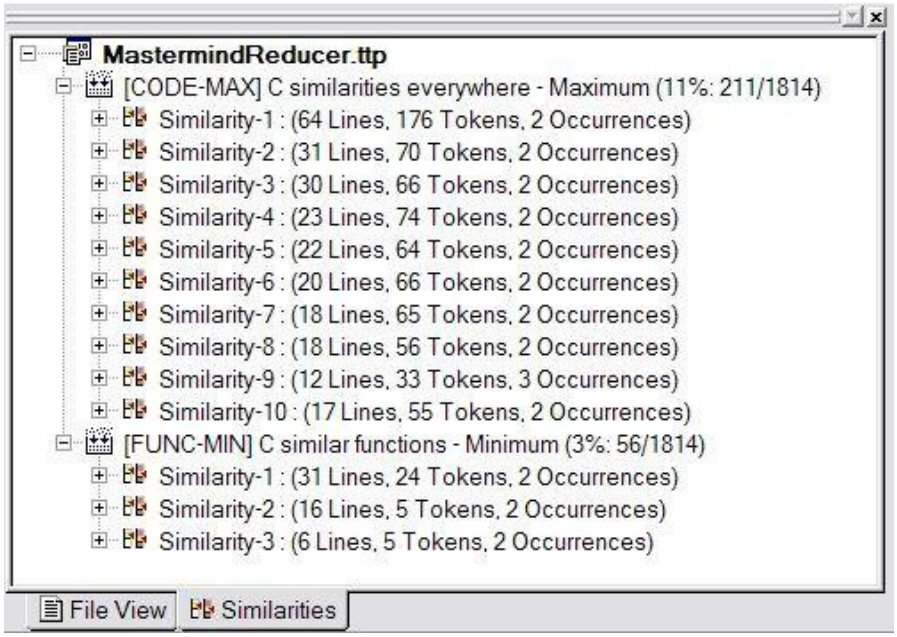
As the project settings have been changed, the project shall be (re-)built to take into account this new configuration.

- Simply use the appropriate menu item: "**Project – Build**" or click on the corresponding icon.

  Caution: Do **not** click on "**Project – Rebuild All**", you would loose the results from your previous configuration.

- As soon as the build process is finished, refresh the "Similarities" tab by clicking on the "Display Code Similarities Tree" icon or activate the **"Browse – Reducer - Similarities Tree"** menu item.

According to the filters set previously, the list of the Top 10 most significant similarities found in the overall project source code is displayed. Some are much more impressive than those found for the "C similar functions" configuration in the previous step.




Indeed, the Similarity-1 corresponds to a section of 64 similar lines containing a sequence of 176 identical tokens that occurs 2 times in the overall Mastermind application. Factorizing this section of code would significantly improve the level of maintainability of the source code.

- Expand the Similarity-1 node to locate these 2 occurrences.


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

- Double click on the node Similarity-1 to display the corresponding source code.

First of all, note that all similar tokens are now highlighted in yellow. You can also use the scrolling of the first window to make all three source code windows scroll synchronously.

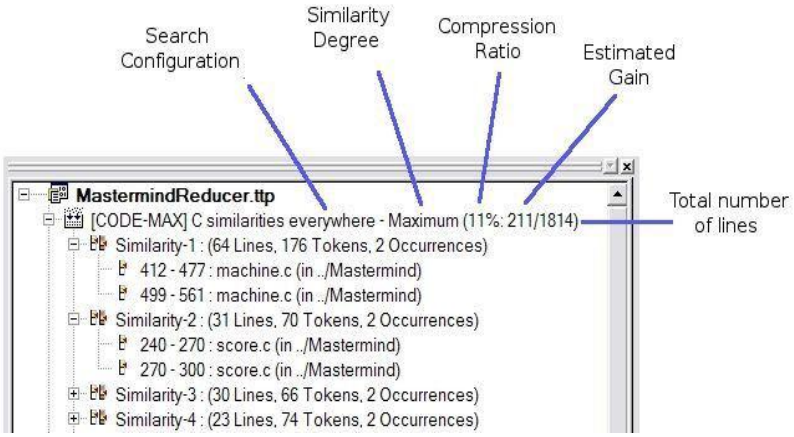
This section of code shows that such a similarity may come from a large “Copy-Paste” action ... Which is definitively a poor coding practice regarding software maintainability requirements.

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

## Step 6: Quantifying the Similar Code: the Compression Ratio

You may have noticed that the similarity tree contains nodes with some figures. Indeed, each *CodeReducer* "Build" node reminds the corresponding scenario: Search Configuration and Similarity Degree but also indicates the Compression Ratio estimated by *CodeReducer*.

The Compression Ratio is the percentage of code that would be removed from the project source files by factorizing the similarities found for the given scenario. It is the ratio between the estimated gain of the similarities found and the total number of lines in the project source files where the estimated gain is the number of lines of code belonging to all occurrences but one (the first) of all the similarities displayed.
When factorizing the code, you will keep one instance / occurrence of the code, and remove the others.




In the example above, *CodeReducer* estimates that 211 lines can be removed from the 1814 lines code in the project source files; producing a compression ratio of 11% for that given scenario.

For "Similarity-1", the code of the second and third occurrences could be factorized with the code of the first occurrence. So, instead of having 2 times 64 lines, you should "compress" / factorize the code to only 1 time 64 lines. Thus, the Estimated Gain for just "Similarity-1" would be of 64 lines.

Finally, note that the Compression Ratio only takes into account similarities that are displayed in the corresponding tree. So the value depends on the results filters that may have been set. This is the case for our example ... meaning that, the Compression Ration would have been greater if not limited to only the "Top 10" similarities.

