

# Oracle<sup>®</sup> Developer Studio 12.5: Code Analyzer Tutorial

June 2016

This tutorial uses a sample program to demonstrate how to use the Oracle Developer Studio compilers, the discover memory error discovery tool, the uncover code coverage tool, and the Code Analyzer GUI to find and correct common programming errors, dynamic memory access errors, and code coverage issues.

- “Introduction to Code Analyzer” on page 2
- “Getting the Sample Application” on page 2
- “Collecting and Displaying Data” on page 3
- “Using the Issues Found by Code Analyzer to Improve Your Code” on page 16
- “Potential Errors Found by Code Analyzer Tools” on page 16

## Introduction to Code Analyzer

Oracle Developer Studio Code Analyzer is an integrated set of tools designed to help developers of C and C++ applications for Oracle Solaris produce secure, robust, and quality software. It works in conjunction with the Oracle Developer Studio compilers; discover, a memory error discovery tool; and uncover, a code coverage tool.

Code Analyzer includes three types of analysis:

- Static code checking as part of compilation
- Dynamic memory access checking
- Code coverage analysis

Static code checking detects common programming errors in your code during compilation. A new compiler option leverages the Oracle Developer Studio compilers' control and data flow analysis frameworks to analyze an application for potential programming and security flaws.

The Code Analyzer uses dynamic memory data collected by discover to find memory-related errors when you run your application. It uses data collected by uncover to measure code coverage.

In addition providing access to each individual type of analysis, Code Analyzer integrates static code checking with dynamic memory access analysis and code coverage analysis, to enable you to find many important errors in your applications that cannot be found by other error detection tools working separately.

## Getting the Sample Application

The source code for the sample program is available in the sample applications zip file on the Oracle Developer Studio 12.5 downloads web page at <http://www.oracle.com/technetwork/server-storage/solarisstudio/downloads/index.html>. After accepting the license and downloading, you can unpack it in a directory of your choice.

The sample application is located in the CodeAnalyzer subdirectory of the OracleDeveloperStudio12.5-Samples directory.

The sample directory contains the following source code files:

```
main.c
prewise_1.c
prewise_all.c
```

```
sample1.c
sample2.c
sample3.c
sample4.c
```

## Collecting and Displaying Data

You can use the Code Analyzer tools to collect up to three types of data.

## Collecting and Displaying Static Error Data

When you build a binary using the `-xprewise` compiler option, the compiler automatically extracts static errors and puts the data in a `static` subdirectory in a `binary-name.analyze` directory in the same directory as the source code. For a list of the types of static errors found by the compiler, see [“Static Code Issues” on page 17](#).

1. In your `sample` directory, build the application by typing the following:

---

**Note** - The `-xprewise` option is synonymous with `-xanalyze=code`, which is being deprecated.

---

- On Oracle Solaris:

```
$ cc -xprewise main.c prewise*.c sample1.c sample2.c sample3.c
```

- On Oracle Linux:

```
$ cc -xannotate -xprewise main.c prewise*.c sample1.c sample2.c sample3.c
```

