Dynamic Memory Solutions

Dynamic Code Coverage ™

User's Guide

Version 1.3 for Solaris™

Document Number DCC20-S-013-0

www.dynamic-memory.com

Notices

Information in this document is subject to change without notice.

Trademarks

Comments

Comments about this document may be sent to:

Dynamic Memory Solutions LLC
Publications Department
30703 Round Lake Road
MT DORA FL 32757

Or sent to *sales@dynamic-memory.com*

Copyright

Contents

What's New in Version 1.3?

Exec and Fork Support

Dynamic Code Coverage has been enhanced to support coverage analysis of programs containing **fork** and **exec** constructs. When your program spawns a child, an additional raw coverage file is created containing code coverage information for the child. Reports can be generated for a child or the results can be merged with the parent. Exec and fork following is disabled via the DYNNOEXEC environment variable.

Dynamic Code Coverage Manual

1. Fork and exec support.

2. Various enhancements including clarification of ***dyncov*** files and merging of results from multiple Dynamic Code Coverage runs.

Preface

Dynamic Memory Solutions

From Day One, we have focused our efforts to provide you with the highest quality software development tools to enable you to maximize your productivity and the quality and stability of your code. We backup our products with the level of technical support that you would expect from experienced software professionals that have been developers themselves. Our goal is to help you use our products in the most effective way possible and we welcome your opinions and suggestions.

You can contact DMS via:

- Our Website -

  www.dynamic-memory.com

- Email -

  Sales -- sales@dynamic-memory.com

  Support – support@dynamic-memory.com

- Telephone -

  Voice – 1-877-293-4144

  Fax – 1-877-293-4144

Support

DMS is fully committed to supporting our products. With each sale, DMS establishes a support agreement where technical support is provided via one or more of the following:

- ***Website -*** Please visit our website at www.dynamic-memory.com for answers to the most frequently asked questions regarding installation and use.

- ***Email -*** support@dynamic-memory.com -- We attempt to answer email questions within three business days though most will be answered by the next business day.

- ***Telephone -*** Telephone support is obtained by calling 1-888-293-4144 weekdays between 9 AM and 6 PM Eastern Time. Telephone support outside these hours may be obtained by special arrangement. Your support agreement may include telephone support, otherwise, there is a charge.

- ***On-site support -*** DMS will travel to your site or connect to your server via an externally accessible network. Our consultants will help diagnose and resolve difficult problems found while using our tools. On-site support is negotiated on a case-by-case basis. Please call our Telephone Support line for more information on this service.

- ***Training –*** DMS will arrange training seminars via electronic delivery or in person at your facility. Please contact us for additional information.

Functional Overview

Dynamic Code Coverage is an easy-to-use tool that indicates which source code is exercised during one or more executions of a program. This information is invaluable in determining how thoroughly a test suite exercises a program.

Insufficient testing is the primary reason programs fail (and usually fail, at the worst possible time.) A single untested error condition can lead to expensive support calls, application downtime and loss of credibility. Ensuring complete code testing coverage is now the single, easiest step to take by a software development organization to ensure software quality.

As with all products in the Dynamic Suite, Dynamic Code Coverage does not require modification of the application to insert compile-time or link-time instrumentation. This remarkable feature raises Dynamic Code Coverage well above the competition.

The major features of Dynamic Code Coverage include:

- Function Coverage – Easy-to-view reports allow a user to determine which functions are called.
- Line Coverage -- An annotated version of the source code shows which source lines are executed.
- Decision Coverage -- Answers the question, "Did each 'if', 'while', 'case', etc statement follow both the 'TRUE' branch and the also the 'FALSE' branch?"
- Branch Coverage -- Gives a measure of how many assembler branch instructions are associated with each line. In addition, a measure of the number of branches taken/not taken is given.
- Summary Reports -- Reports are available by source file, function name, library name or directory. The reports provide important overall statistics on a process.
- Source Code Annotation -- When source code is available, each source file can be marked up. Annotation is an excellent way to zero in on unexecuted lines and branch decisions.
- The ability to perform code coverage analysis on an already executing program.
- C and C++ support – All features work with either programming language.
- 32/64 bit support -- All features work in both memory models.
- Multithread support.
- Fork and exec support – Code coverage analysis continues on children when the user program uses fork or exec to spawn them.