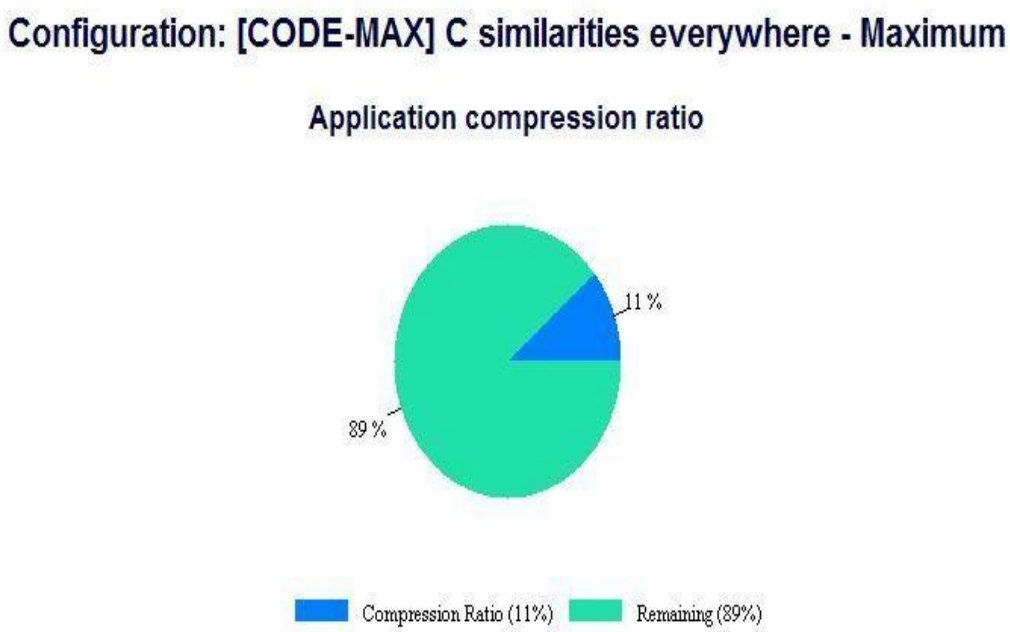
For more details on source code metrics provided by *CodeReducer*, please refer to §4.4 "Source code metrics for *QualityChecker*".

## Step 7: Generating assessment report

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

An HTML format report is automatically generated, displayed and saved in the `reports.dir` folder in the Logiscope project Repository. You can browse within to get access to all results.


- click on the first configuration to access to the corresponding similarities via the Compression ratio summary pie.



## 2.2. Use Case 2: Searching a Reference Code

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

Looking for a reference source code is fairly easy. Suppose you have to modify the code in file `player.c`, starting at line 35: the 2 nested for loops.

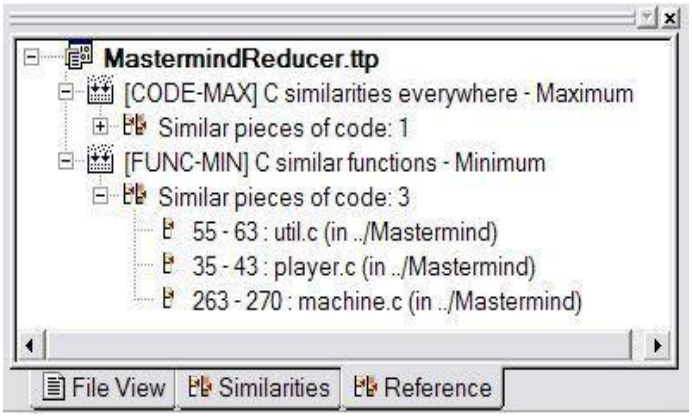
- Double click on the file in the File View tree
- Simply select the reference code in file "player.c", from line 35 to 43.

```
      }
  if (guesses[cur].blacks != 4)
    /* if not game_won, checks white pegs*/
    {
      for (x = 0; x < 4; x++)
        for (y = 0; y < 4; y++)
          if (guesses[cur].pegs[x].color == code.pegs[y].color &&
              !code.pegs[y].used && !guesses[cur].pegs[x].used)
            {
              code.pegs[y].used = guesses[cur].pegs[x].used = WHITE;
              guesses[cur].whites++;
              printf("w");
            }
    }
  else
    game_won = TRUE;
```

- Click on the "Searching code similar to selected one" icon, or choose the "**Browse – Reducer – Searching Code**" menu item.

  In our example, three similar pieces of code have been found based on the scenario "[FUNC-MIN] C similar functions - Minimum", hence producing three nodes in the associated subtree.

- Expand the node on the "Similar pieces of code: 3"



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

Reference code used for the duplication search:

```
1         for (x = 0; x < 4; x++)
2           for (y = 0; y < 4; y++)
3             if (guesses[cur].pegs[x].color == code.pegs[y].color &&
4                 !code.pegs[y].used && !guesses[cur].pegs[x].used)
5               {
6                 code.pegs[y].used = guesses[cur].pegs[x].used = WHITE;
7                 guesses[cur].whites++;
8                 printf("w");
```

C:/Program Files/IBM/Rational/Logiscope/6.6/samples/C/Mastermind/util.c

```
55            for (x = 0; x < 4; x++)
56              for (y = 0; y < 4; y++)
57                if (guesses[cur].pegs[x].color == dummy[z].pegs[y].color &&
58                    !dummy[z].pegs[y].used && !guesses[cur].pegs[x].used)
59                  {
60                    dummy[z].pegs[y].used = guesses[cur].pegs[x].used =
61                      WHITE;
62                    dummy[z].whites++;
```

C:/Program Files/IBM/Rational/Logiscope/6.6/samples/C/Mastermind/player.c

```
35          for (x = 0; x < 4; x++)
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/IBM/Rational/Logiscope/6.6/samples/C/Mastermind/machine.c