---

**Note** - You are not compiling `sample4.c` for this portion of the tutorial.

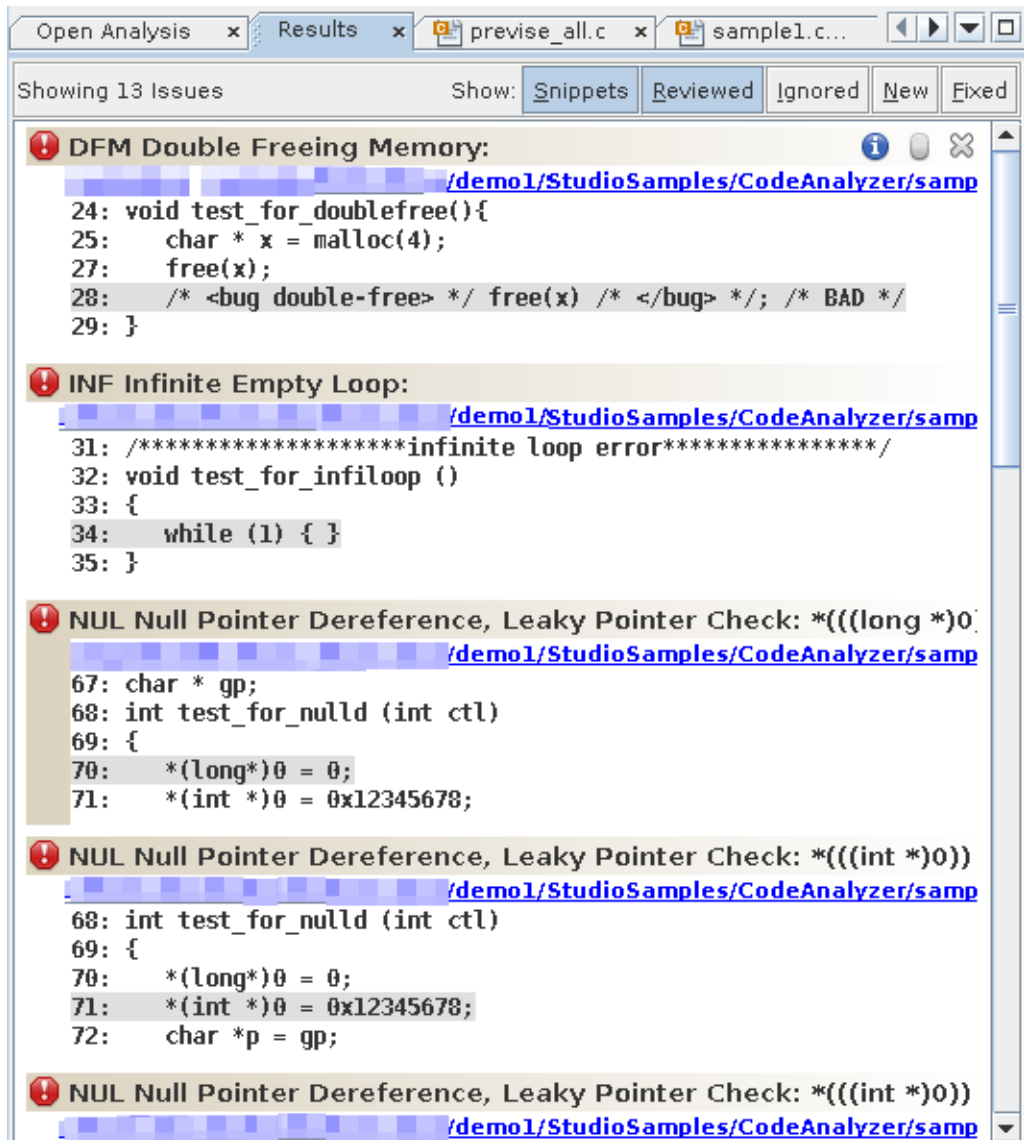
---

The static error data is written to the `sample/a.out.analyze/static` directory.

2. Start the Code Analyzer GUI to view the results:

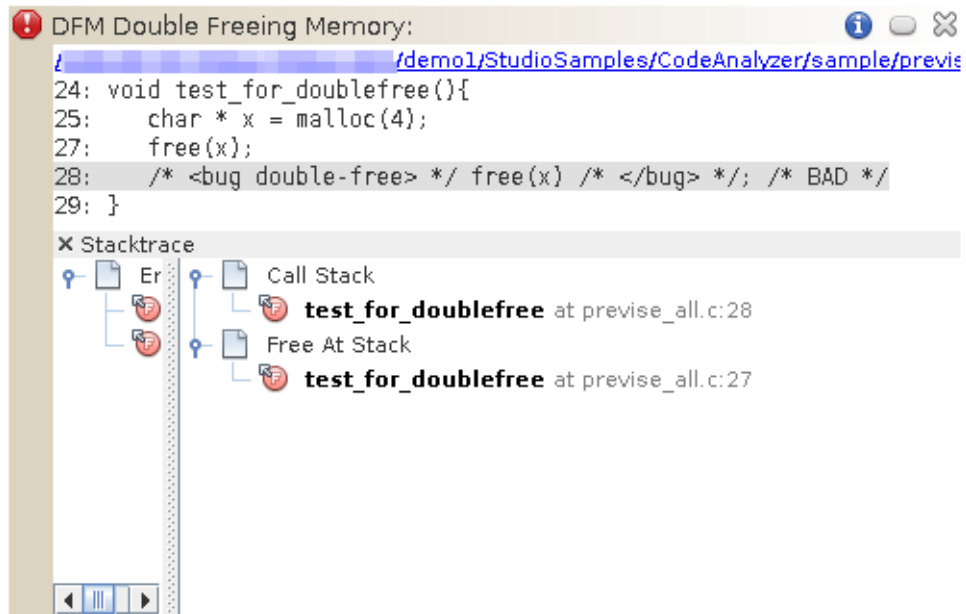
```
$ code-analyzer a.out &
```



3. The Code Analyzer GUI opens and the Results tab displays the static code issues found during compilation. The text at the top left of the Results tab indicates that thirteen static code issues were found.




For each issue, the tab displays the issue type, the path name of the source file in which the issue was found, and a code snippet from that file with the relevant source line highlighted.

- To see more information about the first issue, a Double Freeing Memory error, click the error icon . The stack trace for the issue opens displaying the error path:




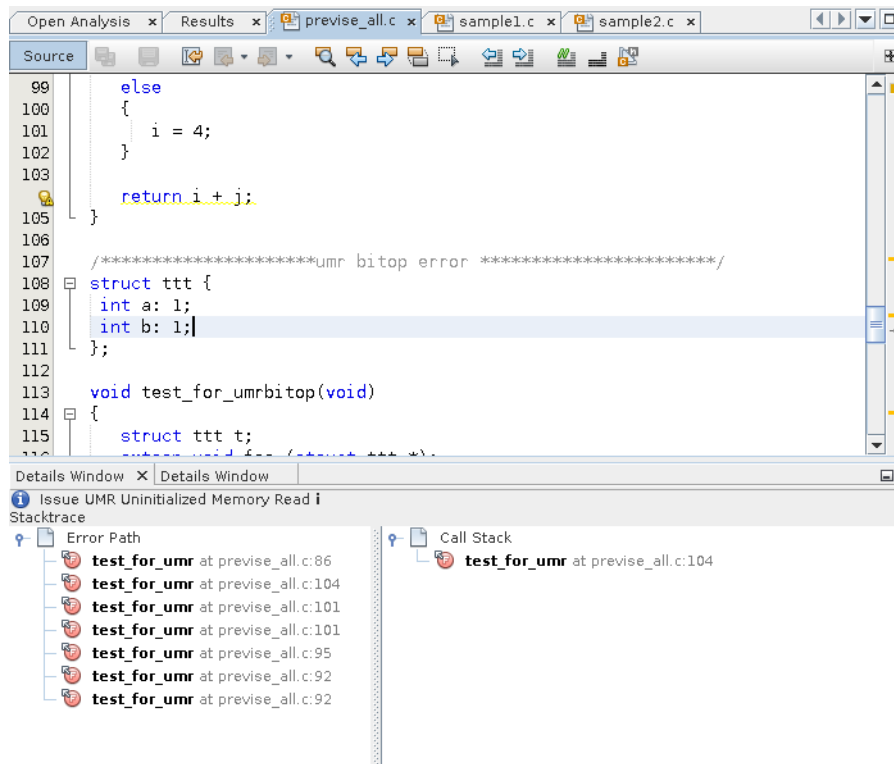
Notice that when you opened the stack trace, the icon in the upper right corner of the issue changed from  to  to indicate that you have reviewed the issue.