The capabilities of Dynamic Code Coverage enable your organization to create a suite of test cases that thoroughly test your code. Since no instrumentation is required, the tool is perfect for use in production environments as well as test and development environments.

Installing Dynamic Code Coverage

Installation Requirements

Minimum system requirements for Dynamic Code Coverage:

- ❖ A Sun Solaris distribution version 2.8 or 2.9. Please contact us for current availability on other versions and platforms.

- ❖ SUNWlibC is required by the Dynamic Code Coverage product. Please ensure it is installed on the target machine. SUNWlibC is located on your Solaris Installation disks and can be added via the **pkgadd** command. To install this package, you need to be superuser on the system.

- ❖ At least 50 MB of available disk space

Obtaining the Dynamic Code Coverage software

There are two versions of Dynamic Code Coverage. A reduced function demonstration version that is available for download from our website and the full function version available for purchase. For either version, the first step is to read and accept the Dynamic Code Coverage software license. If you do not accept our license, your should not download our software and should remove any existing copies immediately.

Demonstration version

The demonstration version of Dynamic Code Coverage may be downloaded from our website. This version does not require a license key. The installation procedure for this version is the same as for the full version except that the steps related to license key installation are skipped.

Full version

Please contact us to obtain a full version copy of Dynamic Code Coverage. We prefer electronic distribution via our website or email but special arrangements can be made for delivery on Compact Disc.

When you contact us, we will request information from you regarding the system you will use to host our software. This information permits us to generate a license key to permit our software to run on that server or workstation. Your license key options are:

- ❖ Evaluation license – An evaluation license key enables full program operation for a short time to allow evaluation of Dynamic Code Coverage. Normally there is no charge for this type of license.

- ❖ Full license – A full license key enables you to use Dynamic Code Coverage on the designated server or workstation and entitles you to minor program updates. Normally this entitles you to use the software for an unlimited period of time as long as you abide by the license.

Installation procedure

Dynamic Memory Solutions offers a number of products to maximize your productivity and software quality. Normally, you will want to install these products in the same directory. Installation of the Dynamic Code Coverage software does not modify any system files and while installation as root is suggested, it is not required but facilitates use by multiple users. Our products can be installed on a shared file system but you will still require a license key for each client machine.

Dynamic Code Coverage is distributed as a compressed tar file (tar.gz) with a top-level directory named dms-1.3/. The complete directory structure is:

dms-1.3/bin	executable binaries
/lib	supporting libraries
/env	environment setup scripts
/license	product license files
/docs	product documentation
/examples	example program and results
/extras	optional or supporting files

Before you install

In preparation for installation please follow the following steps:

- ❖ Obtain the software from Dynamic Memory Solutions. Choose either the reduced feature, demonstration version or the full version.

- ❖ If installing the full version, obtain either an evaluation or full license key. To provide this key, we require information about the system on which our software will run.. The information we require is purely hardware related and requires no disclosure of software or data on your server.

  There are two ways to determine the required information about your hardware system:

  Demonstration version installed

  If you have installed the demonstration version of any DMS product on the target system, you can execute the command **showHostDetails** located in the DMS product bin/ directory. Before issuing the command, be sure to set DYNROOT and update your PATH variable as described in User configuration on page 13.

```
export  DYNROOT=<directory>
. $DYNROOT/env/dyn.env
showHostDetails

host name solaris1
host id 1234ABCD
```

Demonstration version NOT installed

To obtain the required information, issue the commands **hostname** and **hostid** as either root or user and record the values.

For example, the following commands will capture the information to a file that can be sent to us for key generation:

```
hostname  >  /tmp/<descriptive name>.dmshost
hostid  >>  /tmp/<descriptive name>.dmshost
```

Then either email this information, email this file or print it and fax it to us. Please use a descriptive name that identifies the system to you and we will reference this name when we send you the license key. When you request multiple license keys, this allows you to match the key to the corresponding system.

First install

If you are installing your first DMS product follow these steps to install the Dynamic Code Coverage software.

