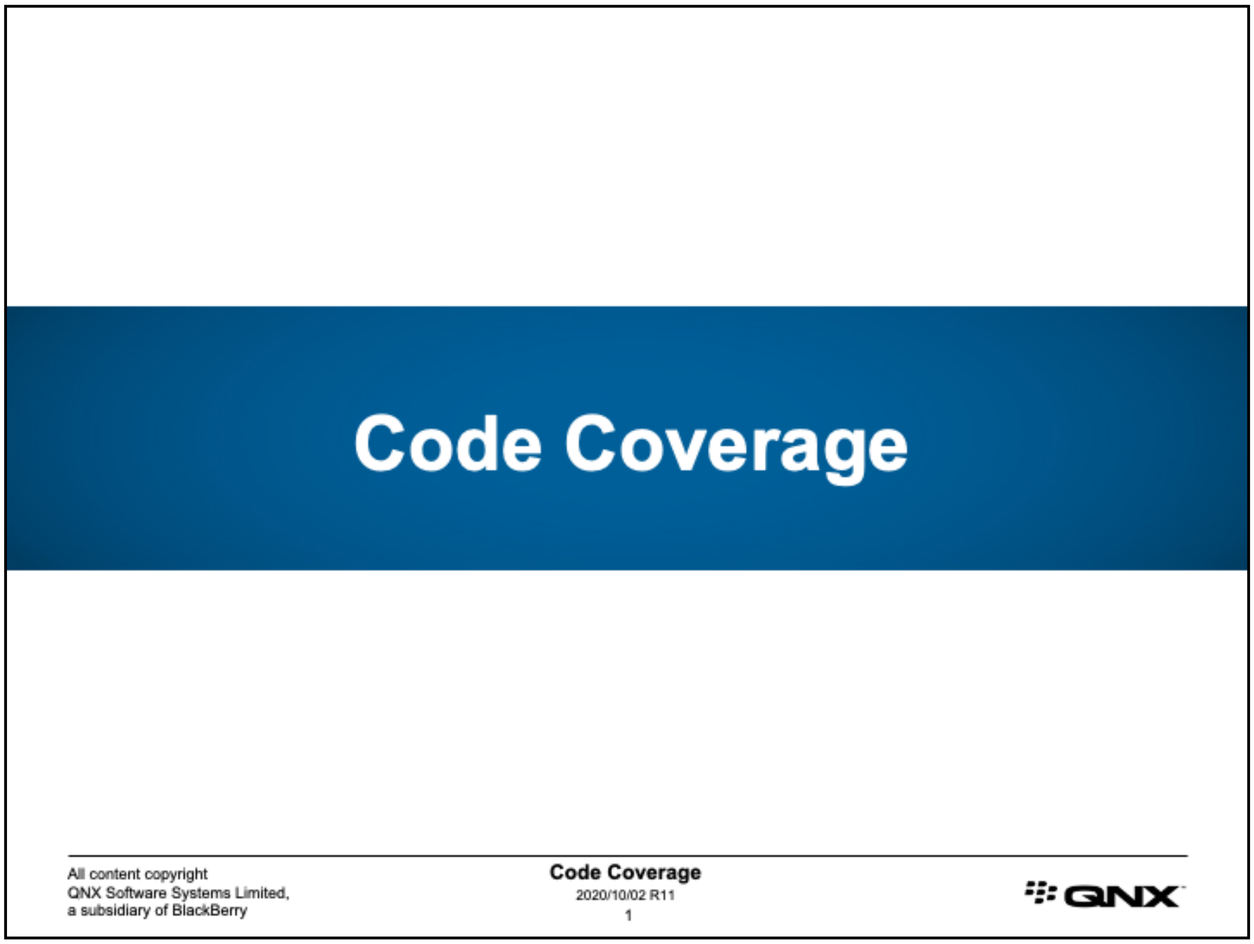
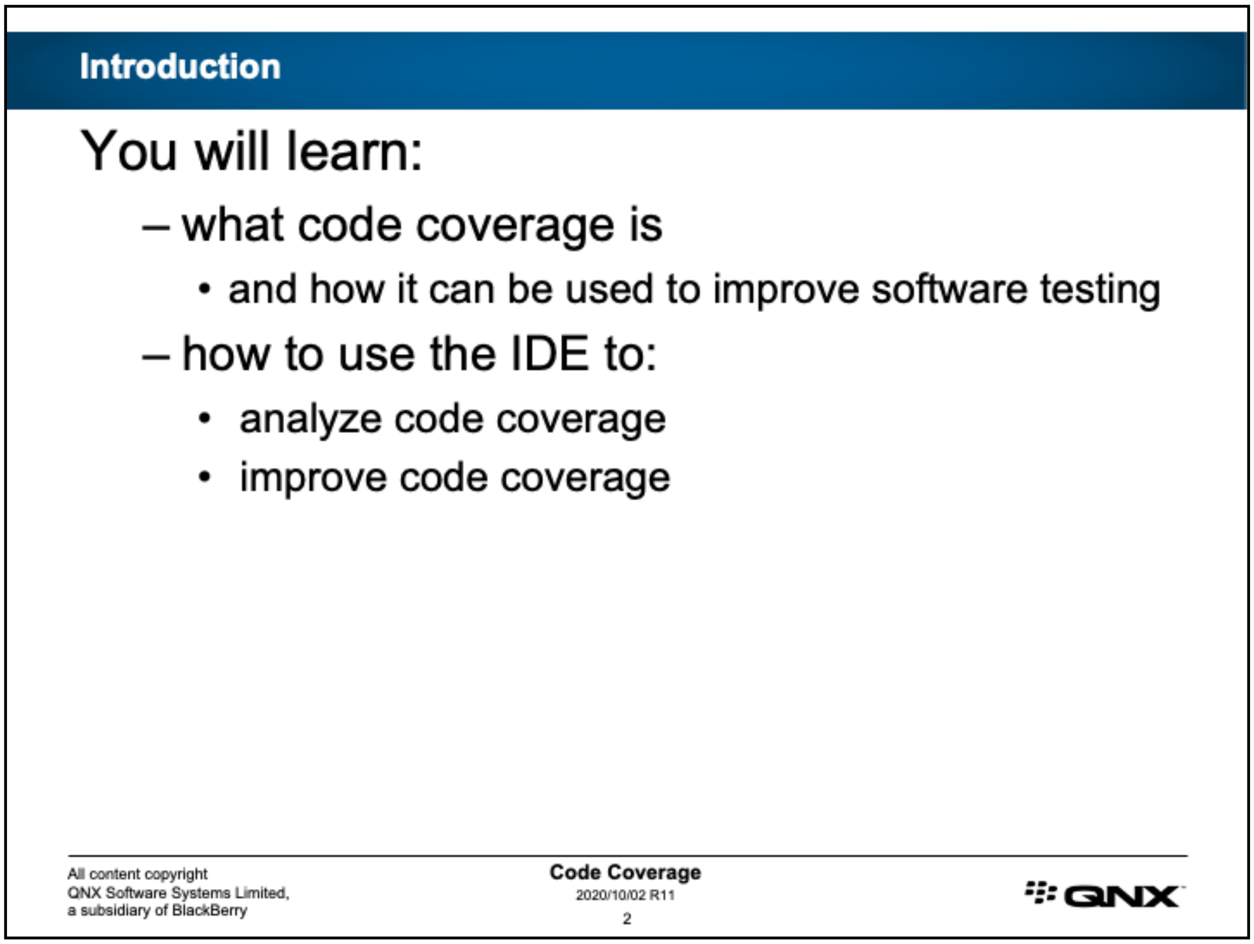


Code Coverage

NOTES:

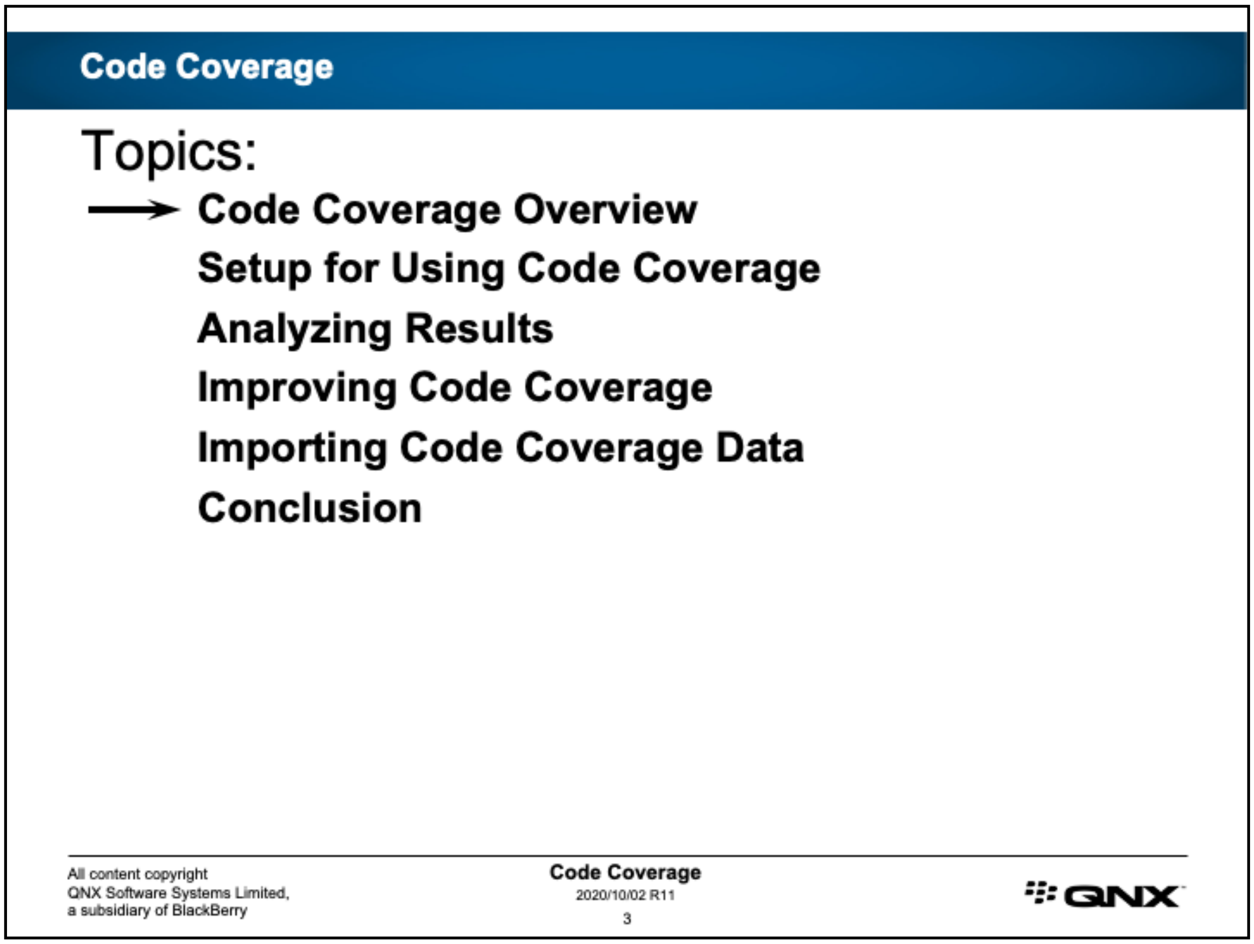


Introduction

You will learn:

- what code coverage is
  - and how it can be used to improve software testing
- how to use the IDE to:
  - analyze code coverage
  - improve code coverage

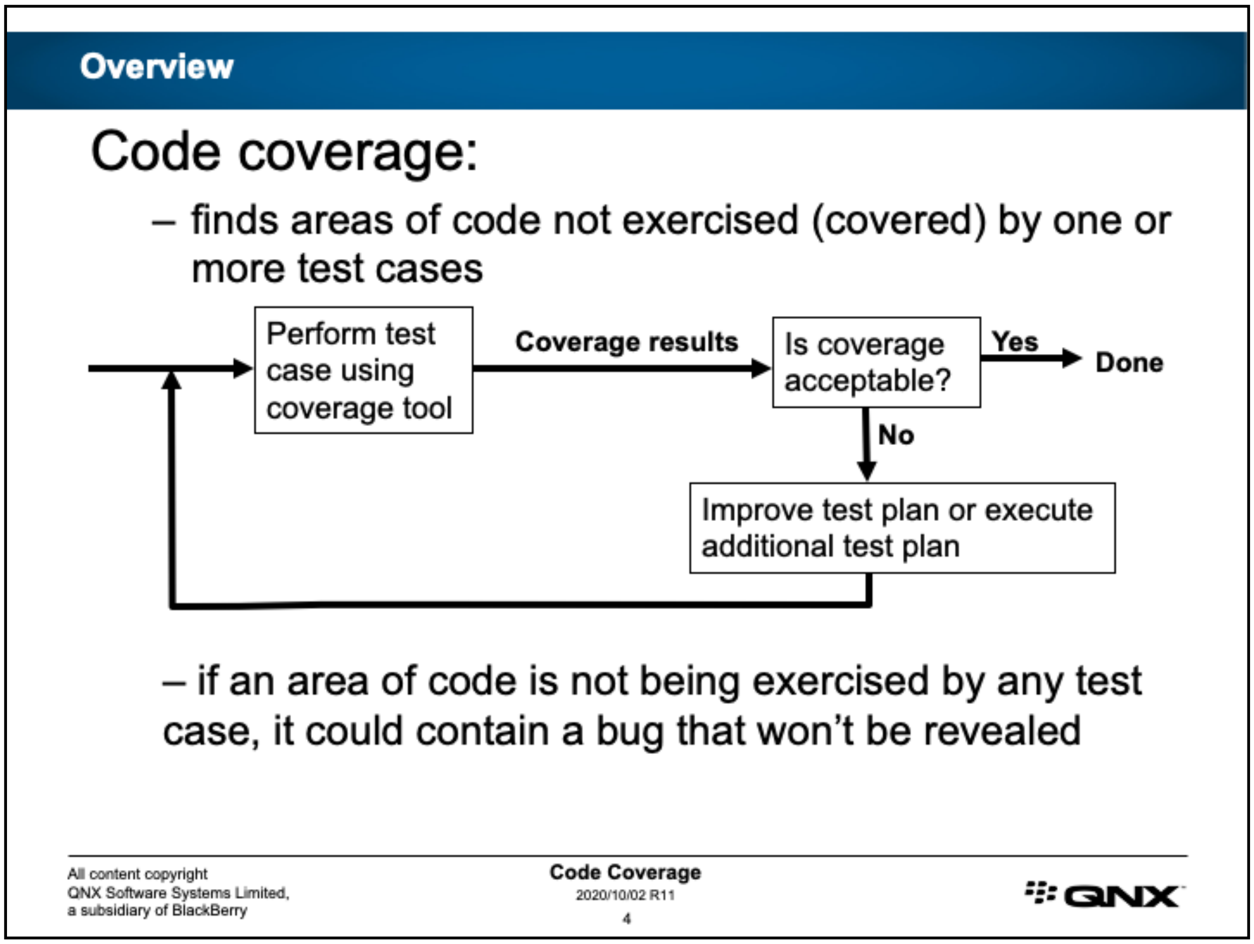
NOTES:



Code Coverage

Topics:

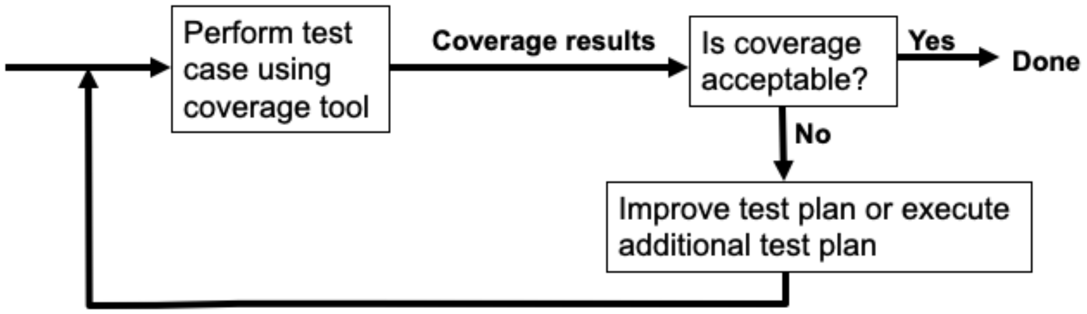
NOTES:



Overview

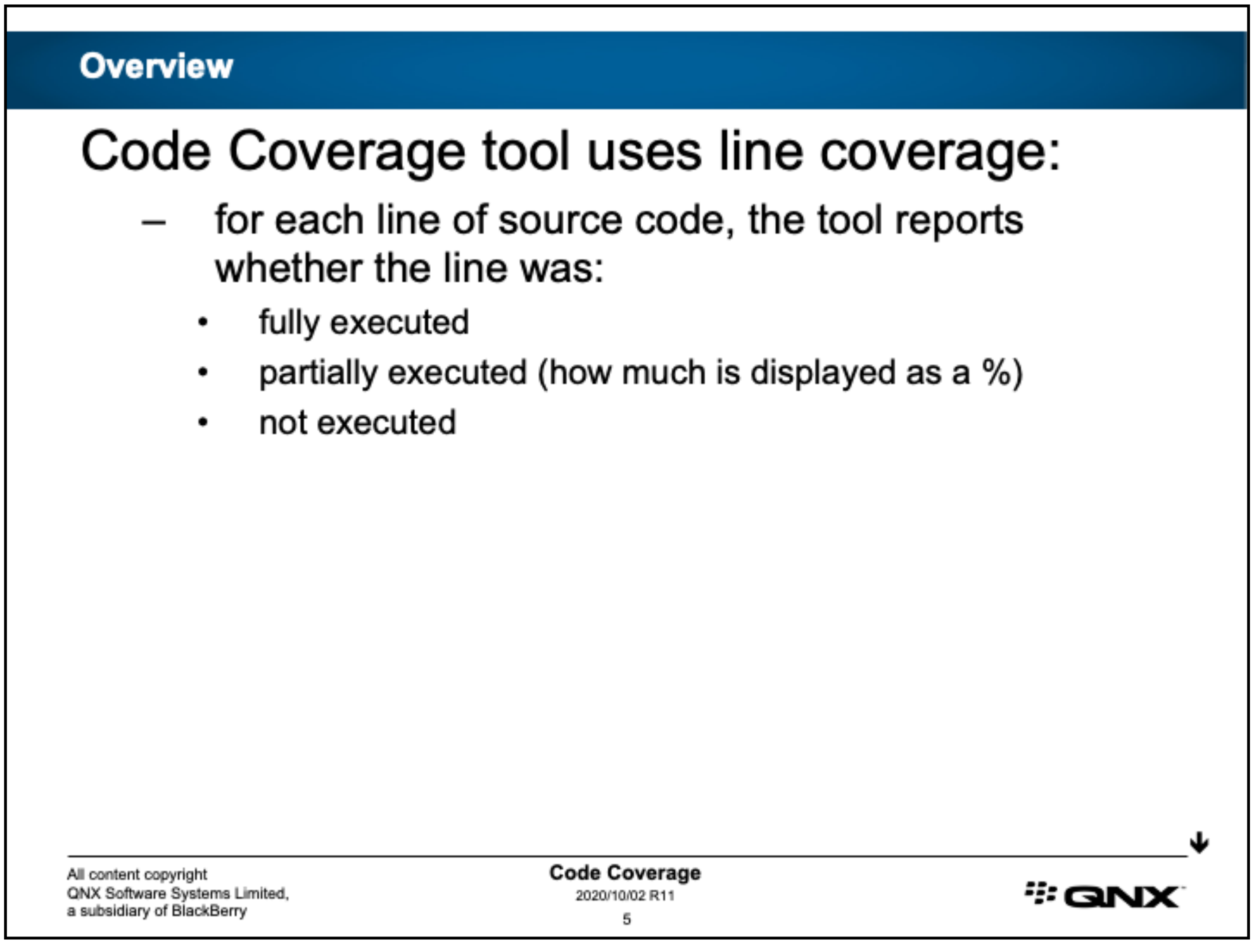
Code coverage:

– finds areas of code not exercised (covered) by one or more test cases


Perform test case using coverage tool
Coverage results
Is coverage acceptable?
Yes
Done
No
Improve test plan or execute additional test plan

– if an area of code is not being exercised by any test case, it could contain a bug that won't be revealed

NOTES:



Overview

Code Coverage tool uses line coverage:

- for each line of source code, the tool reports whether the line was:
  - fully executed
  - partially executed (how much is displayed as a %)
  - not executed

NOTES:

If every line of code were to be instrumented, both the executable size and the execution time would be adversely affected. Thus, only basic blocks are instrumented, and it is assumed that if execution begins in a basic block, then it will reach the end of the basic block. A basic block is a linearly executed region of code, with a single entry point, and one or more exit points. The first executable instruction in each of the following are examples of basic block entry points:

- function
- 'case:' in a switch/case decision
- body of a 'while' loop

A return statement is an example of a basic block exit point.

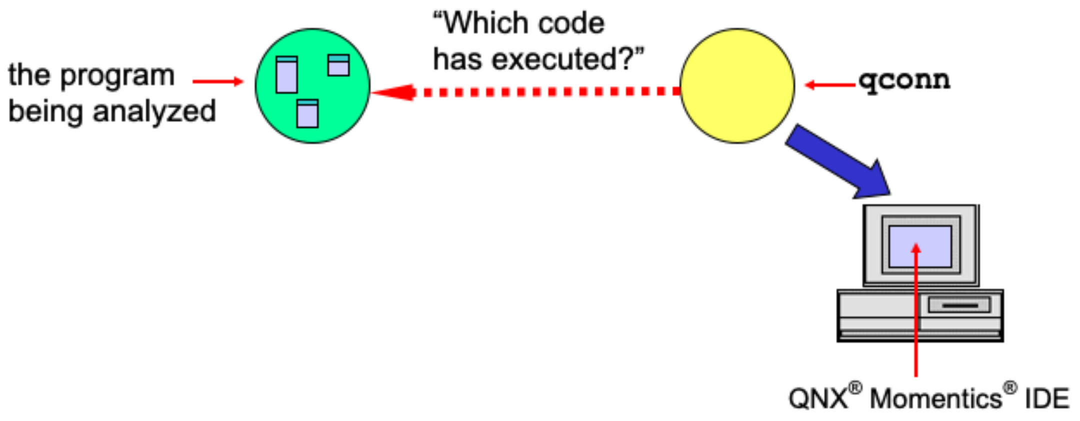
The assumption that if a basic block is entered, that all source code in that block has been covered, breaks in some circumstances, here are some examples:

- a signal, e.g. SIGSEGV is received
- a thread within the program has attached a hardware interrupt handler, and that particular interrupt has occurred, causing execution to asynchronously jump to the interrupt handler

Overview

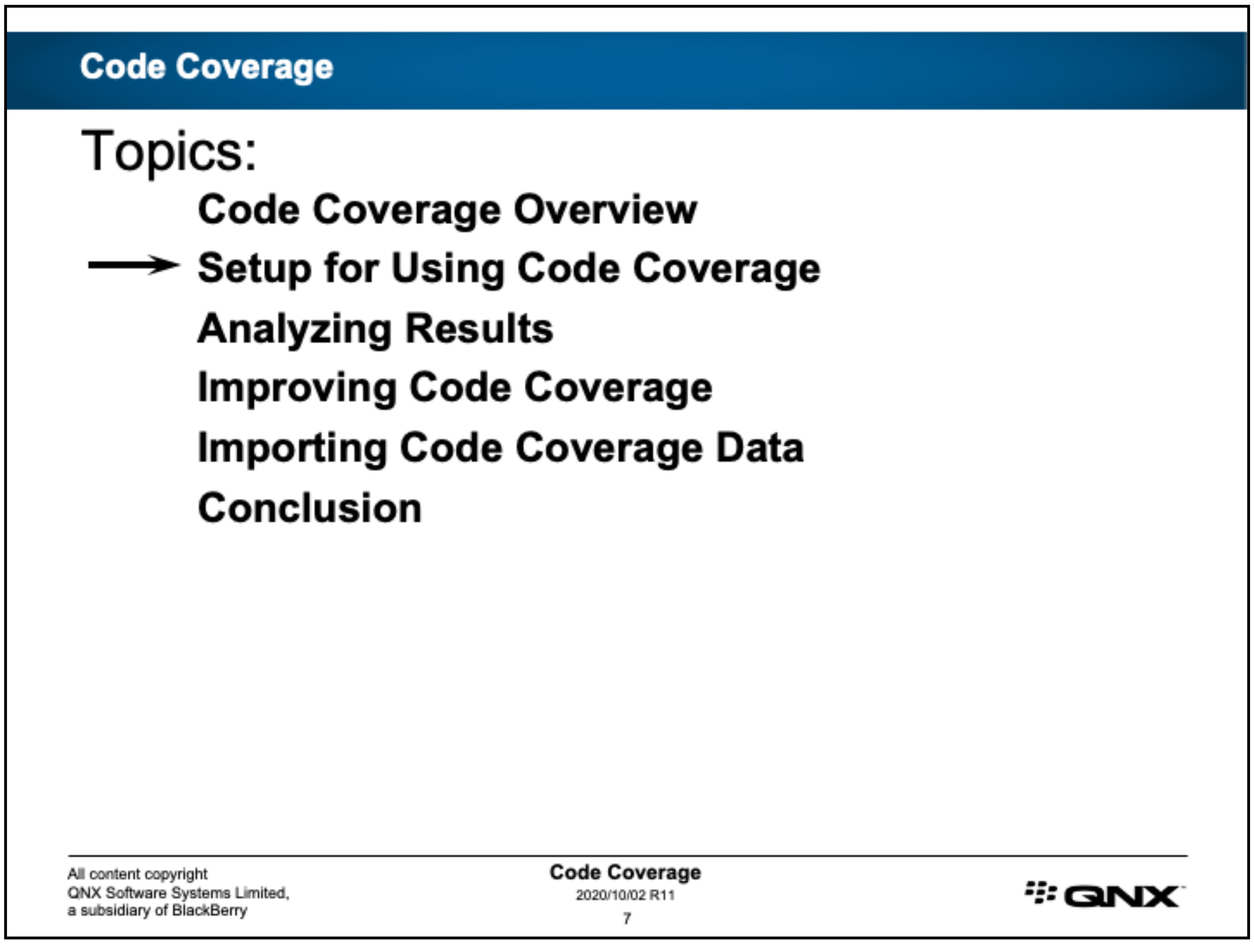
When doing code coverage:

- the compiler instruments the resulting executable, so that it will generate statistics on which lines were executed
- **qconn** collects these statistics and passes them back to the IDE on the host



Code Coverage
2020/10/02 R11

NOTES:



Code Coverage

Topics:

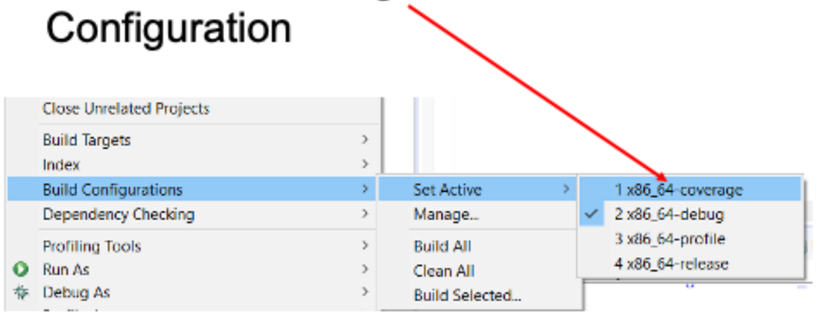
- Code Coverage Overview
- → Setup for Using Code Coverage
- Analyzing Results
- Improving Code Coverage
- Importing Code Coverage Data
- Conclusion

NOTES:

Setup for using Code Coverage - QNX Executable Project

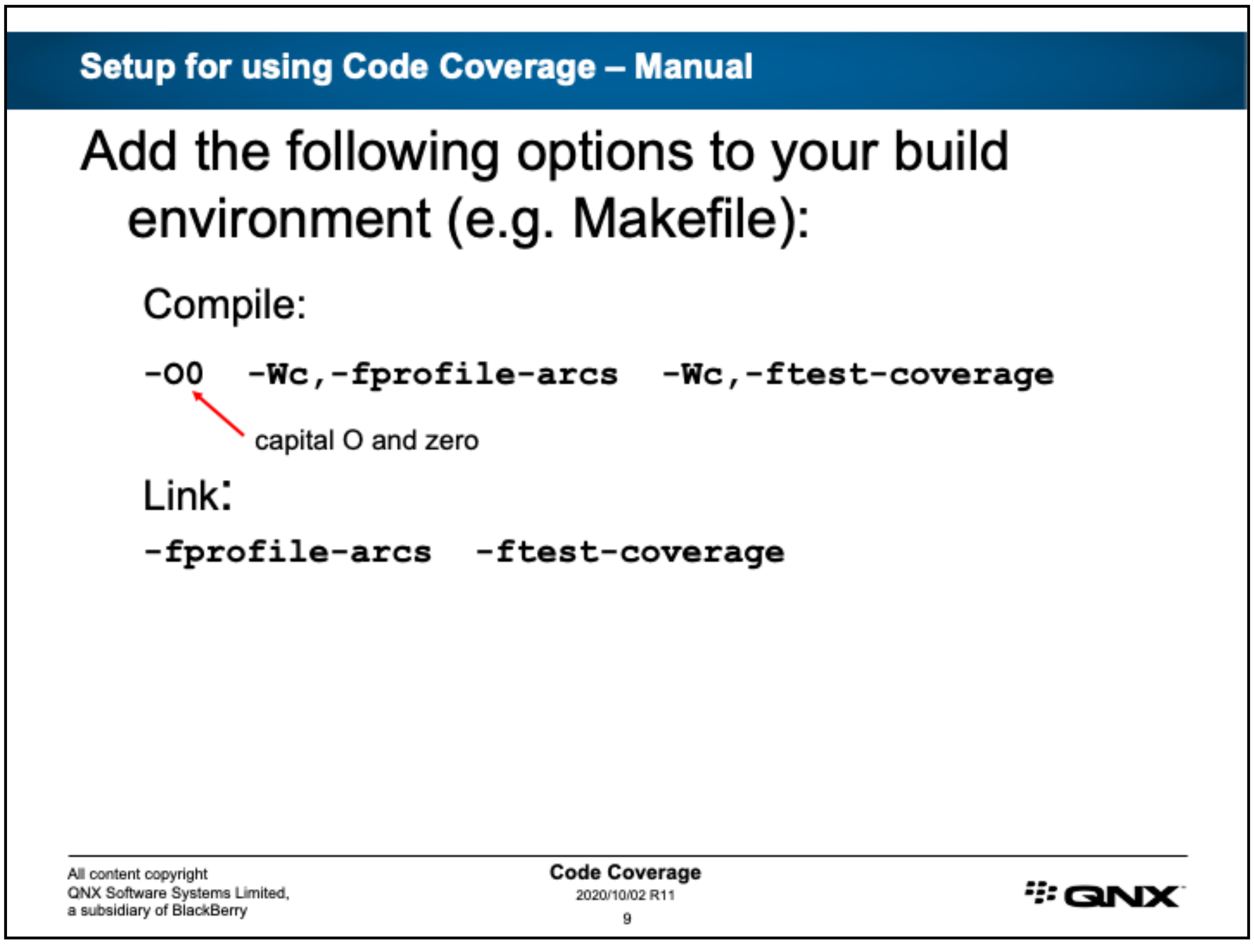
Existing project:

- right-click on the project in the Project Explorer view, choose Build Configurations
- select the “coverage” variant as the active Build Configuration



- and rebuild the project

NOTES:



Setup for using Code Coverage – Manual

Add the following options to your build environment (e.g. Makefile):

Compile:

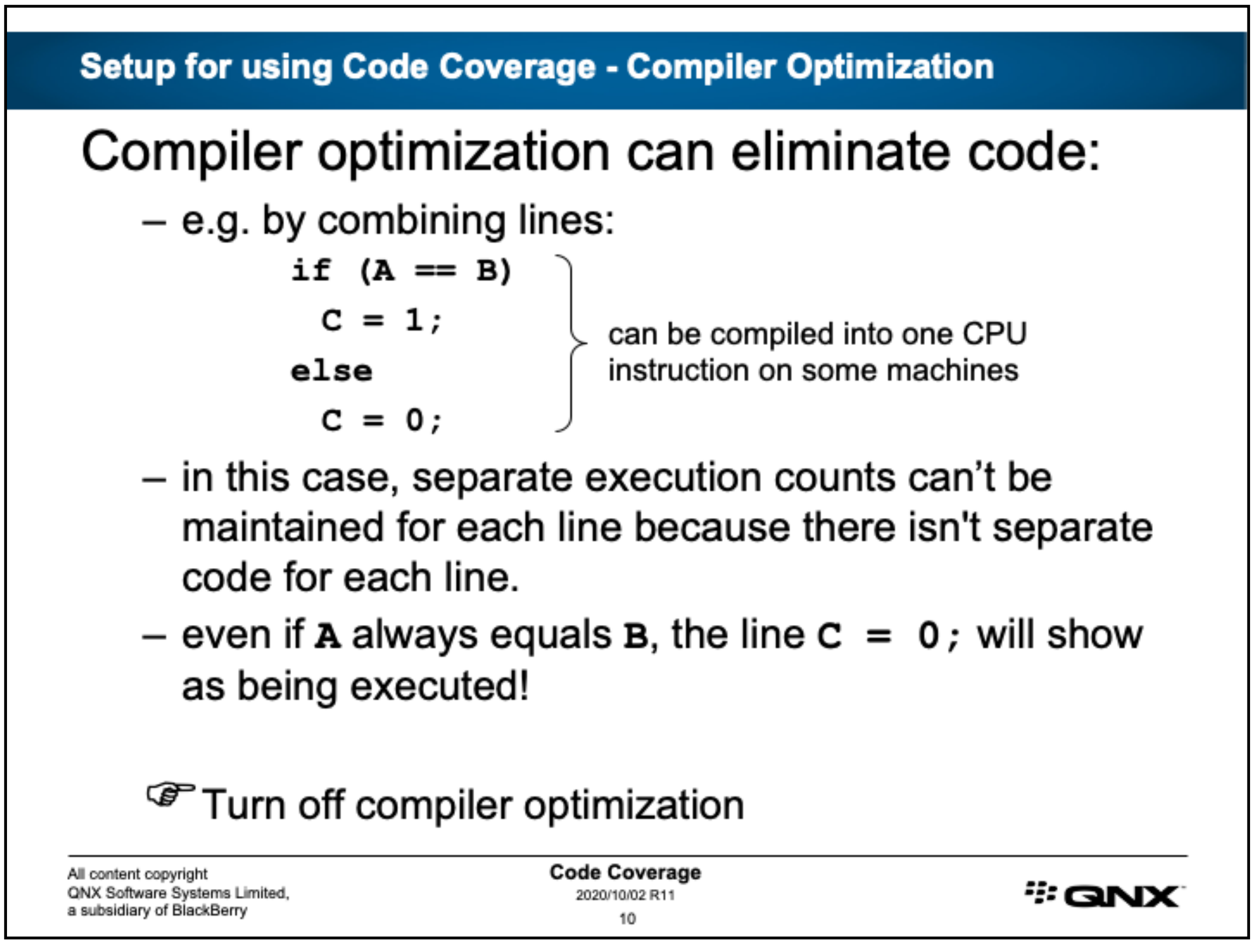
```
-O0  -Wc,-fprofile-arcs  -Wc,-ftest-coverage
```

capital O and zero

Link:

```
-fprofile-arcs  -ftest-coverage
```

NOTES:



Setup for using Code Coverage - Compiler Optimization

Compiler optimization can eliminate code:

- e.g. by combining lines:

```
if (A == B)
  C = 1;
else
  C = 0;
```

can be compiled into one CPU instruction on some machines

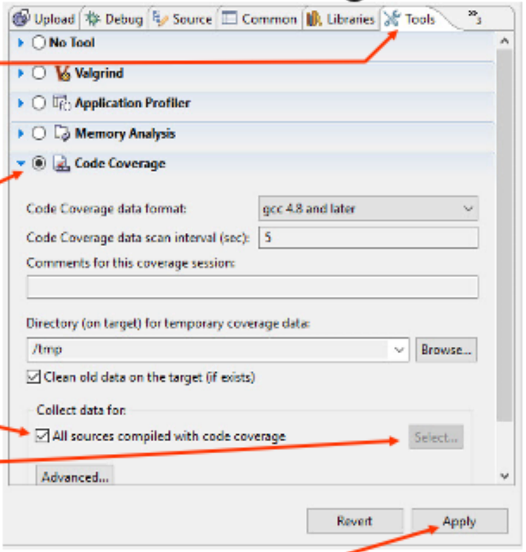
- in this case, separate execution counts can't be maintained for each line because there isn't separate code for each line.
- even if **A** always equals **B**, the line `C = 0;` will show as being executed!

☞ Turn off compiler optimization

NOTES:

Set up for Code Coverage - Create a launch configuration

Create a C/C++ QNX run launch configuration:



No Tool
Valgrind
Application Profiler
Memory Analysis
[✓ Code Coverage]

Code Coverage data format: gcc 4.8 and later

Code Coverage data scan interval (sec): 5

Comments for this coverage session:

Directory (on target) for temporary coverage data: /tmp

☑ Clean old data on the target (if exists)

Collect data for:

☑ All sources compiled with code coverage

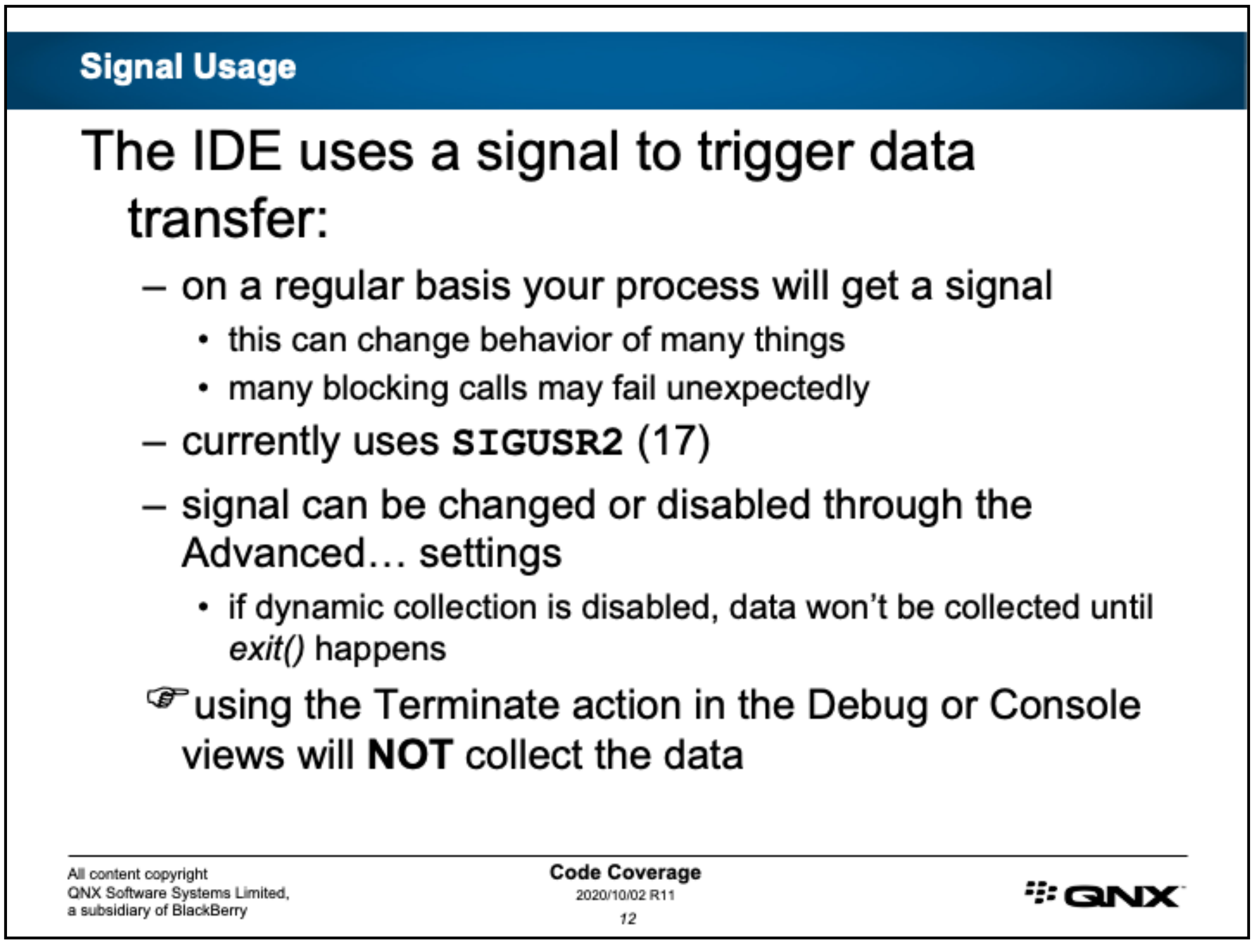
① in the tools tab

② select Code Coverage

if you've got multiple source files with coverage data, but only want data for some, uncheck here and select them.

③ Apply

NOTES:



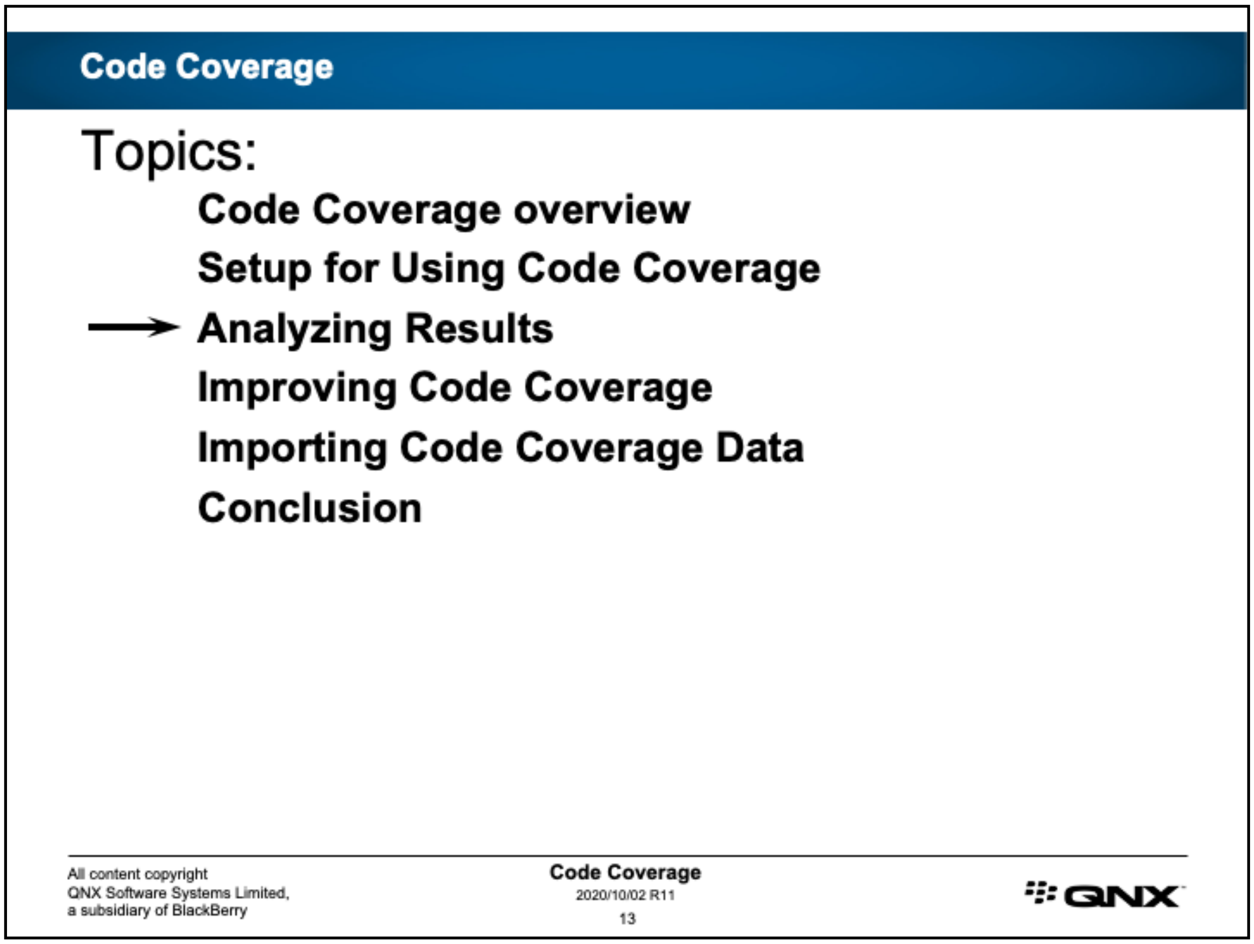
Signal Usage

The IDE uses a signal to trigger data transfer:

- on a regular basis your process will get a signal
  - this can change behavior of many things
  - many blocking calls may fail unexpectedly
- currently uses **`SIGUSR2`** (17)
- signal can be changed or disabled through the Advanced… settings
  - if dynamic collection is disabled, data won't be collected until *exit()* happens

☞ using the Terminate action in the Debug or Console views will **NOT** collect the data

NOTES:



Code Coverage

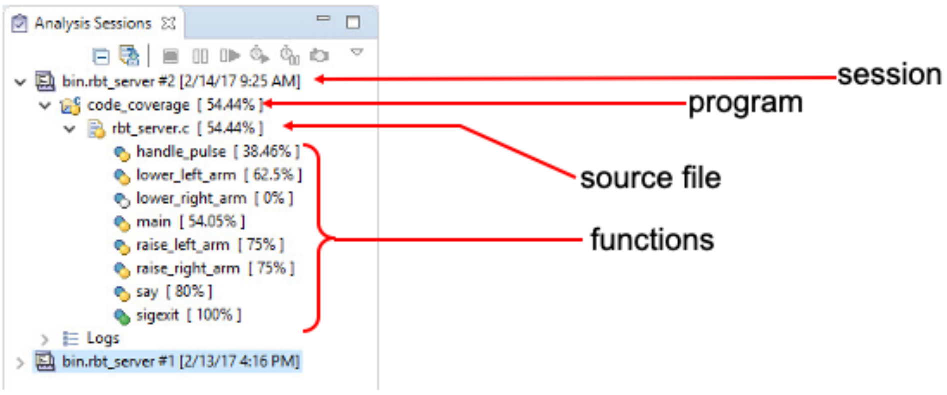
Topics:

- Code Coverage overview
- Setup for Using Code Coverage
- → Analyzing Results
- Improving Code Coverage
- Importing Code Coverage Data
- Conclusion

NOTES:

Analyzing Code Coverage

Open the QNX Analysis perspective:



```
Analysis Sessions
v bin.rbt_server #2 [2/14/17 9:25 AM]
  v code_coverage [ 54.44% ]
    v rbt_server.c [ 54.44% ]
        handle_pulse [ 38.46% ]
        lower_left_arm [ 62.5% ]
        lower_right_arm [ 0% ]
        main [ 54.05% ]
        raise_left_arm [ 75% ]
        raise_right_arm [ 75% ]
        say [ 80% ]
        sigexit [ 100% ]
  > Logs
> bin.rbt_server #1 [2/13/17 4:16 PM]
```

Quantities in square brackets, e.g. [54.55%], are coverage for:

- program
- source code file
- function

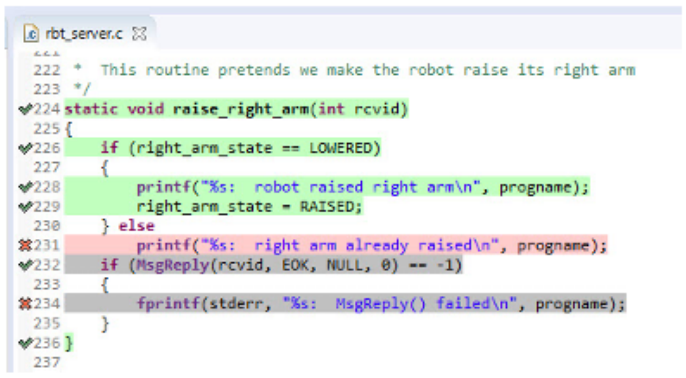
NOTES:

There is a session for each time a program is run.

Had the example program above consisted of multiple source files, each would be listed.

Analyzing Code Coverage

To display coverage markers in the source code, double-click a source file in Sessions view:



```
   * This routine pretends we make the robot raise its right arm
   */
✓ static void raise_right_arm(int rcvid)
  {
✓     if (right_arm_state == LOWERED)
      {
✓         printf("%s:  robot raised right arm\n", progname);
✓         right_arm_state = RAISED;
      } else
✗         printf("%s:  right arm already raised\n", progname);
✓     if (MsgReply(rcvid, EOK, NULL, 0) == -1)
      {
✗         fprintf(stderr, "%s:  MsgReply() failed\n", progname);
      }
✓ }
```

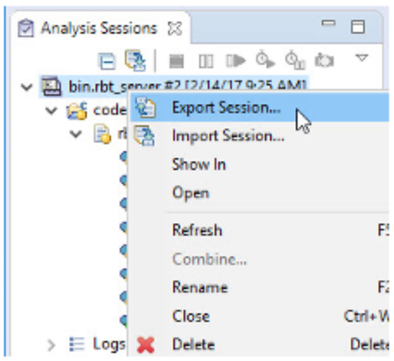
- ✓ (green check) - fully executed
- ● (yellow dot) - partially executed
- ✗ (red X) - not executed

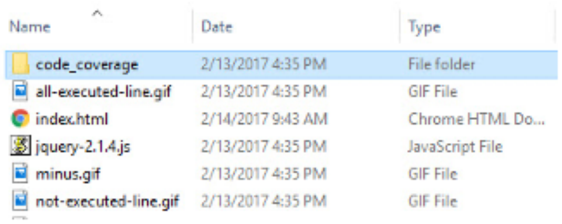
NOTES:

If you hover over a coverage marker that indicates that a line was partially executed, the “hover help” will display the percentage executed.

Anazlyzing Code Coverage

Export a Session:



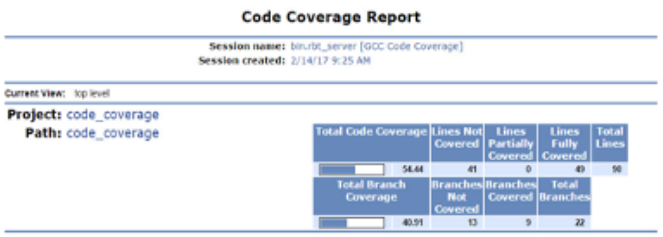


Name	Date	Type
code_coverage	2/13/2017 4:35 PM	File folder
all-executed-line.gif	2/13/2017 4:35 PM	GIF File
index.html	2/14/2017 9:43 AM	Chrome HTML Do...
jquery-2.1.4.js	2/13/2017 4:35 PM	JavaScript File
minus.gif	2/13/2017 4:35 PM	GIF File
not-executed-line.gif	2/13/2017 4:35 PM	GIF File

generates an HTML report with supporting files that can be viewed with most web browsers

export options include:

- source files
- branch coverage
- color coding



Code Coverage Report

Session name: bin.rbt_server [GCC Code Coverage]
Session created: 2/14/17 9:25 AM