---

**Note** - You can hide the issues you have reviewed by clicking the Reviewed button  at the top of the Results tab. Clicking the button again unhides the issues.

---

5. Click the same error icon to close the stack trace.
6. Click the warning icon  of one of the Uninitialized Memory Read warnings to open the stack trace. The error path for this issue contains many more function calls than the one for the Double Freeing Memory issue.
7. Double click the first function call.  
The source file opens with that call highlighted. The error path is displayed in a Details Window below the source code.



8. Double-click the other function calls in the error path to follow the path through the code that leads to the error.
9. Click the Info button  to the left of the issue description to see more information about the UMR error type.  
A description of the error type, including a code example and possible causes, is displayed in the online help browser.
10. Close the Code Analyzer GUI by pressing the X in the upper right corner.

## Collecting and Displaying Dynamic Memory Usage Data

Regardless of whether you have collected static data, you can compile, instrument, and run your application to collect dynamic memory access data. For a list of the dynamic memory access errors found by instrumenting your application with `discover` and then running it, see [“Dynamic Memory Access Issues” on page 17](#).

1. In your `sample` directory, build the sample application with the `-g` option.  
This option generates debug information that enables Code Analyzer to display source code and line number information for errors and warnings.
  - On Oracle Solaris:
 

```
$ cc -g main.c previse*.c sample1.c sample2.c sample3.c
```
  - On Oracle Linux:
 

```
$ cc -xannotate -g main.c previse*.c sample1.c sample2.c sample3.c
```

---

**Note** - You are not compiling `sample4.c` for this portion of the tutorial.

---

2. Save a copy of the binary to use when you collect coverage data because you cannot instrument a binary that is already instrumented.

```
$ cp a.out a.out.save
```

3. Instrument the binary with `discover`.

```
$ discover -a a.out
```

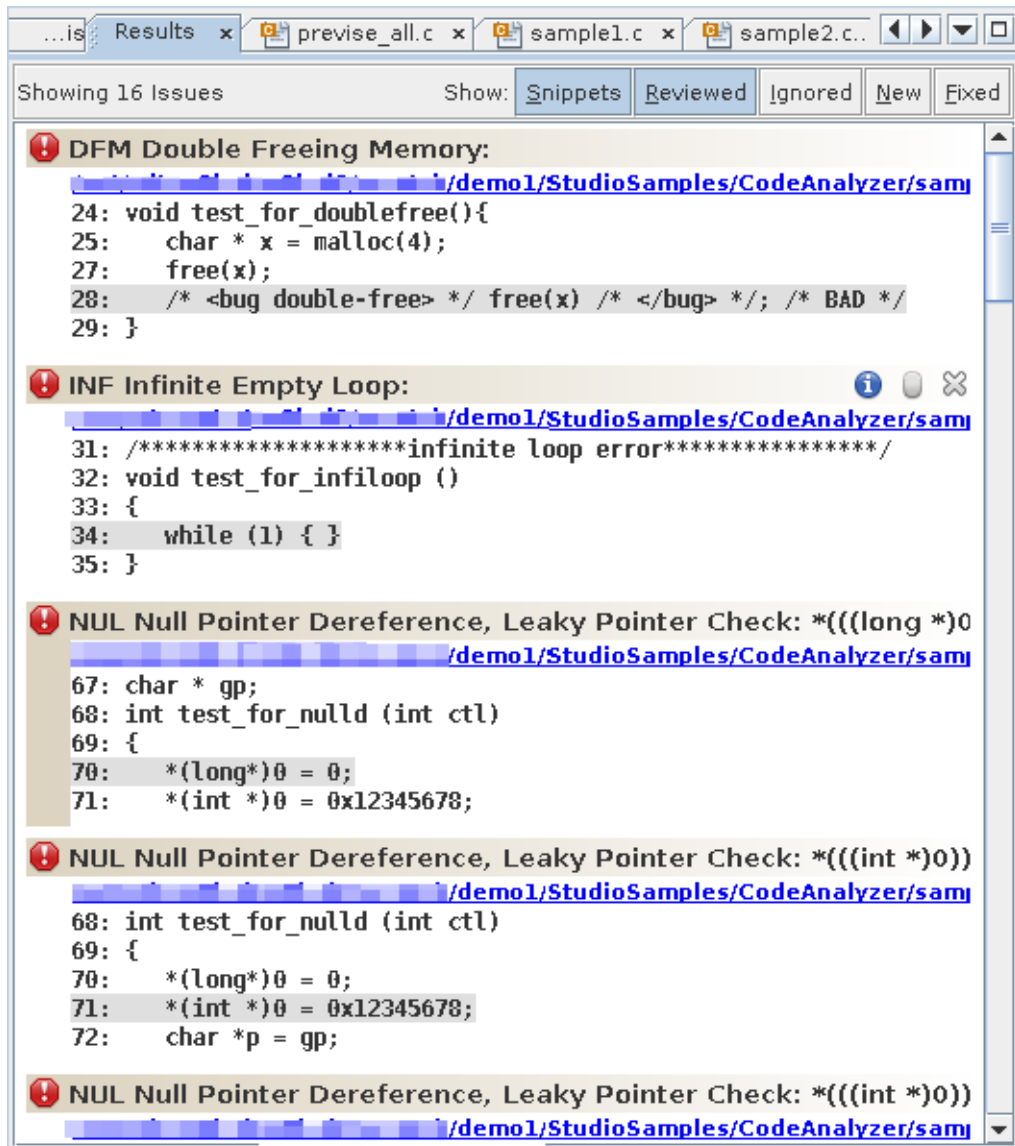
4. Run the instrumented binary to collect the dynamic memory access data.

```
$ ./a.out
```