❖ Decide where you will install Dynamic Code Coverage and create the directory if it does not exist. For example, directory /apps. Ensure that users of the product will have read access to this directory. You may also install the product to a shared file system (e.g. using NFS).

❖ Change to the directory chosen for the install (e.g. /apps). Uncompress and install the product with the following commands:

```
cd /< install directory >
cp <full path / product filename >  .
gunzip < product filename >
tar –xvf  < product filename>
```

This will result in the creation of the dms-1.3/ directory and subdirectories containing the product code. For example, if you installed in the /apps directory, directory /apps/dms-1.3 will be created. This is your DYNROOT directory and you will set the environment variable DYNROOT to it.

❖ Copy the Dynamic Code Coverage license file (sent separately) to a directory where users have read permission. The default location for the license file is $DYNROOT/license/ . License keys may be kept on a shared file system if that is where you installed the software.

If you choose a license key file directory and name other than the default $DYNROOT/license/license.dat, the $DYNLICENSEFILE environment variable must be set to the full path and name of the license file before you run the software.

Additional product install

If you have other Dynamic Memory Solutions products installed, it is highly recommended that you install Dynamic Code Coverage in the same location. This allows users to access any of the products at the same release level without needing to modify environment variables including $DYNROOT, $PATH and $LD_LIBRARY_PATH when switching between products.

When installing a new release of a DMS product, the name of the top-level directory in the archive is different in order to support multiple installed releases. For example, if you have already installed Dynamic Code Coverage release 1.0 in the /apps directory, the product is in directory /apps/dms-1.0. When you install Dynamic Code Coverage release 1.3, it will be installed in directory /apps/dms-1.3. You can use either version by setting your $DYNROOT environment variable to corresponding directory.

User configuration

Before running Dynamic Code Coverage, you must configure the environment by setting a few variables. The DYNROOT environment variable must be set to the directory containing the release (e.g. /apps/dms-1.3). Once this is set, you run a script we provide (dms-1.3/env/dyn.env) to set the other variables PATH and LD_LIBRARY_PATH.

```
# ksh example
export  DYNROOT=<directory>
. $DYNROOT/env/dyn.env
```

The above commands (or similar commands) may be added to a user's shell profile. For example, add the above lines to the .kshrc file to configure the variables and Dynamic Code Coverage will always be available by simply typing its name. For the Solaris default Bourne shell, similar commands are added to the .profile file

Quick Start

Dynamic Code Coverage is very easy to use and comes with default settings appropriate for most C and C++ applications. This section describes how to use the basic features of Dynamic Code Coverage on a program.

To use Dynamic Code Coverage:

1. Compile and link your program as you normally do. It is not necessary to compile with the debug flag, but Dynamic Code Coverage can provide more information if the application is compiled with the debug flag enabled.

2. Ensure that the $DYNROOT environment variable is set and that the $DYNROOT/bin directory is in your $PATH. Executing $DYNROOT/env/dyn.env will add this to your path. Set the $DYNOUTPUT environment variable to the directory where you want Dynamic Code Coverage to write the results.

3. Execute

   ```
   dyncov <programname> [<program options>]
   ```
   where <programname> is the name of your executable program, and <program options> are the command line parameters you use for your program.

   OR

   ```
   dyncovattach <pid>
   ```
   where <pid> is the program id (PID) of an already running program

4. Coverage information is generated when your program exits normally. Terminating Dynamic Code Coverage also causes coverage information to be written to a file. You can use Ctrl-C to end **dyncovattach** and **dyncov** can be ended by stopping the ‘dc32’ program with kill <pid>. Both are acceptable ways to terminate Dynamic Code Coverage.

5. View the coverage reports with one of the following commands. If you did not set $DYNOUTPUT or you do not have write permissions to that directory, the output files will be created in the /tmp directory.

   ```
   dynreport  <programname>
   ```

   Produces the default coverage report, which displays coverage data for each source file in the program.

   ```
   dynreport <programname> -F
   ```

   Displays coverage data for each function in the program.

```
dynreport <programname> -A —s <sourcefile>
```

Displays an annotated source file <sourcefile> indicating which code was executed.

Reports are printed to the standard output but may be redirected to a file.