```
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               {
268                 code.pegs[y].used = guesses[cur].pegs[x].used = WHITE;
269                 guesses[cur].whites++;
270               }
```

To access to each one of the similar code portions, just click on its associated node in the result tree.

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

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

## 2.3. Use Case 3: Tracking changes between two variants of a project

A variant of the project has been released for a second customer. A third one is now also asking for the development of its own special version. This is a start for a product line...

However, it is no more time to have a project based development policy but a product. So, to start this third version, you shall take the best of the two previous variants expecting they are not as different as both development teams say.

In this Use Case, you will use both the "`Mastermind`" and "`Mastermind_Bis`" C samples provided with the standard Logiscope distribution, and compare these two projects using *CodeReducer*.

The `Mastermind_Bis` sample is a copy of the `Mastermind` project, where some modifications have been applied by the new development team:

- The file `machine.c` has been renamed `mymachine.c`. And the first function has been moved to the end of the file.
- In file `master.c`, a bug has been fixed removing the useless statement `val= x-1;` and the preceding comment.
- The function `help()` in the file `player.c` has been moved in a newly dedicated file `help.c`.
- In most of the files, some special comments have been added in function headers to ensure traceability between requirements and source code.
- Only the file `base.c` has not been changed at all.

### Step 1: Creating 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 `MastermindReducer` workspace: i.e. a file with an extension "**.ttw**", which contains / refers to the `MastermindReducer` project: i.e. a file with an extension "**.ttp**".

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

- Start a new session as in the Step 1 of the previous section.
- Open the `MastermindReducer` workspace created while performing the Use Case 1 by using either the "**File > Open ...**" command or the "**File > Recent Workspaces ...**" command.

Once the workspace is open:

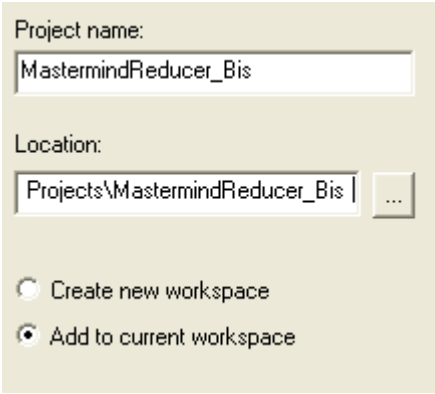
- In the **File** menu, select the **New**... command.

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 `MastermindReducer_Bis`.
- Then, select its **Location:** e.g. `C:\LogiscopeProjects`

- But, in this context, select 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".

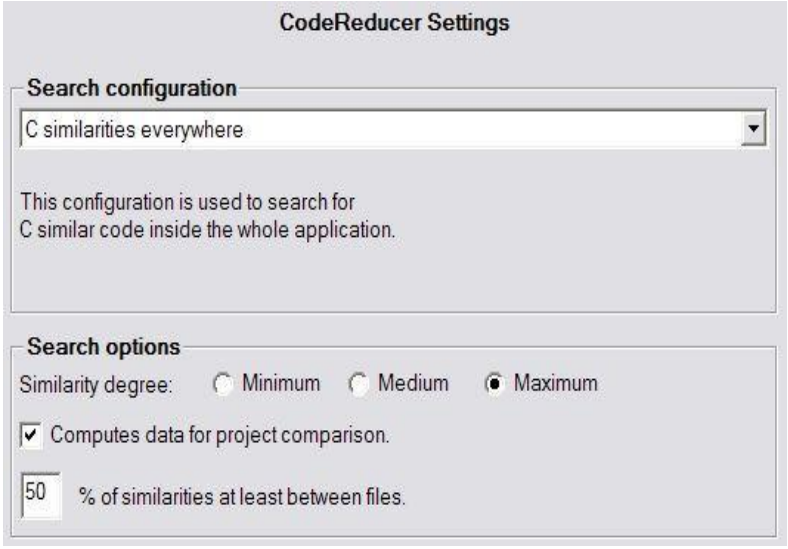
As explained in section 2.1:

- In the **Logiscope Project Definition** dialog box, select C as the Project Language and CodeReducer as the project Module.
- Validate the project definition by pressing "Next >".
- In the **Project Source Files dialog box,** browse to select the **Source files root directory:** i.e. the directory where the version Bis of the "Mastermind" source files are located: i.e. the `\samples\C\Mastermind_Bis` folder in the standard Logiscope installation directory.


- Validate with "Next >".

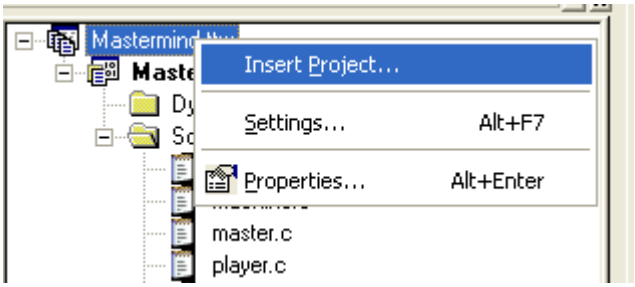
In the **CodeReducer Settings** dialog box:

- **Search configuration**: Select "C similarities everywhere" as you will search for similarities every where in the code ... not only similar functions as seen when starting 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 projects.
- Select the Similarity Degree: **Maximum.**
- Change the value associated to the **% of similarities at least between files** option from 80 to 50.




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

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




## Step 2: "Building" 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: Comparing the project variants

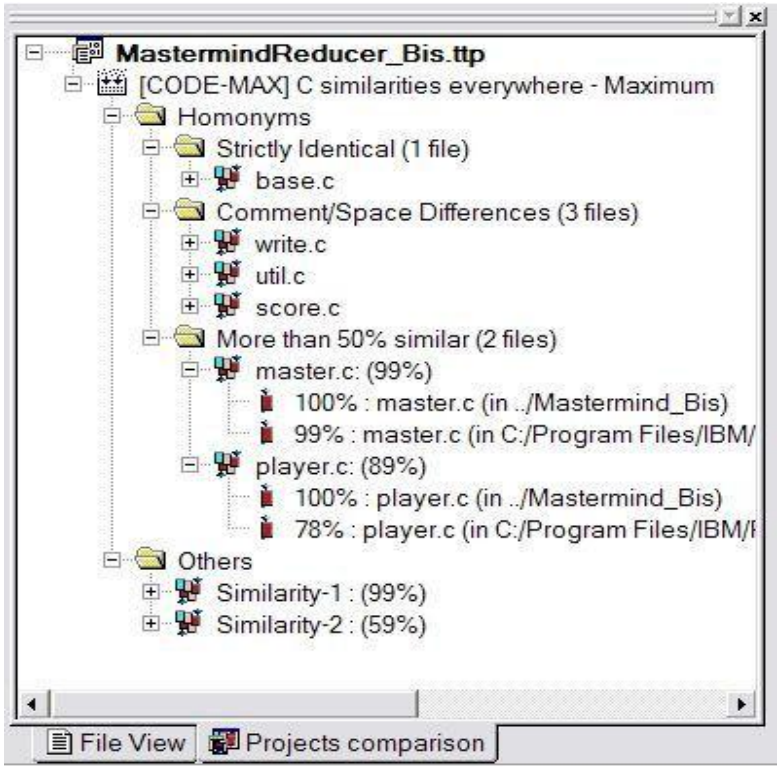
After the build is completed, the projects comparison results can be displayed and interpreted.

- Click on the "**Displays CodeReducer Project Comparison Tree**" icon, or choose the "**Browse – Reducer – Projects Comparison**" menu item.

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

- Homonyms: 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,