The dynamic memory access error data is written to the `sample/a.out.analyze/dynamic` directory.

5. Start the Code Analyzer GUI to view the results.

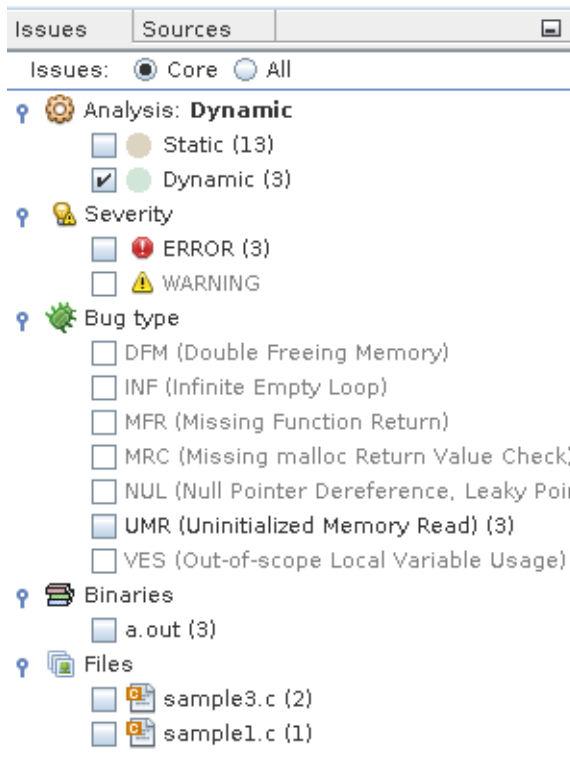
```
$ code-analyzer a.out &
```



The Results tab shows both static issues and dynamic memory issues. The background color behind an issue description indicates whether it is a static code issue (tan) or a dynamic memory access issue (pale green).

- To filter the results and show just the dynamic memory issues, select the Dynamic option in the Issues tab.





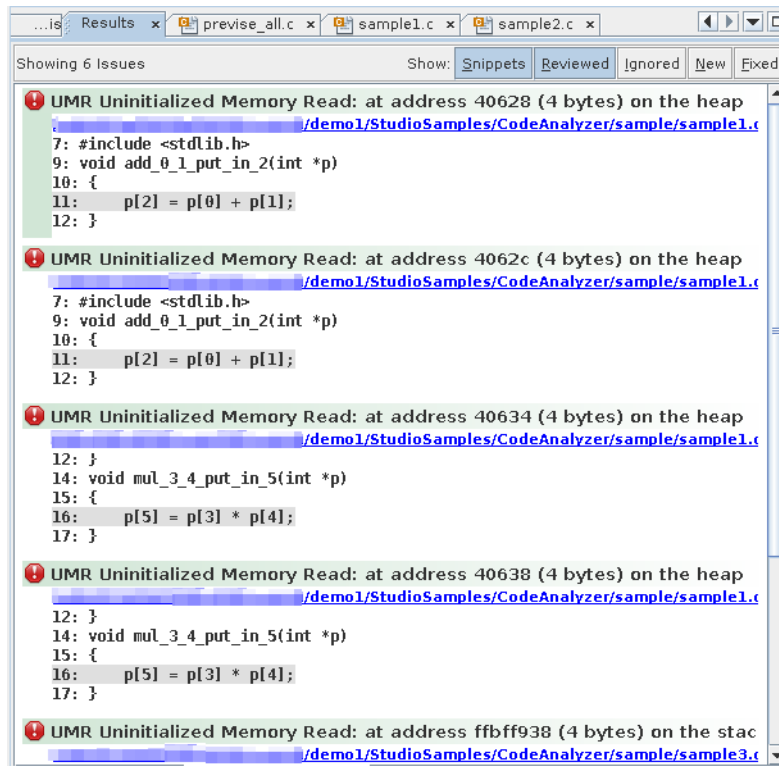
The Results tab now shows just the three core dynamic memory issues.


---

**Note** - Core issues are issues that, when fixed, are likely to eliminate the other issues. A core issue usually combines several of the issues listed in the All view because, for example, those issues have a common allocation point or occur at the same data address in the same function.

