Migration Kit for Solaris OS to Linux

Version 2.0

Migrating applications from Solaris OS to Linux using the Source Checking Tool

Note:

Before using this information and the product it supports, read the general information in **Notices** on page 27.

Second Edition (September 2006)

This edition applies to the Source Checking Tool version 2 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Introduction

Source Checking Tool is a tool that assists in migrating applications from Sun Solaris Operating System to Linux, reducing the time and skill required.

Source Checking Tool supports the following porting environment:

- You can port **from** the following operating systems: Sun Solaris 8 OS and Sun Solaris 9 OS
- You can port **to** the following environments: Linux kernel version 2.6-based distributions that run on any IBM hardware platform, for example, Red Hat Enterprise Linux 4 and 5, and Novell SUSE LINUX Enterprise Server 9 and 10.

Source Checking Tool simplifies porting work by supplying the following:

- Code analysis.

  Source Checking Tool scans files with certain file extensions (.c, .h, .l, .y, .C, .H, .L, .cc, .hh, .cpp, .hpp, .cxx, .hxx) in a code tree. The code tree starts at a directory that you specify. It scans C and C++ code for constructs (APIs, include files, and pragmas) that are specific to Solaris OS and that you need to change for Linux.

  Source Checking Tool does not make changes to the source code. Instead, it highlights the constructs that need to be changed, offers recommendations and examples, and provides an analysis of the size and difficulty of the porting effort.

- Different views.

  Source Checking Tool provides a summary view that organizes the scanned code in different ways. This allows you to work on the code sorted by categorized API functions. For example, you might approach the port by changing all the code related to threading. The summary function assists with this approach by listing all the files that require changes to threading code. In addition to the summary view, Source Checking Tool offers the following views:

  - Portouts. This displays a list of files which have been determined to contain problems. Users can then click a file to open it with an editor.
  - Metrics. This displays a high-level analysis of the porting effort.
  - Graphics. This displays the analysis of the porting effort graphically, using a pie chart.
  - Porting effort analysis.

  Source Checking Tool provides a metrics function to give you an idea of the difficulty of the porting effort. Each flagged function has an associated level of difficulty: Low, medium, high, or to be assessed. Source Checking Tool calculates the percentage of code lines that require changes and the percentage of flagged functions by difficulty level.

Source Checking Tool recognizes over 3800 Solaris OS API calls, pragmas, and include files. While

the porting effort always depends on the applications being ported, an analysis of all Solaris OS calls show that:

- Slightly less than half of all Solaris OS calls are identical to their Linux equivalents, and need not be changed when porting.
- About ten percent of all Solaris OS calls require minimal changes only.
- About five percent require small changes in their local context.
- About 15 percent require greater changes.
- About 25 percent of calls must be assessed in the context of their application before the porting effort can be analyzed.

Getting Started with Source Checking Tool

This chapter gives a brief introduction to the Source Checking Tool and shows a typical porting workflow.

Tool Location

The tool is located in `/opt/IBM/mksl/sct/bin/sct`. Executing this command will start the GUI. (There is not a command-line version of this tool.) It is recommended to set the environment variable EDITOR in your shell to the path of your preferred editor; this will be used by the editing function of the tool.

Overview of porting steps

The following are the typical steps in porting an application from Solaris OS to Linux(R) with the help of Source Checking Tool (see Figure 1).

1. Make a copy of the application's code tree. This will avoid accidental corruption of the original code.
2. Use Source Checking Tool to scan the source code. See **Scanning the application code for the first time** for details.
3. Assess the porting effort using the Source Checking Tool's summary and metrics views, which display information about the size and difficulty of the work. See **Assessing the porting effort** for more information.
4. Plan your approach to porting the code. You can replace Solaris OS code with Linux code or make the code multi-platform. See **Creating multi-platform code** for details.
5. Make changes to the code. Using your favorite editor, make the changes in the application source code files. See **Making changes to the code** for suggestions.

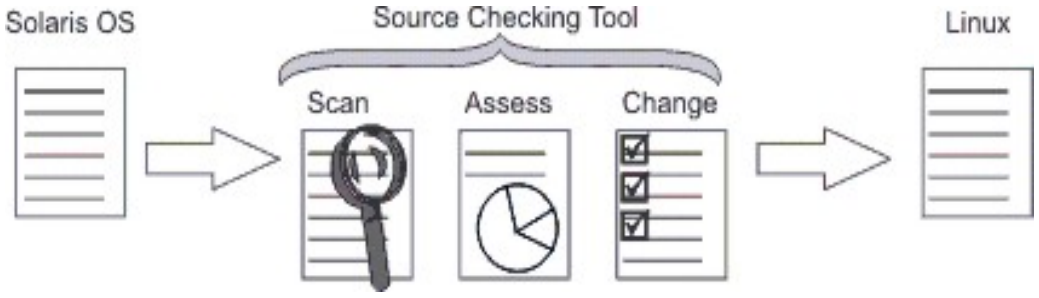

Figure 1. Source Checking Tool supports scanning the code, assessing the work, and changing the code

Scanning the application code for the first time

Scanning a very large code tree may take a few hours. However, you only need to scan once.

To scan the code, follow these steps:

1. On the Source Checking Tool menu bar, click **File --> Scan**.
2. On the Scan Base Directory window, shown in Figure 2, select the directory you want to scan and click **OK**. The tool will also scan any subdirectories under the base directory.