For a complete list of options, see the Reports section later in this document.

Feature Descriptions

Function Coverage

Dynamic Code Coverage reports which functions in a program have been called. The summary report shows the percentage of functions called. The Function Coverage Report and the annotated source code provide the execution coverage status of individual functions.

Line Coverage

Line coverage indicates the number of source lines executed compared to the total number of source lines. While not as rigorous as decision and branch coverage, line coverage provides an estimate of the execution coverage level of the program.

Line coverage results appear in the Dynamic Code Coverage summary report when the DYNVERBOSE option is used. The number of covered source lines, the total number of source lines, and the percentage of source lines covered are reported. The annotated source files can be used to determine which lines of code have been executed and which have not been executed.

Decision Coverage

Several programming structures are used to conditionally execute sections of code. These structures include 'if' branches, 'while' loops and 'for' loops. Multiple conditions are often combined with Boolean operators in a single decision structure.

Dynamic Code Coverage reports whether a decision evaluated as TRUE or FALSE during execution of the program. A thorough program test suite causes each decision to be evaluated as TRUE and FALSE at least one time each.

Branch Coverage

Multiple conditions can be combined with Boolean operators in a single decision structure. When these complex decisions are compiled, the result is implemented with a number of branch instructions in the assembler code. The number of branches is dependent upon the number of conditions. Each branch evaluates as either TRUE or FALSE. Dynamic Code Coverage tracks of each branch and whether it has been taken during program execution.

A thorough test of a program causes each branch to be executed as least once as TRUE and at least once as FALSE.

Summary Reports

Dynamic Code Coverage creates several different types of coverage reports.

Source File reports show the number and percentage of functions and decision/branches executed in each source file in the program. Source files in shared libraries are also included in the report.

Function reports show the number and percentage of decisions/branches executed in each function or method in the program. Functions in shared libraries are also included in the report.

Library reports show the number and percentage of functions and decision/branches executed in each library in the program. The main executable also appears in the report.

Directory reports show the number and percentage of functions and decision/branches executed in each source file directory used to create the program

Source Code Annotation

If the executable contains symbols (typically there unless stripped) and the source files are available, Dynamic Code coverage can produce an annotated source file indicating the execution status of each function, decision/branch and source line in the file. For more information about the annotated output, see the section on the Dynamic Code Coverage Reports.

Multiple Run Accumulation

Dynamic Code Coverage is a two phase tool. The first phase, analysis, collects code coverage information and saves it to a raw coverage file. The second phase, reporting, produces the desired report or reports from the raw coverage file. Dynamic Code Coverage can combine the coverage analysis from multiple executions of a program to produce a cumulative report. This is very useful when a variety of initial conditions are required to fully execute the program and produce the desired level of code coverage. For more information on multiple-run accumulation, see “Merging results from multiple Dynamic Code Coverage runs” on page 22.

Using Dynamic Code Coverage

Setup

Before using Dynamic Code Coverage, it must be installed on your server with a valid license key. Please see the product installation section, starting on page 10 of this document for more information.

In order to use Dynamic Code Coverage, the $DYNROOT environment variable must be set to the directory chosen for the product at installation. Additionally, you will need to add the $DYNROOT/bin directory to your $PATH environment variable. The setting of these variables is independent of the location of your application program and may be included in your profile for increased ease of use. See User configuration on page 13 for detailed instructions.

Output from Dynamic Code Coverage is directed to the directory specified by the $DYNOUTPUT environment variable. In a development environment, it may be most convenient to set this to the directory used to compile and link the executable. In a test environment where the executable is already built and resides in a shared, read-only directory, $DYNOUTPUT may be set to a directory owned by the tester with write access. If $DYNOUTPUT has not been set or is set to a directory to which the user does not have write access, output will be directed to the */tmp* directory.

Options

Dynamic Code Coverage supports a number of options. These options are enabled by setting the environment variable indicated by the option name to the desired value. They are disabled by unsetting the environment variable. For example, to suppress output banner information, execute this statement:

```
export DYNQUIET=1
```

To re-enable banner information after it has been suppressed, execute this statement:

```
unset DYNQUIET
```

The following options are supported:

DYNROOT = [directory]

The DYNROOT variable must be set in the user's environment. It defines the directory where Dynamic Code Coverage is installed.