- Others: these are files that do not have the same name, but contain similar code.

- Expand first the Homonyms subtree :

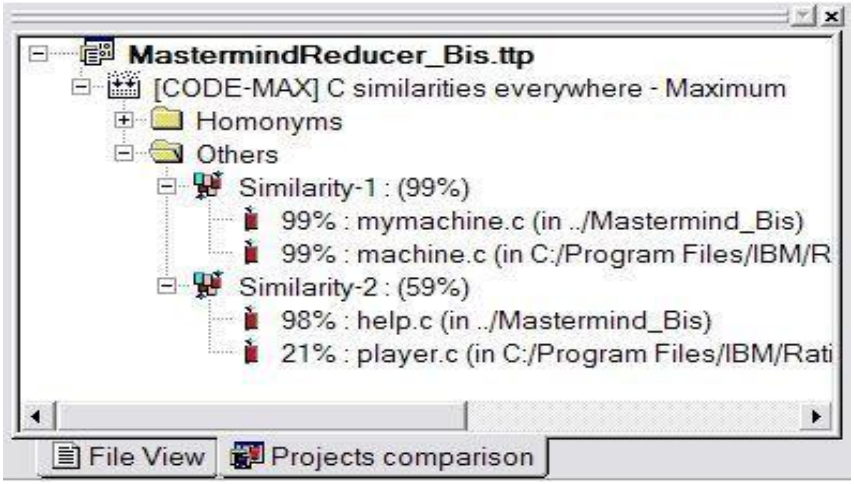



As shown in the above result window:

- the two instances of the file `base.c` are identical in both project,
- the files `write.c, util.c,` and `score.c` only differ by comments and/or indentation; the code is identical.
- the two instances of the file `master.c` are very close, the instance in `Mastermind_Bis` is fully included (100%) in the instance in `Mastermind` whereas 99% of the latter is included in the former, meaning it contains additional code.

- the two instances of the files `player.c` are much different, the instance in `Mastermind_Bis` is again fully included (100%) in the instance in `Mastermind` whereas 78% of the latter is included in the former. Indeed, it contains an additional function.

- Now expand the Others subtree:




As shown in the above result window:

- the files `machine.c` and `mymachine.c` contains nearly fully similar functional code,
- the code of the file `help.c` in `Mastermind_Bis` is almost fully included in the `player.c` in `Mastermind`.

Even if a "difference tool" would have found that most of the files in pairs are different, CodeReducer helped you reach the conclusion that they contains similar functional code.

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: Generating 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®

# Rational Logiscope Projects Comparison Report

Date: 10 Apr 2009

This document contains information concerning the comprison of the project C:\Program Files\IBM\Rational\Logiscope\6.6\samples\C\LogiscopeProjects\MastermindReducer.ttp and the project C:\Program Files\IBM\Rational\Logiscope\6.6 \samples\C\LogiscopeProjects\MastermindReducer_Bis.ttp made with Logiscope CodeReducer which is part of **IBM® Rational® Logiscope**.

The following build configurations are available:

- [CODE-MAX] C similarities everywhere - Maximum

...

- On the left frame, click on "**More than 50% similar**" menu item.

| File | Common | Different lines |
|---|---|---|
| File: master.c | | |
| ../Mastermind_Bis/master.c | 100% | - |
| C:/Program Files/IBM/Rational/Logiscope/6.6/samples/C/Mastermind/master.c | 99% | 55 |
| File: player.c | | |
| ../Mastermind_Bis/player.c | 100% | - |
| C:/Program Files/IBM/Rational/Logiscope/6.6/samples/C/Mastermind/player.c | 78% | 98 to 145 |

The list of corresponding files is now displayed with the location of the differences found:

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

## 2.4. Use Case 4: Excluding code from similarities search

An older project has been analyzed with CodeReducer, which detected that several similarities within the source code could be eliminated by factorization.

However, some of the source code detected as similar is part of a legacy code you don't want to modify.
You need to notify CodeReducer that this particular source code shouldn't be part of any similarity.

In this Use Case, you will use the "`MastermindReducer`" workspace created in Use Case 1 and modified in the following Use Cases.

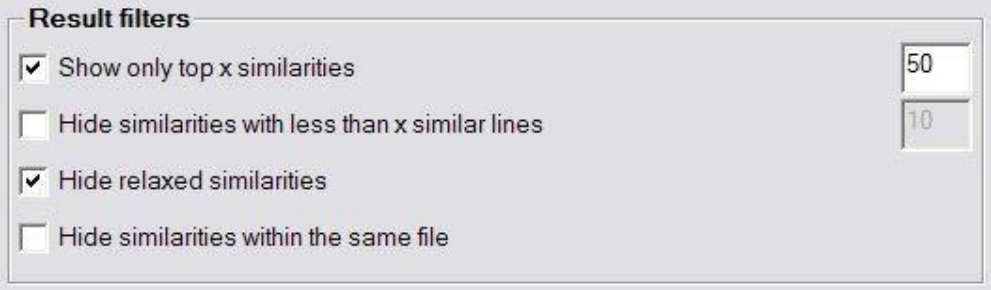
### Step 1: Opening an existing Logiscope Workspace.

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

- Start a new session as in the Step 1 of the previous section.
- Open the `MastermindReducer` workspace by using either the "**File > Open ...**" command or the "**File > Recent Workspaces ...**" command.

In the **Project Settings** dialog box, open the « CodeReducer Advanced » tab:

- Check the **Hide relaxed similarities** option.

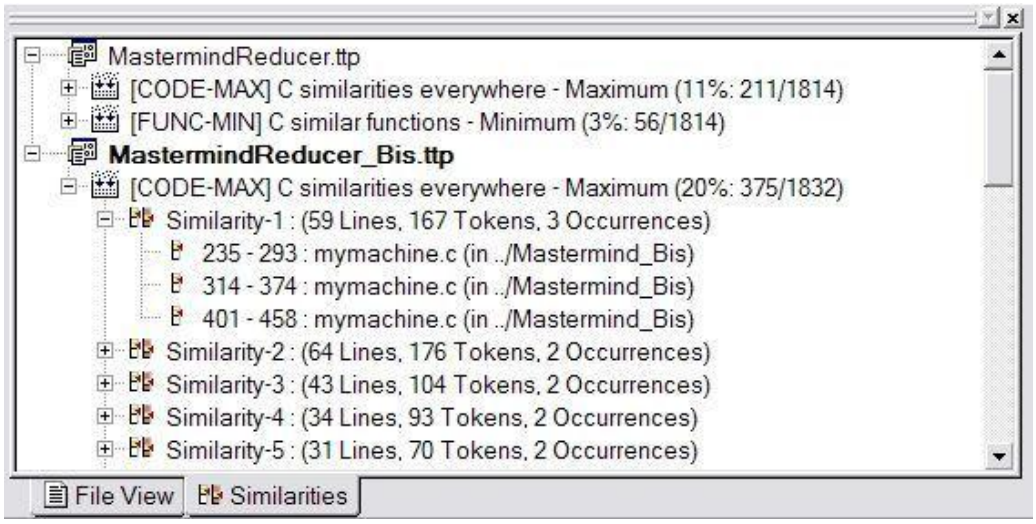


## Step 2: Relaxing a portion of code.

Before deciding what code to relax, click on the « Displays CodeReducer Similarities Tree » icon.

The list of similarities is displayed as a tree. Locate the « [CODE-MAX] C similarities everywhere » for the `MastermindReducer_bis` project. Expand the first similarity, as shown below.

You will notice that this similarity contains three occurrences, all in the `mymachine.c` file. The average size of this similarity is 59 lines.




Suppose the first portion of the code belongs to a legacy function that has to be ignored in the factorization process.

All you have to do is open the `mymachine.c` file, and surround the code to relax with the appropriate relaxation comments, as shown below.