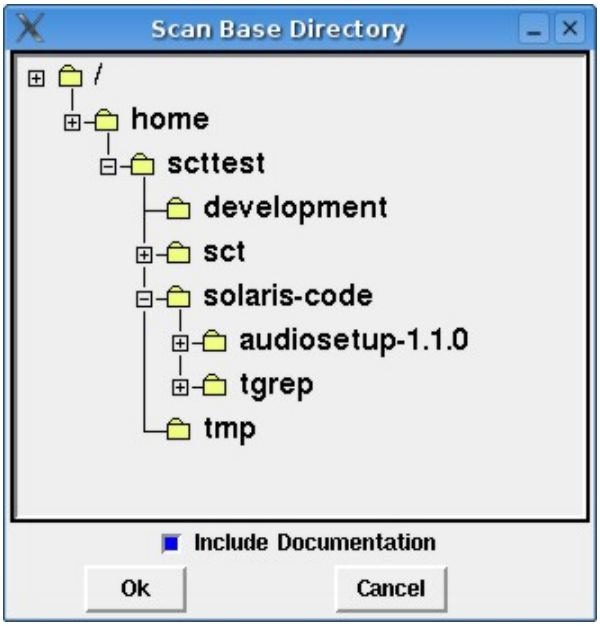

Figure 2. The Scan Base Directory window

Optionally, select **Include documentation** to highlight functions that may require modification and insert documentation about the required modification at each line that it highlights.

You can view the documentation without having inserted it into the file; double-clicking a highlighted function displays the associated documentation in a separate window.

Note:

If you select **Include documentation**, line numbers in the output files will not correspond to line numbers in the original file. This may make it difficult to go back and forth between the two files.

When Source Checking Tool finishes scanning, it displays the code in the summary view. You can also display the scanned code at a later time.

Displaying scanned code

To display already scanned code:

1. On the menu bar, click **File --> Display** to display the scanned directory tree. The code tree contains the original source files, copies of files that have porting issues identified by Source Checking Tool, and some other files that Source Checking Tool creates for its own use.
2. (Optional) On the menu bar, click **File --> Save As** to save the current view (summary, list of output files, metrics, or graphics). The tool saves the information as a formatted report, with a title, time stamp, and indication of the directory. The graphics view is saved as a postscript file; the other views are saved as text files.

As an example, here is a code excerpt as it appears in the original source file:

```
...
        DP(DLEVEL3,("Cascading on %s\n",fpath));
        if (( dp = opendir(fpath)) == NULL) {
            if (!(flags & FS_NOERROR))
                fprintf(stderr,"tgrep: Can't open dir %s, %s. Ignored.\n",
                        fpath,strerror(errno));
            goto DONE;
        }
        while ((readdir_r(dp,dent)) != NULL) {
            restart_cnt = 10;  /* only try to restart the interupted 10 X */

            if (dent->d_name[0] == '.') {
                if (dent->d_name[1] == '.' && dent->d_name[2] == '\0')
                    continue;
```

Figure 3 shows the same code excerpt, scanned and displayed in the Source Checking Tool summary view, including documentation.