---

7. To see all of the dynamic memory issues, select the All radio button at the top of the Issues tab. The Results tab now displays six dynamic memory issues.



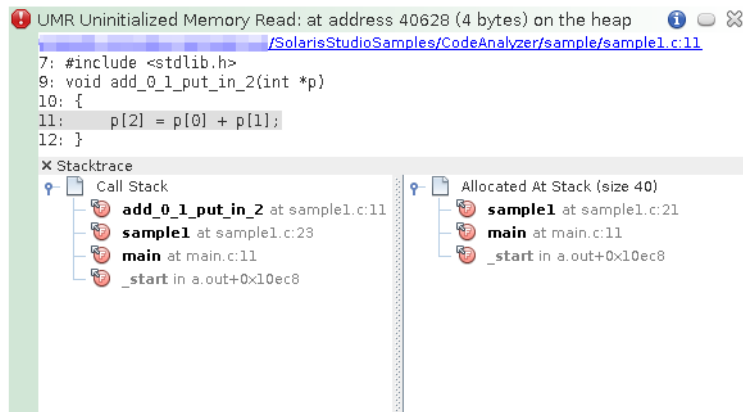
Look at the three issues that were added to the display and see how they are related to the core issues. Fixing the cause of the first issue in the display is likely also to eliminate the second and third issues. To hide the other dynamic memory access issues while you investigate the first one, click the Ignore button  for each of the issues.

---

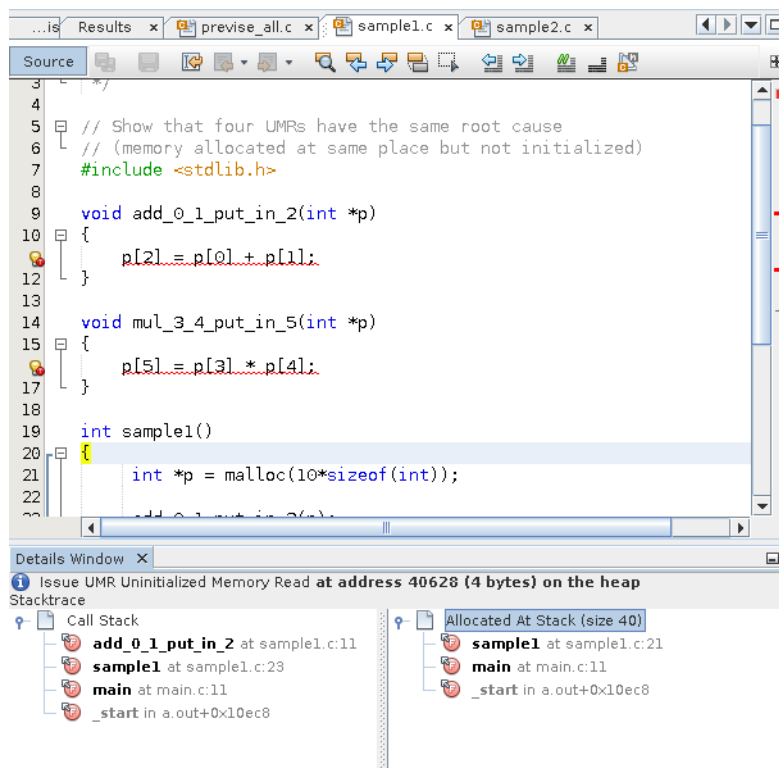
**Note** - You can later redisplay the closed issues by clicking the Ignored button at the top of the Results tab.

---

- Investigate the first issue by clicking the error icon to display the stack trace. For this issue, the stack trace includes the Call Stack and the Allocated At Stack.



9. Double-click function calls in the stacks to see the associated lines in the source file. When the source file opens, the stack trace is displayed in a Details window below the file.



10. Close the Code Analyzer GUI by pressing the X in the upper right corner.

## Collecting and Displaying Code Coverage Data

Regardless of whether you have collected static data or dynamic memory access data, you can compile, instrument, and run your application to collect code coverage data. Note that when you built application with the `-g` option before you collected dynamic memory error data, you saved a copy of the binary before instrumenting it.

1. Copy the saved binary to instrument for coverage data collection.

```
$ cp a.out.save a.out
```

2. Instrument the binary with uncover.

```
$ uncover a.out
```

3. Run the instrumented binary to collect the code coverage data.

```
$ ./a.out
```

The code coverage data is written to an a.out.uc directory in your sample directory.

4. Run uncover on the a.out.uc directory.

```
$ uncover -a a.out.uc
```

The code coverage data is written to the sample/a.out.analyze/uncover directory..