Default value is *not set*.

DYNLICENSEFILE = [path and file name]

This variable overrides the default path to the license file. If the license file is not located in the default directory, then this variable provides its location including full path and file name.

Default value is $DYNROOT/license/license.dat

DYNPATH = [path]

This option designates an alternate path where Dynamic Code Coverage can locate the application source files. Access to the source files allows Dynamic Code Coverage to provide annotated source file reports.

Multiple paths can be specified by delimiting their values with a colon (:) within the value of $DYNPATH. For example:

```
DYNPATH=/myproduct/libs/libsrcdir:/myproduct/src/programsrcdir
```

If this variable is not specified, the path included by the compiler in the debug information within the object file will be used.

Default value is *not set.*

DYNOUTPUT = [directory]

The DYNOUTPUT variable specifies the directory where Dynamic Code Coverage writes its output file. If the DYNOUTPUT option is not specified, or designates a directory where the user does not have write permissions, then the output files will be written to the /tmp directory.

This variable also specifies the directory where **dynreport** looks for the coverage data.

Default value is *not set.*

DYNQUIET

Dynamic Code Coverage writes banner information to standard output. Setting the **DYNQUIET** environment variable prevents this writing to standard output (stdout). This is useful in a variety of situations including processing program I/O from the stdin/stdout, piping program output to another program or running the target program in the background.

Set this variable to prevent banner information from being written to standard output. If this variable is not set, banner information is written.

Default value is *not set.*

DYNVERBOSE

This option increases the amount of information Dynamic Code Coverage displays via standard output. Dynamic Coverage reports additional information including the path to each source file when this option is used.

This option can also be enabled with the **–v** option on the **dynreport** command line.

Default value is *not set.*

DYNBREAKCOUNT

This variable indicates the maximum number of times Dynamic Code Coverage examines the condition controlling a loop. Most loops continue to execute while the condition controlling the loop is TRUE. Dynamic Code Coverage monitors the condition controlling the loop until either it has evaluated as both TRUE and FALSE once each, or it has executed $DYNBREAKCOUNT times. In this case, the annotated report will indicate that the condition overflowed the $DYNBREAKCOUNT limit.

Setting this variable to a large value can negatively impact performance of Dynamic Code Coverage, especially if there are a large number of loops that execute many times each. However, setting it to a low value can cause the FALSE case to never be detected on loop structures.

Default value is 100

DYNNOEXEC

By default, Dynamic Code Coverage performs coverage analysis of code started by the user program via a forking mechanism. There may be times when the user does not require this analysis of child processes. This feature is disabled via the DYNNOEXEC environment variable.

Set this variable to prevent code coverage analysis of programs started with fork or exec. If this variable is not set, all code executed as a result of running the user program under Dynamic Code Coverage is analyzed. Libraries, files and functions that are excluded from analysis are excluded for the parent process as well as any children. A separate raw coverage file is generated for each child.

Default value is *not set.*

Invoking Dynamic Code Coverage

The application program should be compiled and linked in your normal manner. It is not necessary to compile the application with the debug flag, but Dynamic Code Coverage can provide line and decision/branch coverage information only with the debug flag enabled.

Set any additional Dynamic Code Coverage options you require for your test. Options are identified in the previous section and are set using environment variables.

After performing these steps, invoke Dynamic Code Coverage as follows:

```
dyncov <programname> [<program options>]
```

where <programname> is the name of your executable program, and <program options> are the command line parameters you normally use with your program.

A raw coverage output file with the name <programname>.dyncov is created. If this file already exists due to previous Dynamic Code Coverage analysis of your program, the dyncov file is amended to include the coverage information from this execution. If your program spawns children, a raw coverage file with the name <programname>.<PID>.dyncov is created for each child.

After your program completes, you may generate coverage reports. To generate the coverage reports execute:

```
dynreport <programname> [<report options>]
```

For complete information on report options, see the section on Dynamic Code Coverage Reports on page 25.