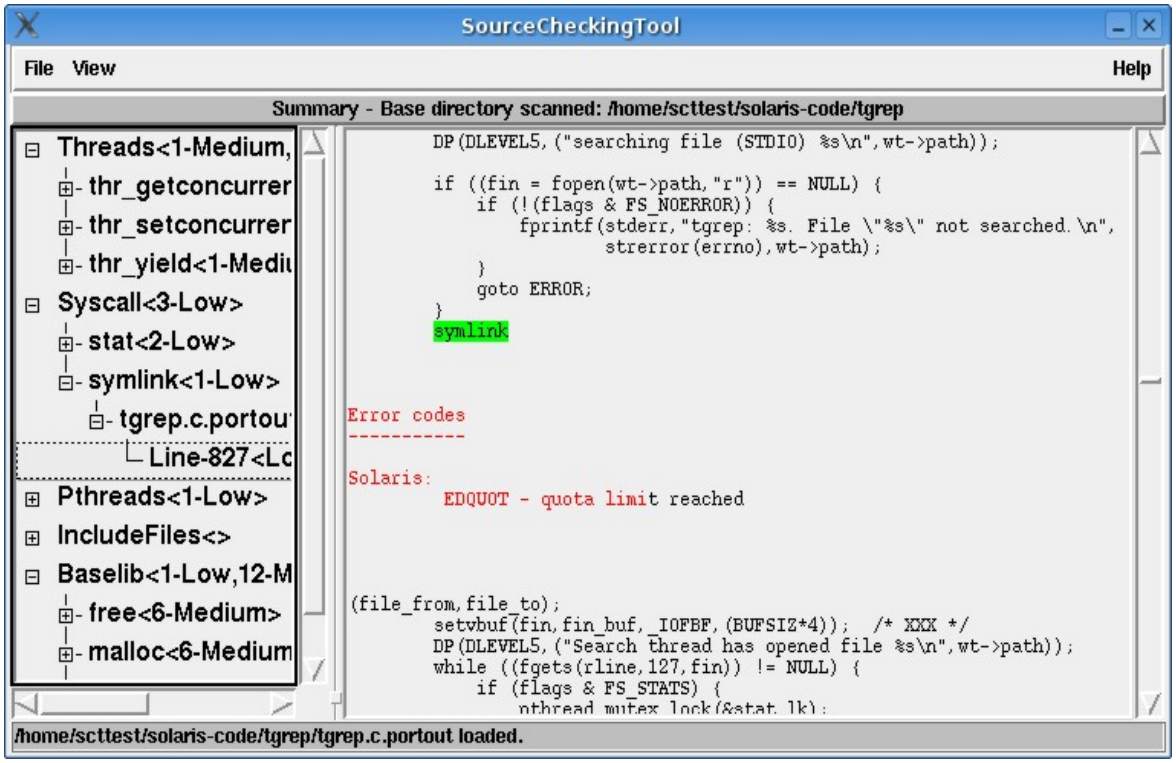

Figure 3. Source Checking Tool summary view example with documentation

Source Checking Tool output files

This section discusses the files that Source Checking Tool produces, portout files, and an error log file.

Portout files

As an alternative to working with the GUI, you can work directly with files the tool creates. The copies of source code that Source Checking Tool creates have an extension of .portout. These portout files are copies of the original source files with markers around recognized porting issues. If you selected **Include documentation** when scanning, the portout files also have porting information and tips inserted as comments. The portout files are created in the directory of the source tree where the original file is located.

Here is the same excerpt as it appears in the portout file:

```
DP(DLEVEL3,("Cascading on %s\n",fpath));
if (( dp = opendir(fpath)) == NULL) {
    if (!(flags & FS_NOERROR))
        fprintf(stderr,"tgrep: Can't open dir %s, %s. Ignored.\n",
                fpath,strerror(errno));
    goto DONE;
}
while ((!%readdir_r%!
```

```
/* Begin SourceCheckingTool Documentation:

Description
-----------

The Solaris OS function call readdir_r() can be used on Linux with minor
modifications. The
Linux default implementation of the readdir_r() call is equivalent to the Solaris
OS POSIX
implementation.

On Solaris OS, there is one more implementation of the readdir_r() call that does
not conform
to the POSIX standards. This implementation of the readdir_r() call is not
available in Linux.

Format
------

Solaris OS:
Default implementation
        #include <sys>
        #include <dirent.h>
        struct dirent *readdir_r(DIR *dirp, struct dirent *entry);
POSIX
        cc [ flag ... ] file ... -D_POSIX_PTHREAD_SEMANTICS [ library ... ]
        int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);

Linux:
        #include <sys>
   #include <dirent.h>
   int readdir_r (DIR *DIRSTREAM, struct dirent *ENTRY, struct dirent **RESULT)

*/

(dp,dent)) != NULL) {
            restart_cnt = 10;  /* only try to restart the interupted 10 X */

            if (dent->d_name[0] == '.') {
                if (dent->d_name[1] == '.' && dent->d_name[2] == '\0')
                    continue;
                if (dent->d_name[1] == '\0')
                    continue;
```

The **bold** indicates text that Source Checking Tool adds to the copy of the file. Note that markers are added to the function name to indicate a degree of difficulty. These markers will cause compiler errors, should you attempt to compile the portout file. Change the portout file before attempting to compile. The markers are explained in Table 1.

Table 1. Portout file markers

Level of difficulty	Beginning marker	Ending marker	Color
High	! ~	~!	Red
Medium	!&	&!	Yellow
Low	!%	%!	Green
To be assessed	!@	@!	White

Error log file

Source Checking Tool creates one log file that is called sct.log. The log file is in the upper-most directory, SCT_files, of the source code tree. Application error messages relevant to problem determination are logged in the log file. For details about messages, see **Appendix. Messages.**

Assessing the porting effort

The metrics view of Source Checking Tool helps you assess the amount of effort involved in your porting project. The information in the metrics view consists of the following:

- Base directory that is scanned
- Total number of lines that are scanned (all files)
- Total number of lines flagged
- Total number of lines flagged with each degree of difficulty (high, medium, low and to be assessed)
- Percentage of total lines that were flagged
- Of the lines that were flagged, the percentage for each degree of difficulty
- Number of assembler files found

To display the metrics view, on the menu bar, click **View --> Metrics**.

The metrics view gives an overview of the entire porting effort, including all files, regardless of category. For example:

```
Base of directory tree scanned: /home/scttest/solaris-code/tgrep

Scanned:                      1969 lines
Flagged:                        20 functions

High difficulty:                 0 functions flagged (0% of the total)
Medium difficulty:              13 functions flagged (65% of the total)
Low difficulty:                  4 functions flagged (20% of the total)
To be assessed:                  3 functions flagged (15% of the total)
```

The degrees of difficulty have the following meanings:

High

There is no equivalent function in Linux, and the code must be re-written.

Medium

There is no exact equivalent function in Linux, but a similar function might be used.

Low

There is an equivalent function in Linux that can be used with only minor changes.

To be assessed

Source Checking Tool cannot determine the degree of difficulty; the porting effort must be assessed in the context of the particular application.

To display the data graphically, on the menu bar, click **View --> Graphics**. Figure 4 shows the data represented in a pie chart.