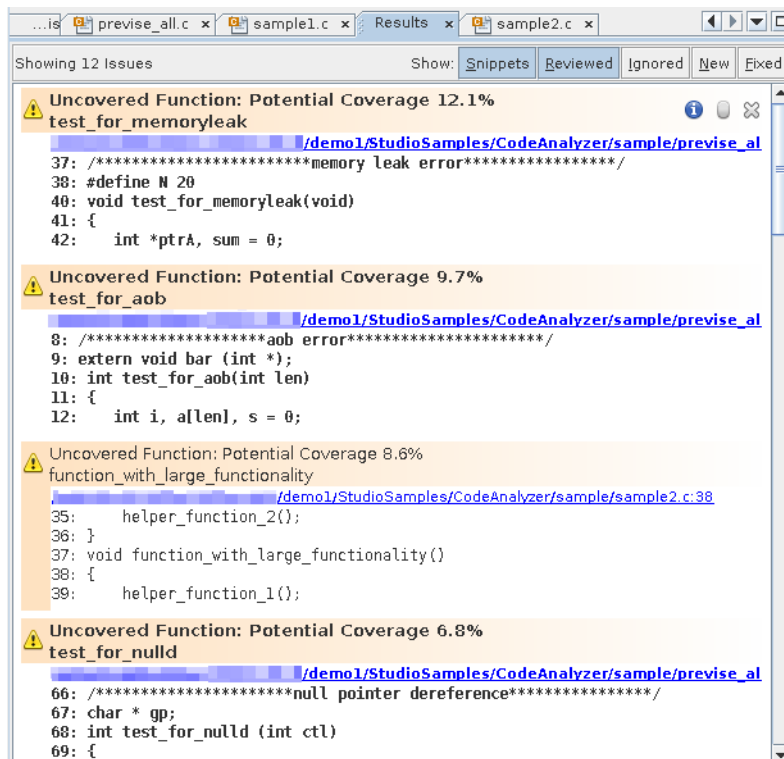
5. Start the Code Analyzer GUI to view the results:

```
$ code-analyzer a.out &
```


The Results tab shows static issues, dynamic memory issues, and code coverage issues.

6. To filter the results and show just the code coverage issues, select the Coverage option in the Issues tab.

The Results tab now shows just the twelve code coverage issues. The description of each issue includes a potential coverage percentage, which indicates the percentage of coverage that will be added to the total coverage for the application if a test covering the relevant function is added.



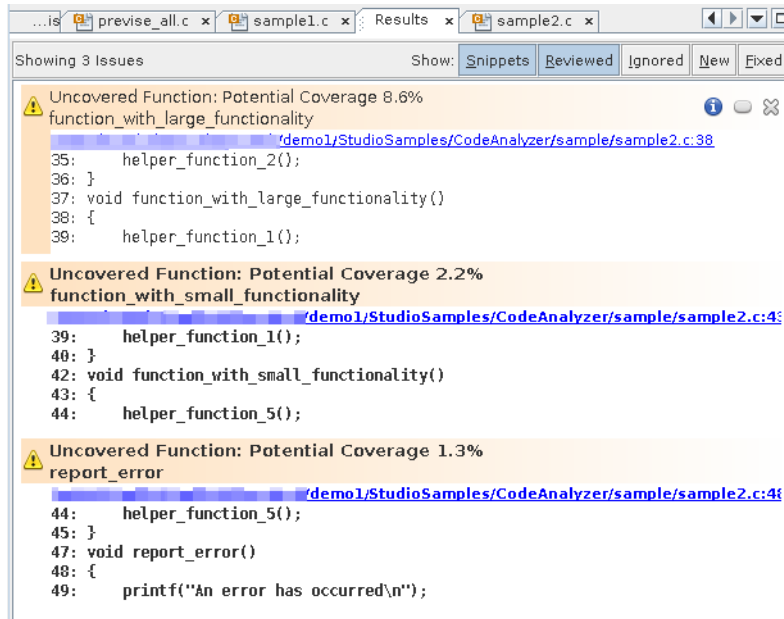
---

**Tip** - To see all the issues without scrolling up and down, click the Snippets button  at the top of the Results tab to hide the code snippets.

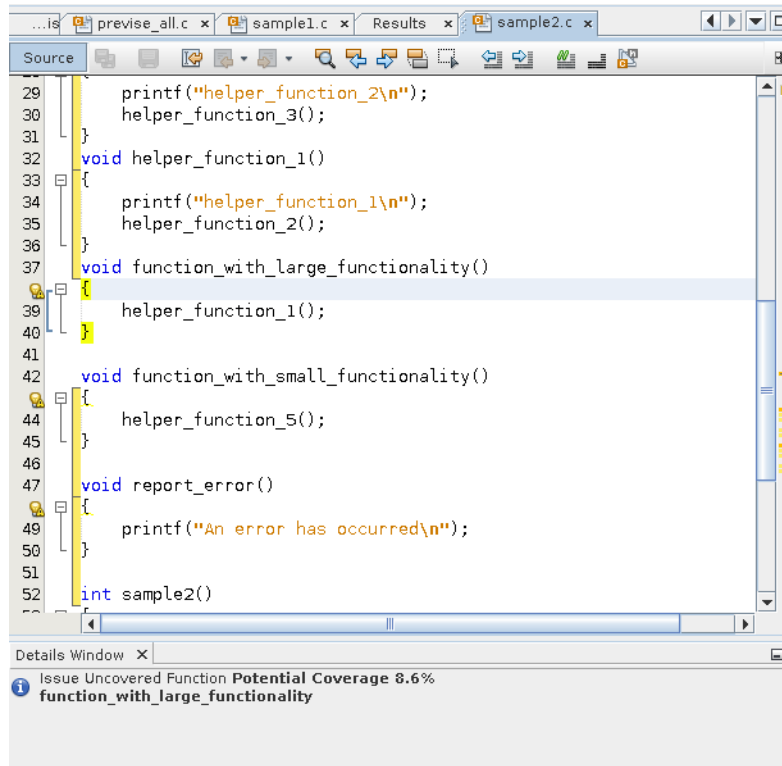
---

In the Issues tab, nine of the coverage issues are in the `prewise_all.c` source file, three are in `sample2.c`, and one is in `prewise_1.c`.