Attaching Dynamic Code Coverage to a running process

Dynamic Code Coverage can be attached to an already running process. In this case, Dynamic Code Coverage reportscoverage data for the period between the time when Dynamic Code Coverage is attached to the process and the time when Dynamic Code Coverage is stopped, or the analyzed process terminates.

The command to attach Dynamic Code Coverage to a process is:

```
dyncovattach <process id>
```

where <process id> is the program id of an already running program

Merging results from multiple Dynamic Code Coverage runs

Many test cases with different initial conditions and data are usually required to completely exercise a program. Dynamic Code Coverage provides two methods to collect and merge the raw code coverage information from these runs.

Automatic Code Coverage Results Merging

When Dynamic Code Coverage is run, an output file with the name of the executable program suffixed with "***dyncov***" is created. This file contains raw code coverage information. If Dynamic Code Coverage is run again against the same executable program, the existing ***dyncov*** raw code coverage file is amended to include this run. Dynamic Code Coverage Reports generated using this ***dyncov*** file provide code coverage analysis of all executions of the user program.

If the user does not want results from multiple runs to be automatically merged, the ***dyncov*** file must be removed or renamed between Dynamic Code Coverage executions.

Explicit Code Coverage Results Merging

When the automatic code coverage results merging feature does not meet your needs, you can create raw code coverage files with different names and merge the results at a later time with the **dynmerge** utility.

Dynamic Code Coverage provides the **dynmerge** utility that allows you to merge the raw coverage results from two independent runs into one coverage file. This utility may be used multiple times to create one coverage file for a test case or even one file for an entire test suite.

The command to merge two Dynamic Code Coverage raw coverage result files is:

```
dynmerge <input coverage file 1> <input coverage file 2> <output
coverage file>
```

where <input coverage file 1> and <input coverage file 2> are coverage files from either a Dynamic Code Coverage run or the resultant coverage file(s) from prior executions of the **dynmerge** command and <output coverage file> is the coverage file from merging the two input files. Note: the <output coverage file> cannot be one of the input coverage files specified to the **dynmerge** command.

The following is an example illustrating the merging of three Dynamic Code Coverage runs whose coverage files are named TestCase1, TestCase2 and TestCase3 into a new file named TestSuite:

```
dynmerge  TestCase1  TestCase2  TestCase1+2
dynmerge  TestCase1+2  TestCase3  TestSuite
```

Coverage reports generated against the coverage file TestSuite can show that the three tests resulted in complete code coverage of the program. If another test case is added, it can also be merged into the TestSuite coverage results file.

By keeping the original TestCase[n] coverage files, a test case can be modified, Dynamic Code Coverage run again, and a new TestSuite coverage results file generated with minimal effort. For example, assume that analysis of the TestSuite merged coverage result file indicated that Test Case 2 missed a branch path it was intended to test. The test case can be modified, Dynamic Code Coverage rerun and a new TestCase2 coverage result file (let's call it TestCase2A) created. Then, the new results can be merged with the results from Test Case 1 and 3 as follows:

```
dynmerge  TestCase2A  TestCase1  TestCase1+2A
dynmerge  TestCase3 TestCase1+2A TestSuite
```

Parent - Child Coverage Results Merging

The **dynmerge** command is also be used to merge the raw code coverage results from parent and children into a single coverage results file. This is done in exactly the same manner described above for **Explicit Code Coverage Results Merging.**

Removing coverage results from prior Dynamic Code Coverage runs

Raw code coverage results may be deleted by removing (deleting) or renaming the ***dyncov*** file for the corresponding user program. Removing the ***dyncov*** file is useful when restarting the code coverage analysis process after modifying source code. Renaming the ***dyncov*** file is also useful when you want to keep code coverage results from multiple Dynamic Code Coverage separate for analysis of each test run or manual merging.

If your source code is modified and you attempt to merge, either automatically or explicitly, with pre-modification code coverage runs, the result is indeterminate. You should remove all existing raw code coverage ***dyncov*** files after you have modified your source code.

Excluding Source Files and/or libraries from reports

By default, Dynamic Code Coverage reports coverage information for all libraries, source files and functions in the program, including the standard C and C++ libraries. This may be more information than you need.