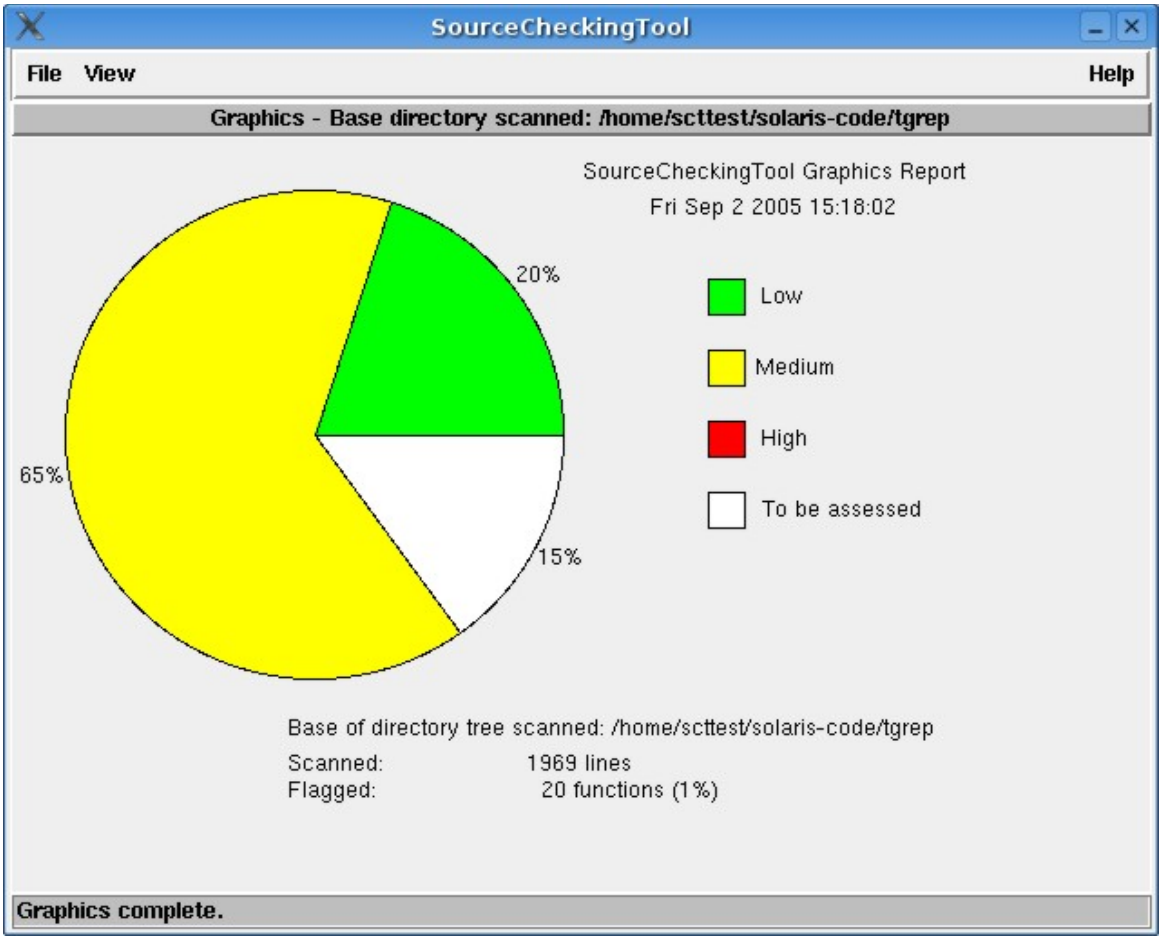

Figure 4. The graphics view

Making changes to the code

When making changes to the code, you can use Source Checking Tool for guidance and planning. Choose the view that suits your approach. Use Source Checking Tool's summary view to display

categorized APIs. The summary view lets you find the files and lines in those files that need changing. Use the portout view when you want to work from the list of files that require changes.

There are many possibilities to access the code and adopt it to Linux, for example:

- Use the Source Checking Tool editor for smaller changes and use **Save As** to replace the original source code file.
- Use Source Checking Tool in conjunction with your favorite editor. In this case you should not enable the option **Include documentation** because the line numbers will not match.
- Use the Source Checking Tool **Include documentation** option to build output files (.portout) with included porting remarks. Copy these files back to your working source code tree and continue with your favorite editor.

The last approach is appropriate for large scale projects that are handled in a larger team and that involve lots of mechanical changes. With the first two approaches you can coordinate your porting work and process the source code by category and function using the Source Checking Tool summary view.

Changing code using the Source Checking Tool editor

Source Checking Tool provides file editing capabilities (invoked from the summary view or portout file list). You may choose which editor to use by setting the EDITOR environment variable in your shell to the path of the editor you wish to use. You may also via the same mechanism open the file in any other program you choose. To invoke these functions, click and hold the right mouse button while in the right pane (showing the contents of the file selected) of the Summary View.

From the portout view in the right panel, you can double-click the highlighted function to get a documentation text. If you use the **Include documentation** option, the documentation is displayed intermixed with the source code.

Creating multi-platform code

You may choose to simply replace Solaris OS code with Linux code, or to make the code multi-platform, including both the Solaris OS and Linux code. To do this, use `#ifdef` statements to tell the compiler what set of code to use.

For example:

```
#ifdef __s390__
pthread_create(&tid[i], NULL, sleeping,(void *)SLEEP_TIME);
#elsif __sun__
thr_create(NULL, 0, sleeping, (void *)SLEEP_TIME, 0, &tid[i]);
#endif
```

The symbols "__s390__" and "__sun__" are examples of symbols that are defined on Sun Solaris OS and Linux for zSeries systems. You can either use other system-defined symbols or use the -D compiler option to define user-created symbols.