```
  if (check) /* get the player code before the search */
    get_code_mac(&code);
  cur = 0;
  make_code(&guesses[cur]);
//  {{RELAX<CODEREDUCER> justification
  set_dummy();
  if (!check)
    skipline(1);
  if (cur < 5)
    format_output(" ",0);
  /* displays the computer guess */
  printf("    Guess #%d:   %-8s %-8s %-8s %-8s\n\n",cur + 1,
         col[dig[guesses[cur].pegs[0].color]],
         col[dig[guesses[cur].pegs[1].color]],
         col[dig[guesses[cur].pegs[2].color]],
         col[dig[guesses[cur].pegs[3].color]]);
//  }}RELAX<CODEREDUCER>
  u = -1;
```

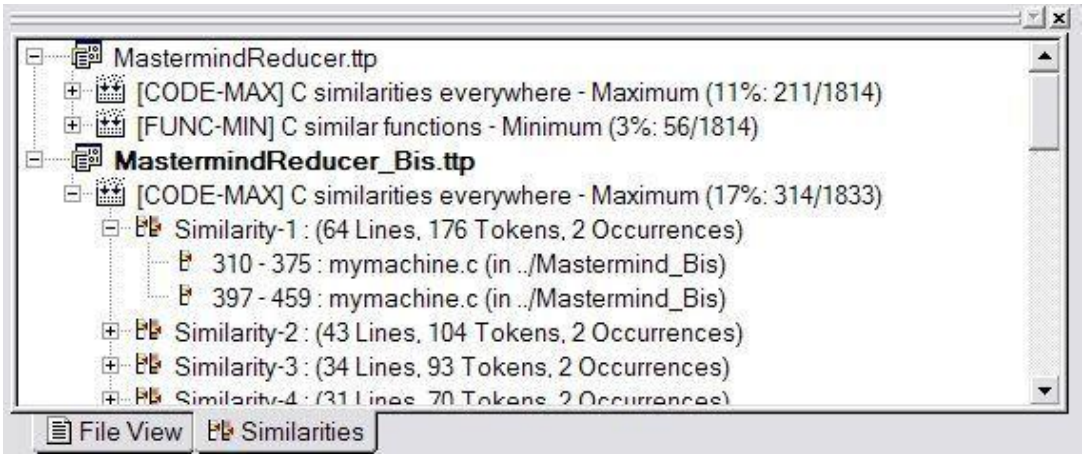
## Step 3: Relaxed similarities results.

Once the project has been rebuilt, click on the « Displays CodeReducer Similarities Tree » icon.

Locate the « [CODE-MAX] C similarities everywhere » for the `MastermindReducer_bis` project, and expand its first similarity.

Several things can be noticed:

- The similarity now only contains two occurrences. The one associated to the relaxed code does not belong to this similarity anymore.
  In fact, the relaxed code is now absent from all similarities.
- The two remaining occurrences now span more source code than before the legacy code was relaxed.
  This is due to the fact that CodeReducer found that the remaining occurrences were obviously still similar, but that some additional tokens (that did not appear in the now relaxed legacy code) could be added to the similarity, producing a bigger similarity with an average size of 64 lines (as opposed to 59 lines before relaxation of the legacy code).




You can now factorize the code based on CodeReducer results, knowing that your legacy code will not be impacted by this operation.

# 3. Command Line Mode

## *3.1. Logiscope create*

Logiscope projects: i.e. "**.ttp**" file are usually built using Logiscope **Studio** as described in the previous chapter.

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

## From the Command Line

When started from a standard command line, the **create** tool creates a new project file with the information provided on the command line. See the 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 -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 *CodeReducer* verification module will be activated.

## Within a Makefile

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 Logiscope **create** options are the following:

create -reducer

| | |
|---|---|
| `<ttp_file>` | name of the Logiscope project to be created (with the ".ttp" extension).<br>Path has to be absolute if the option -- is used. |
| `[-lang Ada\|C\|C++\|Java\|]` | the language of the source files to be analysed. |
| `[-root <directory>]` | where <directory> is the starting point of the source search.<br>Default is the current directory.<br>This option is exclusive with -list option. |
| `[-recurse]` | if present the source file search is done recursively in sub-folders. |

| | |
|---|---|
| `[-list <list_file>]` | where <list_file> 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 -root option. |
| `[-repository <directory>]` | where <directory> is the name of the directory where Logiscope internal files will be stored. |
| `[-source <suffixes>]` | where <suffixes> is the list of accepted suffixes for source file to be placed in project folder "Source Files" |
| `[-no_compilation]` | avoid compiling the files if the -- option is used |
| `[--]` | when used in a makefile, introduces the compilation command with its arguments. |
| `[-similarity_degree <value>]` | where <value> is the desired similarity degree for similarities search. Accepted value are MIN, MED, MAX. |
| `[-percent <value>]` | where <value> is the desired percentage of similarities to retain files.<br>Accepted value is an integer between 0 and 100. |
| `[-multi]` | if present, project comparison will be activated when the current project is part of a workspace containing two projects. |
| `[-config <config-id>]` | where <config-id> is the identifier of one of the search configuration available for the current language.<br>Possible values are:<br>- C: (c-function \| c-everywhere )<br>- C++: (cpp-class \| cpp-function \| cpp-everywhere )<br>- Java: (java-class \| java-package \| java-function \| java-everywhere )<br>- Ada: (ada-package \| ada-function \| ada-everywhere ) |
| `[-sort <sort-id>]` | where <sort-id> is the identifier of one of the sort algorithm available. Possible values are (nb-occur \| length). |
| `[-filter <filter-id>]*` | where <filter-id> is the identifier of one of the filter algorithm available.<br>Possible values are (top-x \| nb-lines \| relax \| same-file ). |
| `[-top-x <filter-value>]` | where <filter-value> is the integer value for the 'top-x' filter. |
| `[-nb-lines <filter-value>]` | where <filter-value> is the integer value for the 'nb-lines' 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 :

- either Logiscope **Studio**, refer to Section 1,
- or Logiscope **create**, refer to the previous section.

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

## Options

The Logiscope **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. |
| `[-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 `addin` is the name of the addin to be activated and `options` the associated options generating the reports. |
| `[-table]` | 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. |
| `[-noframe]` | generate reports with no left frame. |
| `[-v]` | display the version of the **batch** tool. |
| `[-h]` | display help and options for **batch**. |
| `[-err <log_err_folder>]` | 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 the previously created Logiscope project named **`aProject.ttp`** where:

- the *CodeReducer* verification module has been activated,
- the Logiscope Repository is located in the folder **MyProject/aProject**,

(Refer to the previous section to learn how creating a Logiscope project).

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