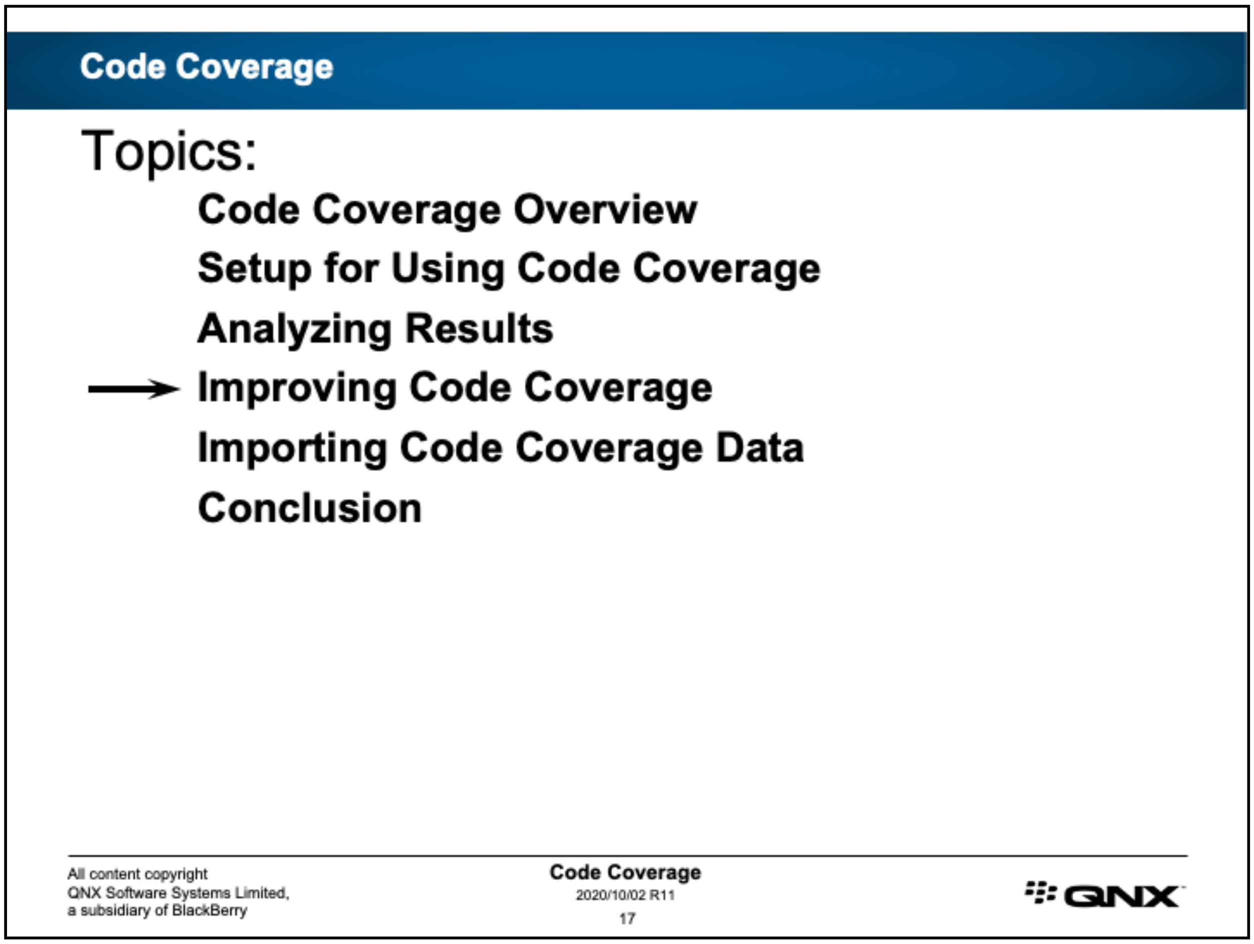
Current View: top level

Project: code_coverage
Path: code_coverage

Total Code Coverage	Lines Not Covered	Lines Partially Covered	Lines Fully Covered	Total Lines
54.44	41	0	49	90

Total Branch Coverage	Branches Not Covered	Branches Covered	Total Branches
40.91	13	9	22

NOTES:

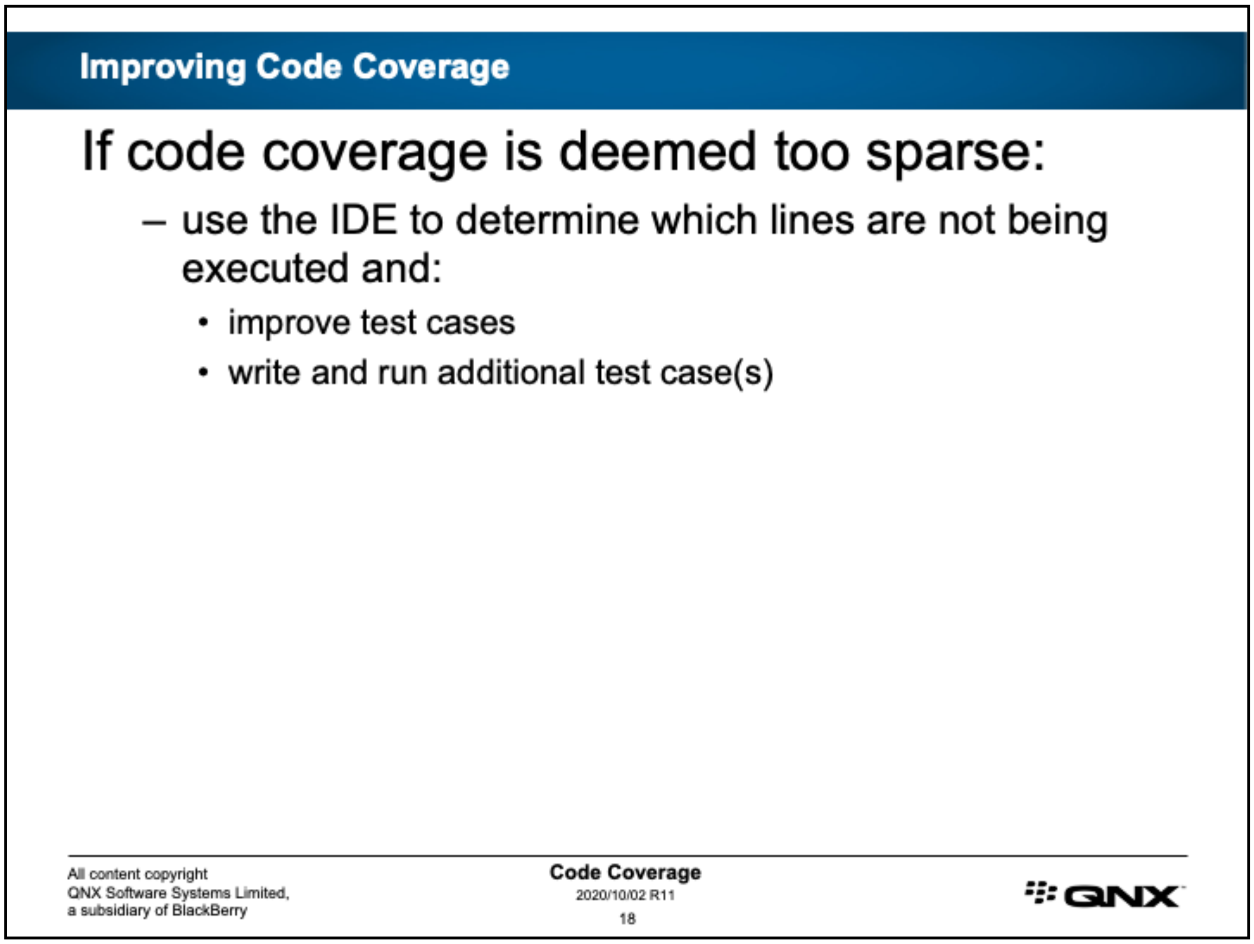


Code Coverage

Topics:

- Code Coverage Overview
- Setup for Using Code Coverage
- Analyzing Results
- → Improving Code Coverage
- Importing Code Coverage Data
- Conclusion

NOTES:



Improving Code Coverage

If code coverage is deemed too sparse:

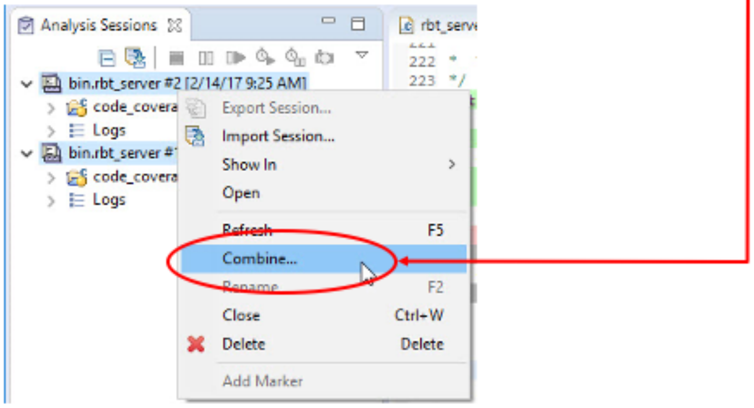
- use the IDE to determine which lines are not being executed and:
  - improve test cases
  - write and run additional test case(s)