When using gcc, there are a number of predefined preprocessor #defines. These include:

__s390__
: When compiling for an IBM zSeries in 64- or 31-bit mode, or an S/390 in 31-bit mode.

__s390x__
: When compiling for a zSeries processor which supports 64-bit addressing.

__powerpc__ or __PPC__
: When compiling for a PowerPC processor.

__powerpc64__ or __PPC64__
: When compiling for a 64-bit PowerPC processor.

__gnu_linux__
: When compiling for Linux.

Source Checking Tool Reference

This chapter presents the details of the Source Checking Tool graphical user interface and lists Source Checking Tool menu items and functions.

The Source Checking Tool window

Figure 6 shows the parts of the Source Checking Tool window.

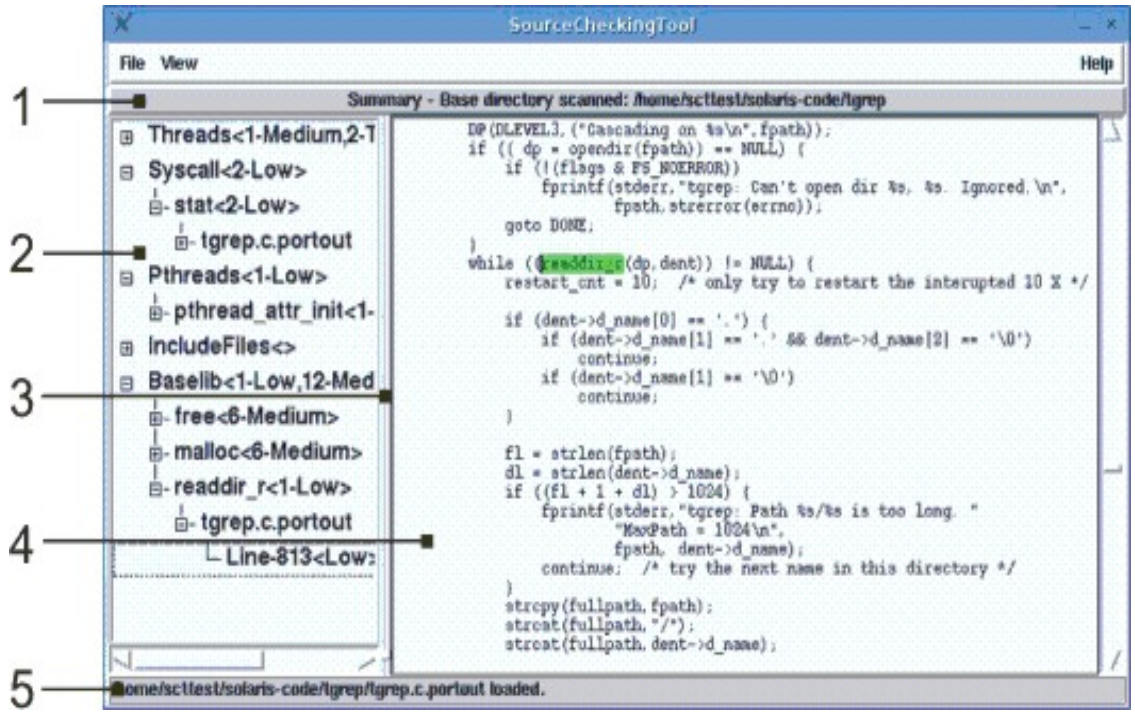

Figure 6. The Source Checking Tool main window

The window parts are:

1. Title line, showing the title of the current view, either Summary, Portout list, Metrics or Graphics, and the base directory that was scanned.
2. Left pane, used for the summary or the list of portout files.
3. Vertical separator between the panes; it can be slid right and left to resize the panes.
4. Right pane, used for displaying the porting documentation or portout file with an editor.
5. Message line or status bar.

Metrics and graphics are shown in a single pane. See **Functions** for more examples of the window.

The menu bar

The items on the menu bar are described below. For details on the associated functions, see **Functions**.

File

Scan

This displays a tree in a pop-up window, from which you can select the directory that contains the files you want to scan.

Display

This displays the tree from which you can select a previously scanned directory. Select the directory to open the summary view (the default).

Save Summary As | Save Portout List As | Save Metrics As | Save Graphics As

This saves the current view (in contrast to **Save As** in the edit window). It displays a file selection that lets you select a path for saving the current view (Summary, Portout list, Metrics, or Graphics). This action is not available when the window is empty. It does not save the contents of the right pane when the window is split.

Exit

This ends the tool.

View

Mutually exclusive actions under **View** control the contents of the main window. They also control the results of the next **File-->Scan** action, so the options are available even if the window is empty.

Portouts

This displays a list of files which have been determined to contain problems. Users can then click a file to open it with an editor.

Summary

This is the default. It displays the summary, which describes and acts as a map to the porting effort.

Metrics

This displays a high-level analysis of the porting effort.

Graphics

This displays the metrics in a graphical format, using a pie chart.

Help

Help

This displays help for the tool in a pop-up window.

About

This displays copyright and release information in a pop-up window

Functions

This section describes the main functions of Source Checking Tool.