```
batch aProject.ttp
```

will:

- perform the parsing of all source files specified in the Logiscope project: **`aProject.ttp`**,
- run the standard TCL script **Reducer_report.tcl** located in **Scripts f**older of the Logiscope standard installation directory to generate the standard *CodeReducer* HTML reports named **testfilescomparison.html** and **testsimilarities.html** in the default a**Project/Logiscope/reports.dir** folder.

# 4. Reference Guide

This section lists all Logiscope *CodeReducer* settings, with their values and effects.

**Important notes:**

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

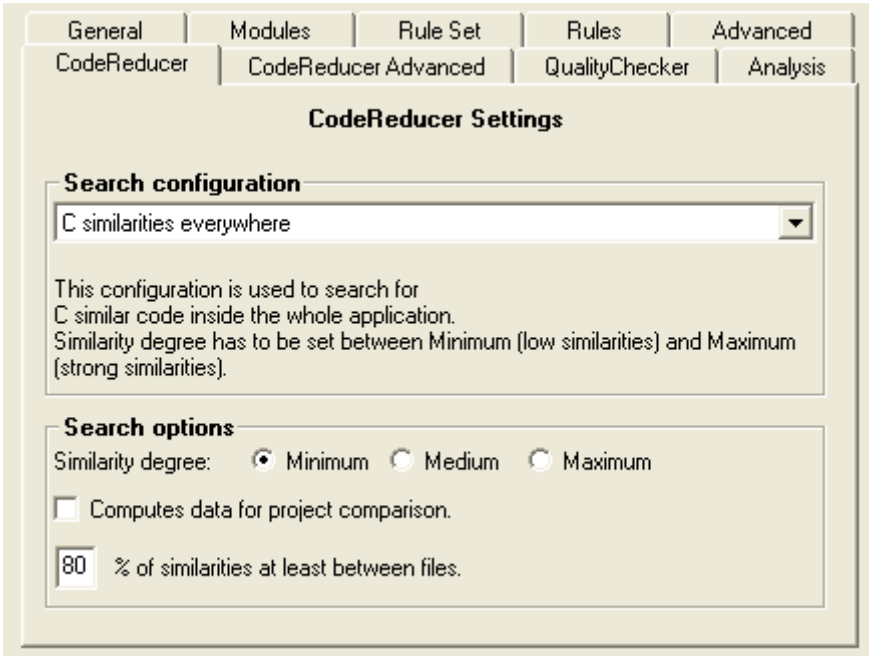
This means that if you change a parameter, and don't rebuild the project, the displayed results will be those of the previous parameters values.

**Do not use the Project > Rebuild All command**
**if you want to keep results from previous scenarios.**

In this chapter, <Language> designates either Ada, C, C++, or Java.

## *4.1. General settings*

The following settings are accessible from Logiscope **Studio** using the **Project > Settings...** command and selecting the "CodeReducer" tab.




### 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 tokens from the "Minimum Degree", and additional tokens depending on the Similarity Degree.

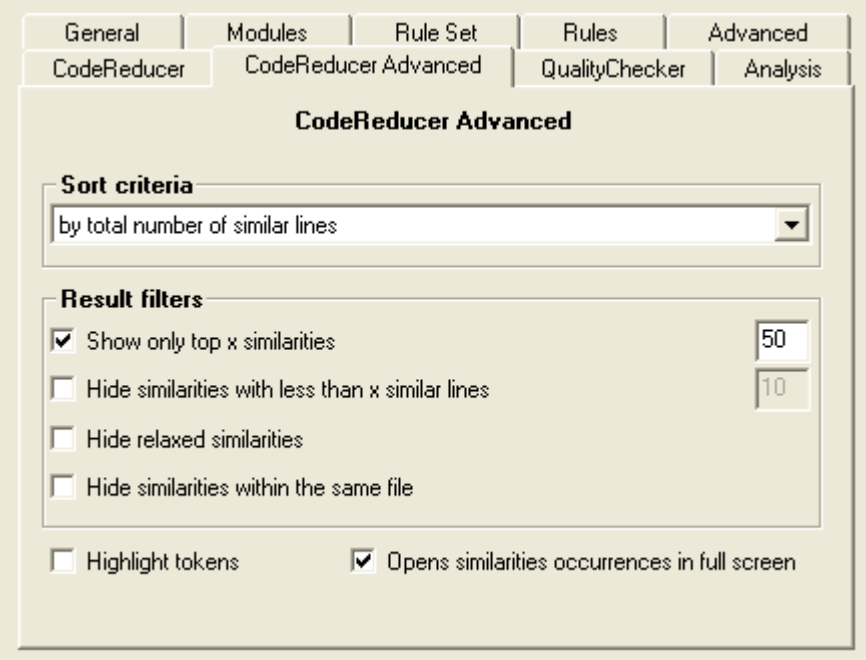
| Search configuration | |
|---|---|
| <Language> similar functions | Scopes are associated to each function source code.<br><Language>: Ada \| C \| C++ \| Java |
| <Language> similar procedures | Scopes are associated to each procedure source code.<br><Language>: Ada |
| <Language> similar packages | Scopes are associated to each package source code.<br><Language>: Ada \| Java |
| <Language> similar classes | Scopes are associated to each class source code.<br><Language>: C++ \| Java |

## Search options

| Similarity Degree | |
|---|---|
| Minimum \|<br>Medium \|<br>Maximum | The similarity degree determines which kind of tokens the search engine must consider when looking for similarities.<br>Choosing a lower or higher degree will produce a coarse-grained or fine-grained result, respectively. |
| **Computes data for projects comparison** | |
| Checked or unchecked (default) | If checked, data necessary for project comparison will be produced when building the project.<br>To activate the projects comparison feature, the workspace must contain two projects exactly (cf. Use Case 3 in previous chapter).<br>You should check this parameter for one of the projects only, or the search will be performed twice. |
| **% of similarities at least between files** | |
| Between 0 and 100<br>(default is 80) | This threshold is the percentage of similarities two files must reach to be considered similar.<br>This parameter only has effect on the file and project comparison features, not on the “Display similarities” feature. |

## 4.2. Advanced settings

The following settings are accessible from Logiscope **Studio** using the **Project > Settings...** command and selecting the "CodeReducer Advanced" tab.




| **Sort criteria** | |
|---|---|
| by max number of similar lines | orders the similarities by the maximum number of similar lines of code found in each occurrence involved in the similarity. |
| by number of similar occurrences | orders the similarities by the number of occurrences found. |
| by number of similar tokens | orders the similarities by the total number of similar token found in the occurrences. |
| by total number of similar lines | orders the similarities by the total number of similar lines of code found in the occurrences. |
| **Result filters** | |
| Shows only the top x similarities | Restricts the number of similarities displayed to the X most relevant ones.<br>Checked by default with value of 50. |
| Hide similarities with less than x similar lines | Similarity occurrences must be at least X lines long to be displayed.<br>The higher this threshold is, the fewer similarities will be found. Default value is 10. |
| Hide relaxed similarities | Similarity occurrences where a relaxation comment has been found are not displayed<br>For more details see Section "Relaxation Mechanism" |

| **Result filters** | |
|---|---|
| Hide similarities when limited to only one file | Discard similarities that were found within the same file. |
| **Highlight Tokens** | |
| Checked or unchecked (default) | If checked, similar tokens are highlighted -in yellow- when displaying similarities occurrences source code.<br>Note: Tokens are not highlighted when displaying results of projects or files comparison.<br>The reason being that the comparison algorithm can detect similar portions of code even if they are not in the same order. In this situation, highlighting tokens would not be relevant. |
| **Opens similarities occurrences in full screen** | |
| Checked (default) or unchecked | If checked, similarities occurrences source code is displayed in full screen.<br>Use the **<ESC>** key to escape full screen mode.<br>The number of occurrences open simultaneously is limited to 5. |

## *4.3. Relaxation mechanism*

As Logiscope *RuleChecker* relaxing acceptable exceptions to violations of programming rules, *CodeReducer* also integrates a mechanism to discard some occurrences of similarities if they are not so accurate or of prior interest for improving maintainability. This can be very useful when analysing similarities in a context where multiple code reviews are performed at various stages of the project.

The *CodeReducer* relaxation mechanism is also based on comments inserted into the code where the tolerated occurrences are. When pieces of code contain such a special comment, they will no longer appear in the list of similarities occurrences when the filter "Hide relaxed similarities" is set in the **CodeReducer Advanced** settings (see previous section "Advanced Settings").

The similarities occurrences that have been relaxed will remain accessible for future reference.

### Comment Relaxation format

The comments to be used to discard pieces of code from similarities occurrences is the following (applicable to C++ or Java):