7. To show just the issues for the `sample2.c` file, select the option for that file on the Issues tab. The Results tab now shows just the three code coverage issues found in `sample2.c`.



8. Open the source file by clicking the source file path link in one of issues. Scroll down in the source file until you see the warning icons in the left margin.



Code that is not covered is marked with a yellow bracket.

The coverage issues found in the file are marked with warning icons .

## Using the codean Command-Line Tool

You can similarly use all of the features in Code Analyzer with the codean command . This section is a short tutorial on how to use the codean command to catch new static code issues in your code, using the same sample program from OracleDeveloperStudio12.5-Samples.

1. Previous sections of this tutorial did not compile the sample4.c. Preview this file with the cat command.

```
$ cat sample_4.c
int another_new_umr()
{
    int i;
    if (i)
        return 0;
    else
        return 1;
}
```

Notice that int i is uninitialized.

2. Compile the source and generate the static report.

On Oracle Solaris:

```
$ cc -g -xprewise main.c prewise_1.c prewise_all.c sample1.c sample2.c sample3.c
```

On Oracle Linux:

```
$ cc -xannotate -g -xprewise main.c prewise_1.c prewise_all.c sample1.c sample2.c sample3.c
```

3. Save the static report using the codean --save option to a.out.

```
$ codean --save -s a.out
```

4. Recompile the sample application, this time including sample4.c.

On Oracle Solaris:

```
$ cc -g -xprewise *.c
```

On Oracle Linux:

```
$ cc -g -xannotate -xprewise *.c
```

This new function is never called from main, however it will introduce a new UMR error.

5. Use the --whatisnew option to get a report on the newly added static issue.

```
$ codean --whatisnew -s a.out
```

```
STATIC report of a.out showing new issues:
```

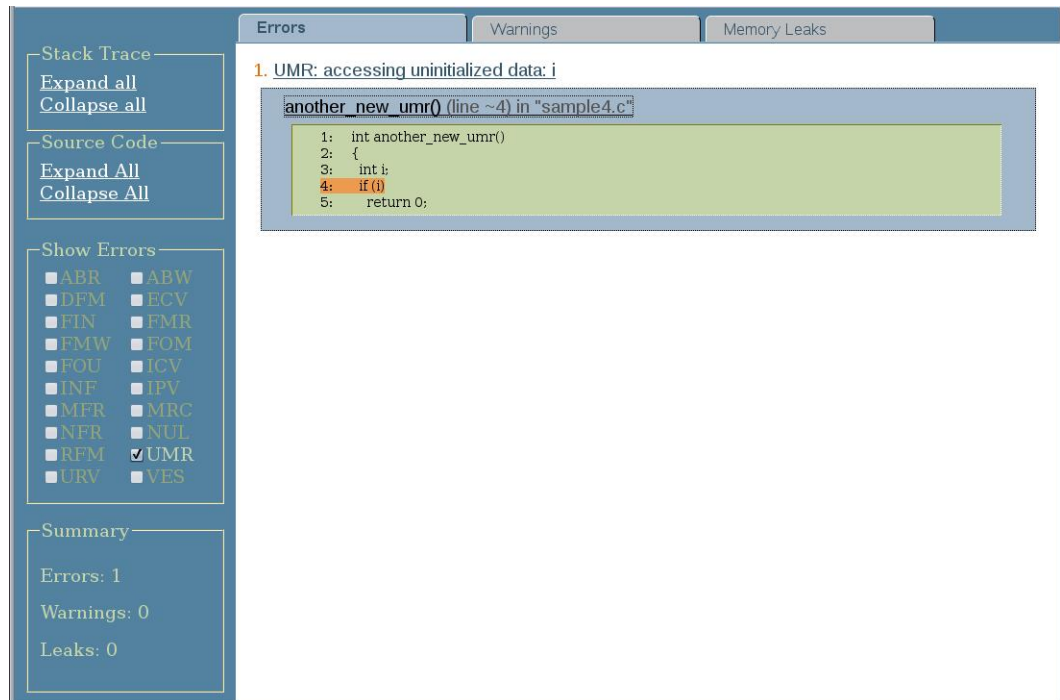
```
Compare the latest results against a.out.analyze/history/2014.8.4.14.49.56...
```

```
ERROR 1 (UMR): accessing uninitialized data: i at:
```

```
another_new_umr() <sample_4.c : 4>
1:     int another_new_umr()
2:     {
3:         int i;
4:=>     if (i)
5:         return 0;
```

```
PREWISE SUMMARY for a.out: 1 new error(s), 0 new warning(s), 0 new leak(s) in total
```

The following figure shows the HTML report on static code issues generated by codean.



For more information about codean, see [“Using the Code Analyzer Command-Line Tool \(codean\)”](#) in *Oracle Developer Studio 12.5: Code Analyzer User’s Guide* and the codean(1) man page.

## Using the Issues Found by Code Analyzer to Improve Your Code

By fixing the core issues found by Code Analyzer, you should be able to eliminate many other issues found in your code, and make major improvements in its quality and stability.

Although static error checking can find the risky code in your application, it can also generate false positives. Dynamic checking can help verify and eliminate these errors, providing a more accurate picture of the issues in your code. Code coverage checking can also help you improve your dynamic test suites.

Code Analyzer integrates the results of these three types of checking to give you the most accurate analysis of your code all in one tool.