Scan

Launch the scan function by clicking **File -->Scan**. The Scan Base Directory window, which contains a directory tree, opens as shown in Figure 7.

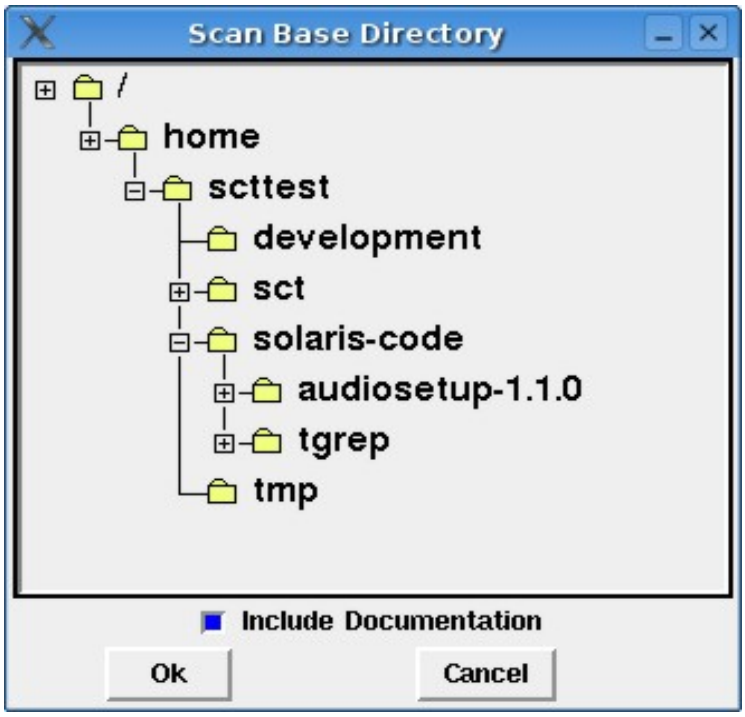

Figure 7. Directory tree example

The window includes an **Include documentation** option. When the option is selected the scan inserts porting documentation at each line that it flags. The default is that the option is not selected. The option is most useful when viewing the portout files outside of Source Checking Tool. When viewing the portout files within Source Checking Tool, users can click a highlighted function to display the documentation in a separate window.

Select a directory from the tree and click **OK** to dismiss the window and cause the tool to scan the files in the directory for potential problems. When the tool finds a file that requires code changes, it creates the portout copy of the file. When the scan is complete, the main window opens with the appropriate output (by default, the summary).

Display

Launch the Display function by clicking **File -->Display** from the action bar. This displays the Display Base Directory window, which contains the same tree as the Scan function. The directory selected on the window is searched for the results of a previous scan, which are then displayed. Note that this pop-up does not include the option to include documentation, as that option cannot be changed with the Display function.

Save as

To launch the **Save as** function, click **File -->Save As** on the menu bar. This displays a file chooser that lets the you specify where to save the current view (summary, list of portout files, metrics or graphics). The information is saved as a formatted report, with a title, time stamp and indication of the directory. The graphics view is saved as a postscript file; the other views are saved as text files.

The following example shows a list of portout files saved and then viewed with the vi editor:

```
                  SourceCheckingTool Summary Report
                  -------------------------------

Time stamp      : Fri Sep 2 2005 15:37:40
Search directory: /home/scttest/solaris-code/tgrep

Threads<1-Medium,2-To be assessed>
  thr_getconcurrency<1-To be assessed>
    tgrep.c.portout

When a summary is saved, the report uses indentation to indicate the tree
structure, for example:
```

```
Syscall<2-Low>
  stat<2-Low>
    tgrep.c.portout
      Line-456<Low>
      Line-1192<Low>
```

Views

This section describes the summary, metrics, and graphics views.

The summary view

The summary is displayed in the left pane of the Source Checking Tool window. It uses one tree to show functions and a second tree to show include files.

The summary groups problem functions by category. The categories are sorted in descending alphabetical order. The functions are in alphabetical order within the category. Under each function

within a category is a list of files that were found to contain it. Under each file name is a list of the specific lines containing the function. The file list uses the portout files rather than the original source files.

The summary also provides some metrics. This includes both a quantitative assessment, the number of lines flagged, and a qualitative assessment, a degree of difficulty. For example, a line in the summary would show `<1-High>` to indicate one line is flagged. It is considered to be a high degree of difficulty. The degree of difficulty can be low, medium, high, or to be assessed.

A degree of difficulty is omitted if no lines were flagged for it. The summary shows the degree of difficulty for every level of the tree except portout files.

From the tree you can:

- Expand or collapse branches by clicking the nodes, marked by a + (when the branch is collapsed) or a - (when the branch is expanded).
- Open the file in an edit session by double-clicking the file name or a line number. Clicking a line number opens the file to that line number. The edit session is in the right pane. The editor is a simple editor, which includes basic function such as search and save as. Users who wish to use another editor must launch that editor separately. See **The editor** for more information.
- Display porting documentation by double-clicking a category or function. The documentation is displayed in the right pane, using the editor, and is appropriate to the selected category or function.

An example of the split window with a summary on the left and a portout file on the right is shown in Figure 10.