Dynamic Code Coverage provides the ability to exclude certain functions, source files or entire libraries from the coverage reports.

Functions to be excluded should be listed in the file:

dyncov.function.excl

Libraries to be excluded should be listed in the file:

dyncov.library.excl

Source files to be excluded should be listed in the file:

dyncov.sourcefile.excl

These files can be placed either in the user's HOME directory, or in the directory DYNROOT/env

Files in the two directories are merged to create a complete list of code elements to exclude from the reports.

The exclusion files in DYNROOT/env are typically used to list standard C and C++ libraries and third-party libraries linked into the application and the exclusion files in the HOME directory are used by individual developers.

Libraries to be excluded should be added to the exclusion file prior to running **dyncov.** Functions and/or source files to be excluded can be added to the exclusion file either before running **dyncov** or before generating reports with **dynreport.**

Dynamic Code Coverage Reports

Creating Dynamic Code Coverage Reports

After your code coverage data is collected using the **dyncov** command, coverage reports may be created. Several report options are available:

-S show coverage summary by Source file.

-F show coverage summary by Function.

-L show coverage summary by Library and executable.

-D show coverage summary by Directory.

-A Annotate source files with coverage information.

-v Verbose output, provides line coverage information and increases the amount of information displayed in several areas.

-s <source file name> include a specific Source file.

-f <function name> include a specific Function.

-d <directory name> include a specific Directory.

-l <library name> include a specific Library or executable.

The lower case options can be used together, and each may be used multiple times on the command line.

Source File Report (-S)

The Source File Report displays the covered number and total number of functions and decisions/branches in each source file. If neither -S, -F, -L or -D is specified, then this report is generated.

By default, coverage data for all of the source files in the program is generated. Using one or more of the lower case options can reduce the volume of data in the report. If the lower case options are used, then only the specified source files, libraries, directories and functions are reported.

Function Report (-F)

The Function report displays whether or not each function is covered and the number and percentage of decisions and branches within each function that are covered during the program execution.

Library Report (-L)

The Library Report displays the number and percentage of functions, decisions and branches within each library that are covered during the program execution.

Directory Report (-D)

The Directory Report displays the number and percentage of functions, decisions and branches within each directory that are covered during the program execution.

Annotated Source File (-A)

The –A option is used with any other option to produce an annotated source file output indicating which source lines are covered and which decisions/branches are taken during program execution. The coverage status of each source line is indicated in the left margin.

C	The function is **C**overed. The capital C appears on the first executable line of a function.
U	The function is **U**ncovered. The capital U on the first executable line of a function indicates that a function has not executed. No lines, decisions or branches have been executed within the function.
u	The line is **u**ncovered. The lower case u on a line indicates that the line has not been executed. The lower case u only appears in functions that are Covered.
T	A decision evaluated as TRUE. Decisions are associated with control flow statements such as if or while. The upper case T only appears in functions that are Covered.
F	A decision evaluated as FALSE. The upper case F only appears in functions that are Covered.
O	A branch evaluation overflowed. The upper case O only appears in functions that are Covered. For performance reasons, branches are evaluated a limited number of times. The limit can be set with the DYNBREAKCOUNT option.
#	Two numbers can appear on each line. These numbers indicate the number of decisions and branches taken during program execution and the number of possible decisions and branches. A condition or branch can either be taken or not taken and both must be executed for full coverage. For example, an ‘if’ statement with 2 conditions will have 6 possibilities. Each condition can either be TRUE or FALSE (4 combinations), and the entire ‘if’ statement—the decision—can either be TRUE or FALSE.

Report Examples

These examples are created from a program with 3 source files. Two of the source files are is shared library. The other source file is the main executable program.

The name of the program is example32.