## Potential Errors Found by Code Analyzer Tools

The compilers, discover, and uncover find static code issues, dynamic memory access issues, and coverage issues in your code. This section lists the specific error types that are found by these tools and analyzed by Code Analyzer.

For more information about these errors and warnings, see [Appendix A, “Errors Analyzed by Code Analyzer,”](#) in *Oracle Developer Studio 12.5: Code Analyzer User’s Guide*.



## Static Code Issues

Static code checking finds the following types of errors:

- ABR: beyond array bounds read
- ABW: beyond array bounds write
- DFM: double freeing memory
- ECV: explicit type cast violation
- FMR: freed memory read
- FMW: freed memory write
- INF: infinite empty loop
- MLK: memory leak
- MFR: missing function return
- MRC: missing malloc return value check
- NFR: uninitialized function return
- NUL: null pointer dereference, leaky pointer check
- RFM: return freed memory
- UMR: uninitialized memory read, uninitialized memory read bit operation
- URV: unused return value
- VES: out-of-scope local variable usage

## Dynamic Memory Access Issues

Dynamic memory access checking finds the following types of errors:

- ABR: beyond array bounds read
- ABW: beyond array bounds write
- BFM: bad free memory
- BRP: bad realloc address parameter
- CGB: corrupted guard block
- DFM: double freeing memory
- FMR: freed memory read
- FMW: freed memory write
- FRP: freed realloc parameter
- IMR: invalid memory read
- IMW: invalid memory write
- MLK: memory leak
- OLP: overlapping source and destination
- PIR: partially initialized read
- SBR: beyond stack bounds read
- SBW: beyond stack bounds write
- UAR: unallocated memory read
- UAW: unallocated memory write
- UMR: uninitialized memory read

Dynamic memory access checking finds the following types of warnings:

- AZS: allocating zero size

- MLK: memory leak
- SMR: speculative uninitialized memory read

## Code Coverage Issues

Code coverage checking determines which functions are uncovered. In the results, code coverage issues found are labeled as Uncovered Function, with a potential coverage percentage, indicating the percentage of coverage that will be added to the total coverage for the application if a test covering the relevant function is added.

**Part No: E60758**

Copyright © 2014, 2016, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

**Référence: E60758**

Copyright © 2014, 2016, Oracle et/ou ses affiliés. Tous droits réservés.

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois sur la propriété intellectuelle. Ils sont concédés sous licence et soumis à des restrictions d'utilisation et de divulgation. Sauf stipulation expresse de votre contrat de licence ou de la loi, vous ne pouvez pas copier, reproduire, traduire, diffuser, modifier, accorder de licence, transmettre, distribuer, exposer, exécuter, publier ou afficher le logiciel, même partiellement, sous quelque forme et par quelque procédé que ce soit. Par ailleurs, il est interdit de procéder à toute ingénierie inverse du logiciel, de le désassembler ou de le décompiler, excepté à des fins d'interopérabilité avec des logiciels tiers ou tel que prescrit par la loi.

Les informations fournies dans ce document sont susceptibles de modification sans préavis. Par ailleurs, Oracle Corporation ne garantit pas qu'elles soient exemptes d'erreurs et vous invite, le cas échéant, à lui en faire part par écrit.

Si ce logiciel, ou la documentation qui l'accompagne, est livré sous licence au Gouvernement des Etats-Unis, ou à quiconque qui aurait souscrit la licence de ce logiciel pour le compte du Gouvernement des Etats-Unis, la notice suivante s'applique :

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Ce logiciel ou matériel a été développé pour un usage général dans le cadre d'applications de gestion des informations. Ce logiciel ou matériel n'est pas conçu ni n'est destiné à être utilisé dans des applications à risque, notamment dans des applications pouvant causer un risque de dommages corporels. Si vous utilisez ce logiciel ou ce matériel dans le cadre d'applications dangereuses, il est de votre responsabilité de prendre toutes les mesures de secours, de sauvegarde, de redondance et autres mesures nécessaires à son utilisation dans des conditions optimales de sécurité. Oracle Corporation et ses affiliés déclinent toute responsabilité quant aux dommages causés par l'utilisation de ce logiciel ou matériel pour des applications dangereuses.

Oracle et Java sont des marques déposées d'Oracle Corporation et/ou de ses affiliés. Tout autre nom mentionné peut correspondre à des marques appartenant à d'autres propriétaires qu'Oracle.

Intel et Intel Xeon sont des marques ou des marques déposées d'Intel Corporation. Toutes les marques SPARC sont utilisées sous licence et sont des marques ou des marques déposées de SPARC International, Inc. AMD, Opteron, le logo AMD et le logo AMD Opteron sont des marques ou des marques déposées d'Advanced Micro Devices. UNIX est une marque déposée de The Open Group.

Ce logiciel ou matériel et la documentation qui l'accompagne peuvent fournir des informations ou des liens donnant accès à des contenus, des produits et des services émanant de tiers. Oracle Corporation et ses affiliés déclinent toute responsabilité ou garantie expresse quant aux contenus, produits ou services émanant de tiers, sauf mention contraire stipulée dans un contrat entre vous et Oracle. En aucun cas, Oracle Corporation et ses affiliés ne sauraient être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'accès à des contenus, produits ou services tiers, ou à leur utilisation, sauf mention contraire stipulée dans un contrat entre vous et Oracle.

**Accès aux services de support Oracle**

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> ou le site <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> si vous êtes malentendant.