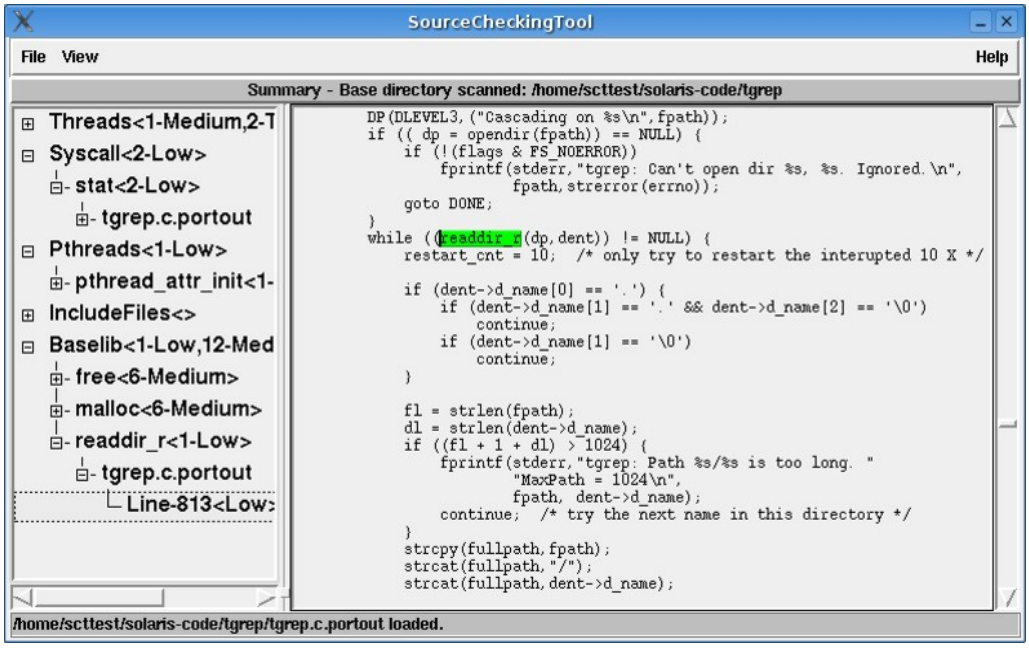

Figure 10. Example of a split window

The metrics view

The metrics view gives an overview in a single pane of the entire porting effort, including all files being ported, regardless of category.

The information in the metrics view consists of:

- Base directory scanned
- Total number of lines scanned (all files)
- Total number of lines flagged
- Total number of lines flagged with each degree of difficulty (high, medium, low and to be assessed)
- Percentage of total lines that were flagged
- Of the lines that were flagged, the percentage for each degree of difficulty
- Number of assembler files (for example, ".s") found

For example:

```
Base of directory tree scanned: /home/scttest/solaris-code/tgrep

Scanned:                      1969 lines
Flagged:                        20 functions

High difficulty:                 0 functions flagged (0% of the total)
Medium difficulty:              13 functions flagged (65% of the total)
Low difficulty:                  4 functions flagged (20% of the total)
To be assessed:                  3 functions flagged (15% of the total)
```

The graphics view

The graphics view shows the metrics in graphical form, using a pie chart in a single pane. Levels of difficulty with no lines flagged are omitted from the pie, though the level of difficulty is included in the legend. For example, the following graphics view shows a case with no high degree of difficulty lines.