```
//  {{RELAX<CODEREDUCER> justification
 the piece of code
//  }}RELAX<CODEREDUCER>
```

where: - `justification`: is free text, allowing to justify the relaxation of the occurrence

Note: The combination of characters introducing / closing a comment shall be adapted according to

the syntax applicable to the programming language of the project, e.g.:

- `-- ...` : for Ada.
- `/* ... */` : for C

## *4.4. Programming Rules for Logiscope RuleChecker*

When a Logiscope project is created with both Logiscope *CodeReducer* and Logiscope *RuleChecker* verification modules, the user can benefit from additional programming rules for checking software quality and identifies complex, error-prone components.

Logiscope *CodeReducer* provides Logiscope *RuleChecker* with the following programming rules.

| | |
|---|---|
| `NoSimilarCode` | The code shall not contain similar pieces of code. |
| | All occurrences of found similarities are considered as violations |
| `NoSimilarFiles` | The code shall not contain similar files. |
| | All occurrences of found similarities using the file comparison are considered as violations. |

The rules have 2 parameters that allow to specify the applicable scenario based on:

- <CONF>: the search configuration:
  - CODE: for the "<Language> similarities everywhere",
  - PKG: for the "<Language> similar packages" (Ada and Java only),
  - CLASS: for the "<Language> similar classes" (C++ and Java only),
  - FUNC: for the "<Language> similar functions" (C, C++ and Java only),
  - PROC: for the "<Language> similar procedures" (Ada only)

- <DEG>: the similarity degree :
  - MIN: Minimum similarity degree,
  - MED: Medium similarity degree ,
  - MAX: Maximum similarity degree.

Using jointly parameters and rule renaming facilities of Logiscope *RuleChecker* allows to raise violations from several scenarios simultaneously.

Examples are provided in the Rule Set `CodeReducer.rst` in the standard Logiscope Reference of the Logiscope standard distribution.

For instance, the Rule Set in `RuleSets/C/CodeReducer.rst` for C source code proposes 4 rules derived from the 2 standard rules with different parameters:

```
STANDARD NoDuplicatedFiles RENAMING nosimilarfiles ON CODE MAX END STANDARD
```

```
STANDARD NoDuplicatedCode RENAMING nosimilarcode ON CODE MAX END STANDARD

STANDARD NoDuplicatedFunctions RENAMING nosimilarcode ON FUNC MAX END STANDARD

STANDARD NoSimilarFunctions RENAMING nosimilarcode ON FUNC MIN END STANDARD
```

## *4.5. Source Code Metrics for Logiscope QualityChecker*

When a Logiscope project is created with both Logiscope *CodeReducer* and Logiscope *QualityChecker* verification modules, the user can benefit from additional source code metrics for assessing software quality and identifies complex, error-prone components.

Indeed, a software component containing many similar pieces of code may have a limited level of maintainability.

Logiscope *CodeReducer* provides Logiscope *QualityChecker* with the metrics specified below.

In the following specification:

- <CONF>: designates the search configuration and can be instantiated with the following values:
  - CODE : for the "<Language> similarities everywhere",
  - PKG: for the "<Language> similar packages" (Ada and Java only),
  - CLASS: for the "<Language> similar classes" (C++ and Java only),
  - FUNC: for the "<Language> similar functions" (C, C++ and Java only),
  - PROC: for the "<Language> similar procedures" (Ada only)
- <DEG>: designates the similarity degree and can be instantiated with the following values:
  - MIN: Minimum similarity degree,
  - MED: Medium similarity degree ,
  - MAX: Maximum similarity degree.

- A local similarity is a similarity where all occurrences of similar code are in the same file.
- Value of metrics depends of the activated results filters (cf. §4.2)

Examples are provided in the Quality Model file `QualityModelWithReducer.ref` in the standard Logiscope Reference of the Logiscope standard distribution.

### Application Scope

| | |
|---|---|
| `ap_cr_comp_<CONF>_<DEG>` | Compression ratio for the application regarding the corresponding search configuration <CONF> and similarity degree <DEG>.<br>The compression ratio is the ratio between the estimated gain and |

| | |
|---|---|
| | the total number of lines of the application that may be removed from the source code by factorizing all found similarities. |
| ap_cr_gain_<CONF>_<DEG> | Estimated gain for the application regarding the corresponding search configuration <CONF> and similarity degree <DEG>.<br>The estimated gain is the estimated number of lines that may be removed from the source code by factorizing all found similarities. |
| ap_cr_sim_<CONF>_<DEG> | Number of similarities found in the application regarding the corresponding search configuration <CONF> and similarity degree <DEG>. |
| ap_cr_sline_<CONF>_<DEG> | Total number of lines of occurrences of found similarities in the application regarding the corresponding search configuration <CONF> and similarity degree <DEG>. |

## Module Scope

| | |
|---|---|
| md_cr_lline_<CONF>_<DEG> | Number of lines of code in occurrences of local similarities found in the file regarding the corresponding search configuration <CONF> and similarity degree <DEG>. |
| md_cr_locc_<CONF>_<DEG> | Number of occurrences of local similarities found in the file regarding the corresponding search configuration <CONF> and similarity degree <DEG>. |
| md_cr_lsim_<CONF>_<DEG> | Number of local similarities found in the file regarding the corresponding search configuration <CONF> and similarity degree <DEG>. |
| md_cr_occ_<CONF>_<DEG> | Number of the occurrences of similarities found in the file regarding the corresponding search configuration <CONF> and similarity degree <DEG>. |
| md_cr_sim_<CONF>_<DEG> | Number of distinct similarities where at least a piece of the code file is involved in regarding the corresponding search configuration <CONF> and similarity degree <DEG>. |
| md_cr_sline_<CONF>_<DEG> | Total number of lines of code found in occurrences of similarities found in the files. |

# Notices



The licensed program described in this document and all licensed material available for it are provided by Kalimetrix under terms of the Kalimetrix Customer Agreement, Kalimetrix 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-Kalimetrix products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Kalimetrix has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Kalimetrix products. Questions on the capabilities of non-Kalimetrix 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 Kalimetrix, 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. Kalimetrix, 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 Kalimetrix Corp. Sample Programs. © Copyright Kalimetrix Corp. _enter the year or years_.

# Trademarks