The name of the library is libexample32.so

Source File Coverage Report Example

This report is generated using the command line:

```
dynreport example32
```

Source File Coverage Report

```
Dynamic Memory Solutions
Dynamic Coverage Report
Summarizing ...
Covered    Total      Percent    Covered  Total      Percent    Source File
Functions  Functions  Covered    Decision Decision   Covered
1          1          100        6        8          75         examplelib2.c
1          1          100        3        6          50         examplelib1.c
1          1          100        0        0          0          examplemain.c
3          3          100        9        14         64         Totals
```

Library Coverage Report Example

This report is generated using the command line:

```
dynreport example32 –L
```

Library Coverage Report

```
Dynamic Memory Solutions
Dynamic Coverage Report
Summarizing ...
Covered    Total       Percent    Covered  Total      Percent    Library
Functions  Functions   Covered    Decision Total Decision   Covered
2          2           100        9        14         64         libexample32.so
1          1           100        0        0          0          example32
3          3           100        9        14         64         Totals
```

Function Coverage Example

This report is generated using the command line:

```
dynreport example32 –F
```

Function Coverage Report

```
Dynamic Memory Solutions
Dynamic Coverage Report
Summarizing ...
Func Src   Covered  Total    Percent   Source File          Function
Cov  Line  Decision Decision Covered
T    7     6        8        75        examplelib2.c        function2
T    11    3        6        50        examplelib1.c        function1
T    9     0        0        0         examplemain.c        main
100%       9        14       64                             Totals
```

Annotated Source Code Example

This report is generated using the command line

```
dynreport example32 -s examplelib1.c –A
```

Annotated Source Code Report

```
Dynamic Memory Solutions
Dynamic Coverage Report
Summarizing ...
Covered    Total       Percent    Covered  Total      Percent    Source File
Functions  Functions   Covered    Decision Decision   Covered
1          1           100        3        6          50         examplelib1.c
1          1           100        3        6          50   o      Totals
Annotated source: /export/home/dave/example/examplelib1.c
---------------------
0008                    #include <alloca.h>
0009
0010                    int* function1(const int inNbrToMalloc)
0011    C c             {
0012      c                printf("In function1\n");
0013
0014      c                int nbrToMalloc = 2;
0015
0016      c T      3/6     if ((inNbrToMalloc > nbrToMalloc) && (nbrToMalloc <
100000000))
0017                       {
0018      c                    nbrToMalloc = inNbrToMalloc;
0019                       }
0020
0021                       //Allocate an array of chars
0022      c                char *myCharArray = (char *) malloc(nbrToMalloc *
sizeof(char));
0023
0024                       //Allocate an array of ints
0025      c                int *myIntArray = (int *) malloc(nbrToMalloc *
sizeof(int));
0026
0027                       //Advance the pointer to an address within the
allocation, but not at the start
0028      c                myIntArray+=1;
0029
0030                       //Return the pointer to the middle of the array of
ints.  The array of chars is leaked
0031      c                return myIntArray;
0032      c             }
0033
End Of File
```

The annotated source code appears after the summary report for the source file named with the –s option.

The ‘c’ in the left margin indicates that the source line was covered. The ‘C’ on line 11 indicates that the function was covered. The ‘T’ on line 16 indicates that the decision (the ‘if’ statement) was evaluated as TRUE. Note, there is no ‘F’ on line 16. This indicates that the decision was never evaluated as FALSE. ‘3/6’ on line 16 indicates that 3 of the possible 6 decisions/branches on line 16 were executed.

Special report conditions

Using macros can cause the compiler to associate multiple statements to the same line of source code. In this situation, it is possible for Dynamic Code Coverage to report a line of source code as both covered and uncovered. When all statements in the macro are executed, then the source line is reported as covered.

The use of templates can cause functions to appear as both covered and not covered. In this case, multiple instances of a template function can exist in the same program. Each instance may or may not be executed. To gain complete coverage of these functions, each instance must be executed.

Limitations of Dynamic Code Coverage

Dynamic Code Coverage can be used to analyze code execution coverage without recompiling or relinking the application. The information available to Dynamic Code Coverage when reporting coverage is limited to the source code information that is included in the executable program and shared libraries.

Known Limitations

1) Dynamic Code Coverage can only produce annotated source reports when the source files are available.

2) Source code compiled without the debug option (-g), or executables that have been stripped of symbols do not include source line debugging information. Dynamic Code Coverage is unable to provide line coverage or annotated source reports for these programs.

3) Dynamic Code Coverage supports only the ‘stabs’ format of debug information. Other formats such as ‘dwarf’ are not supported. If necessary, compile your source code with the –gstabs option.