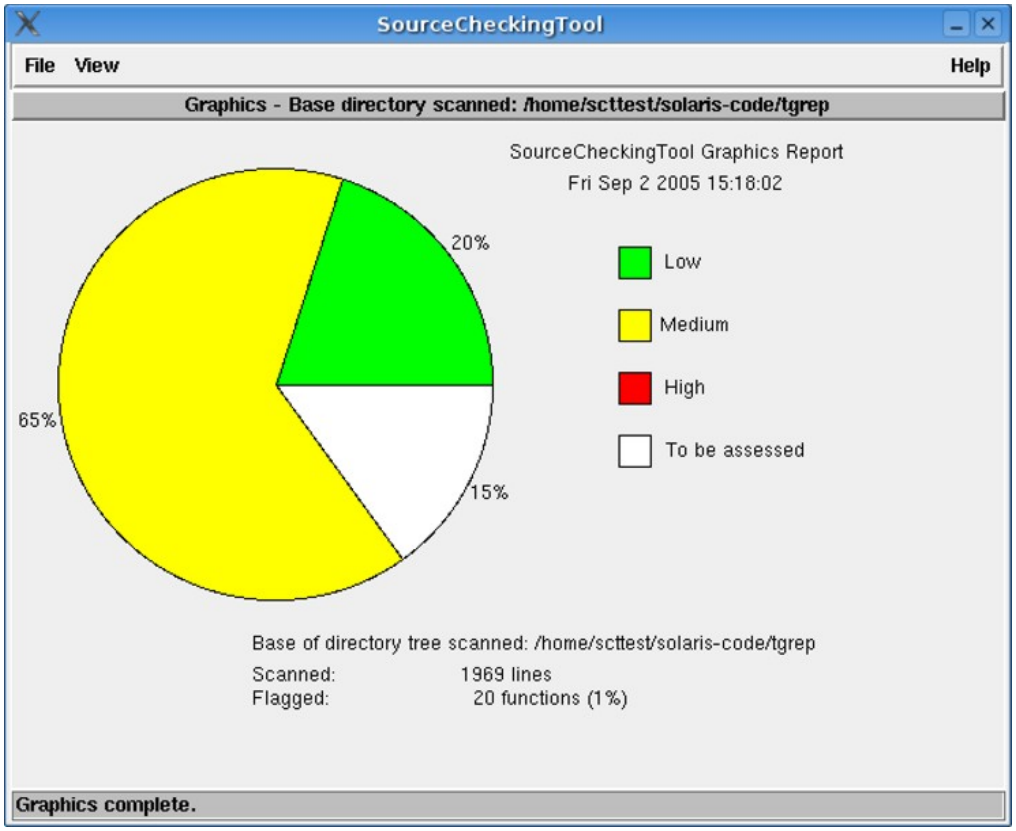

Figure 11. The graphics view

Hints and tips

Search strings are found outside code

Source Checking Tool can search for APIs, include files, and pragmas for text strings. When searching, Source Checking Tool will not only detect API calls that are part of the executable code, but also those in comments and character constants. Finding API names in comments is useful in order to keep comments and code consistent.

Example: Searching for "monitor" in the following code will find one occurrence.

```
#include <stdio.h>
int
main ()
{
  printf(" SCT will find this occurrence of monitor()\n");
}
```

Similar effects exist concerning comments, and combinations of nested comments and string constants.

A typical example of where this might be useful is a program generator where the text output is itself a C program.

Large code samples

When scanning large code samples (more than 100 MB), the tool needs several hours. You might want to plan accordingly and run the scan during lunch break, or another convenient time.

Line numbers in original code and portout files differ

In rare cases, the line breaks in the original file and the portout file differ. All given line numbers refer to the line numbers in the portout file.

Code markers

If a code marker (see Table 1) occurs without a corresponding end marker, the code will not show up as marked in the GUI. Code that is surrounded by a pair of code markers will show up as marked in the GUI, even though the code might not need porting.

Appendix: Messages

Interactive messages are displayed in a message area at the bottom of the main window; error messages are displayed in a pop-up window.

Cannot create the output file file. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

Source Checking Tool could not create a portout file.

User response:

See the message log for more information. This may be a problem with permissions. If the permissions are correct, the disk may have been corrupted. The message log is located in SCT_files under the directory that was scanned.

Cannot find the file file under the installation directory. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

Source Checking Tool could not find a required file.

User response:

Source Checking Tool may not have been installed correctly. Try reinstalling Source Checking Tool as root.

Cannot open the file file under the installation directory. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

Source Checking Tool could not open a required file.

User response:

Source Checking Tool may not have been installed correctly. Try reinstalling Source Checking Tool as root.

Could not open all files under base-directory. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

Source Checking Tool could not open all files to scan, due to permission errors with one or more of the files.

User response:

See the log file for a complete list of files that could not be opened. The message log is located in SCT_files under the directory that was scanned.

Could not open the file file. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

The file may have been corrupted, or the permissions for the file may have been changed while Source Checking Tool was running.

User response:

See the message log. The message log is located in SCT_files under the directory that was scanned.

Could not read one or more of the documentation files. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

Source Checking Tool could not open one or more documentation files.

User response:

See the message log for more information. Ensure that permissions on the documentation folder are correct. If that does not solve the problem, reinstall Source Checking Tool. The message log is located in SCT_files under the directory that was scanned.

Could not remove one or more pre-existing .portout files.

Explanation:

Source Checking Tool cannot remove one or more .portout files.

User response:

None required.

Could not save view file.

Explanation:

The indicated view could not be saved to the indicated file.

User response:

Specify another file, ensuring that you have the necessary permissions.

Could not scan all folders under base-directory. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

Source Checking Tool could not open all the folders in the tree structure to scan for source files.

User response:

See the message log for more information. The message log is located in SCT_files under the directory that was scanned.

File loaded.

Explanation:

The indicated file has been loaded for display in the Source Checking Tool window.

User response:

None required.

File not found.

Explanation:

The indicated file could not be loaded for display in the Source Checking Tool window.

User response:

If it is a portout file that was inadvertently deleted, you can recreate the file by scanning the source again.

File not found. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

The indicated file could not be found.

User response:

See the message log for more information. If the missing file is a documentation file, there may have been an error during the installation process. Reinstall Source Checking Tool as root to avoid any possible permission errors. If the missing file is port_categories.out or port_metrics, the file or disk may have been corrupted, or the file may have been deleted while Source Checking Tool was running. Try scanning your source code again. If this does not solve the problem, trying reinstalling Source Checking Tool as root.

Nothing to display. You must first scan a directory.

Explanation:

You have changed the view without conducting a scan.

User response:

None required. The view will apply to the next scan.

No data found in file. See the Source Checking Tool log file located at directory/SCT_files for more information.

Explanation:

There has been an internal error with Source Checking Tool.

User response:

Try scanning the source code again. The message log is located in /SCT_files under the directory that was scanned.

No source code found under base-directory.

Explanation:

Source Checking Tool did not find any source code files to scan.

User response:

Ensure that you selected the correct directory, and that you have the required permissions.

View file was saved.

Explanation:

The indicated view was saved to the indicated file.

User response:

None required.

Notices

This information was developed for products and services offered in the U.S.A. IBM(R) 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 give you any license to these patents. You can send license inquiries, in writing, to:

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

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.

The licensed program described in this information 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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- IBM
- PowerPC
- S/390
- zSeries(R)

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

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

The shown code samples are excerpts from tgrep.c by Ron Winacott.