NOTES:

Improving Code Coverage

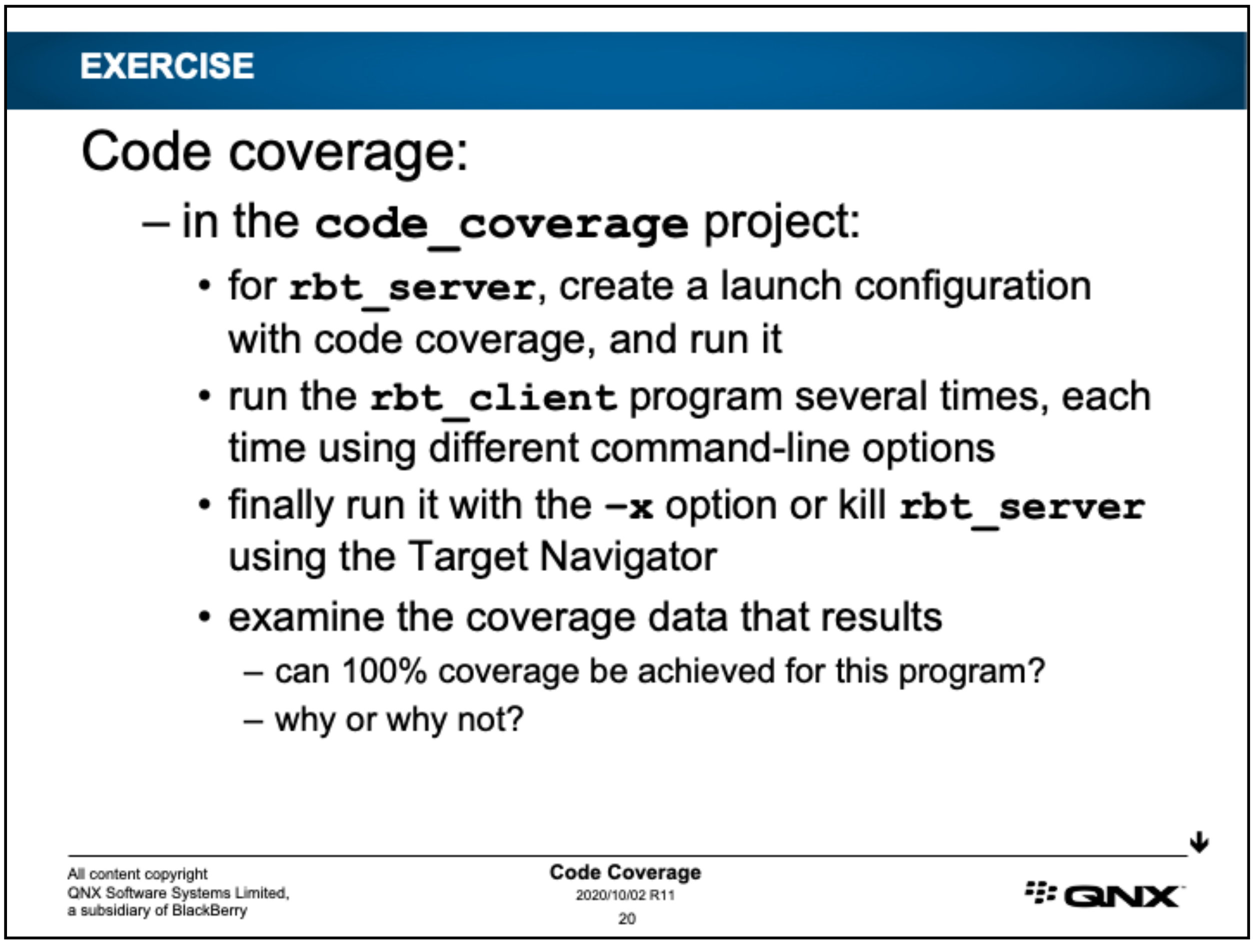
The Code Coverage tool can "Combine Sessions", to show cumulative coverage across multiple runs.

- hold down CTRL, select multiple sessions, then right-click, select



- this will show cumulative coverage for both instances when this program was run

NOTES:



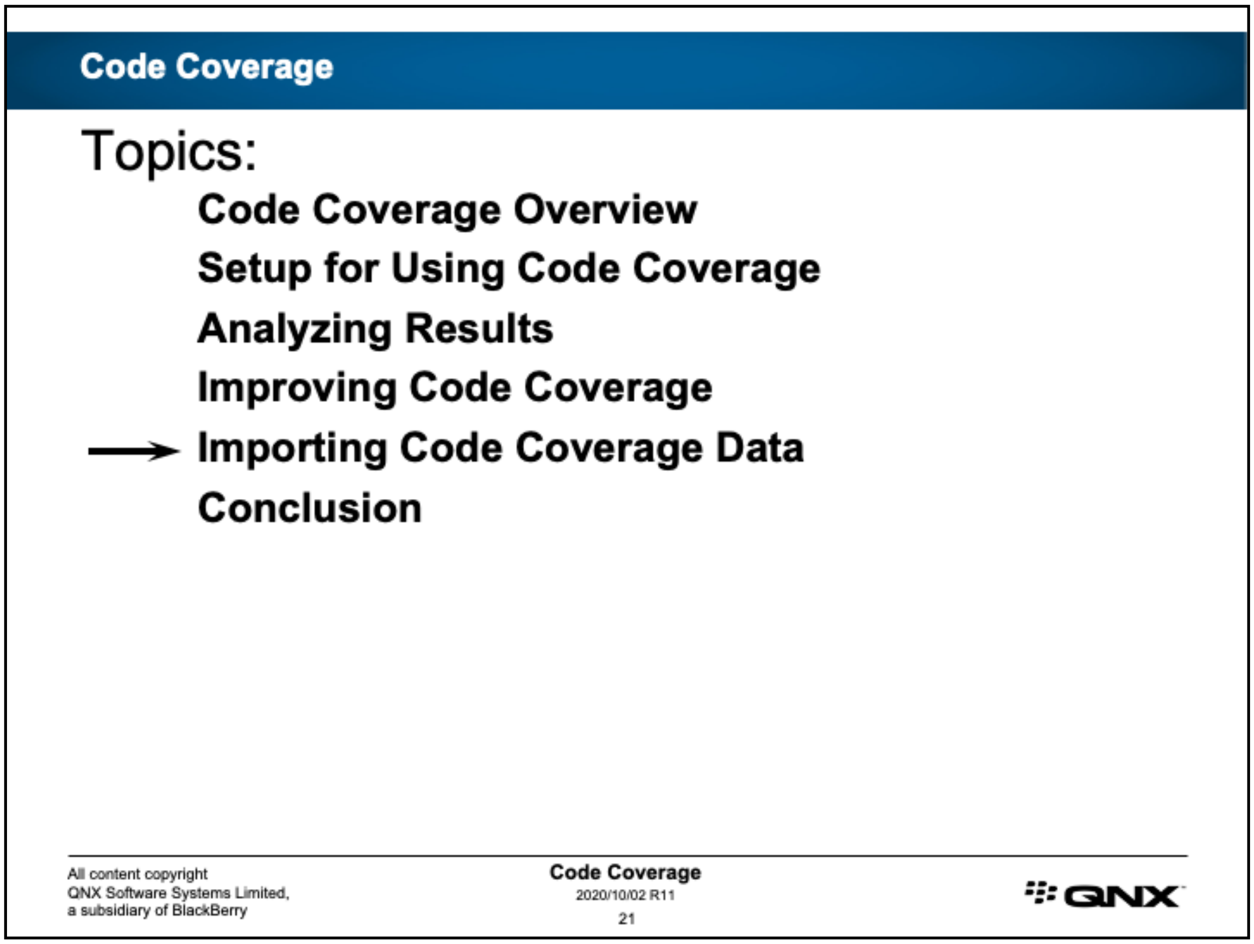
EXERCISE

Code coverage:

- in the **`code_coverage`** project:
  - for **`rbt_server`**, create a launch configuration with code coverage, and run it
  - run the **`rbt_client`** program several times, each time using different command-line options
  - finally run it with the **`-x`** option or kill **`rbt_server`** using the Target Navigator
  - examine the coverage data that results
    - can 100% coverage be achieved for this program?
    - why or why not?

NOTES:

There is no need to compile/launch the `rbt_client` test program with code coverage.

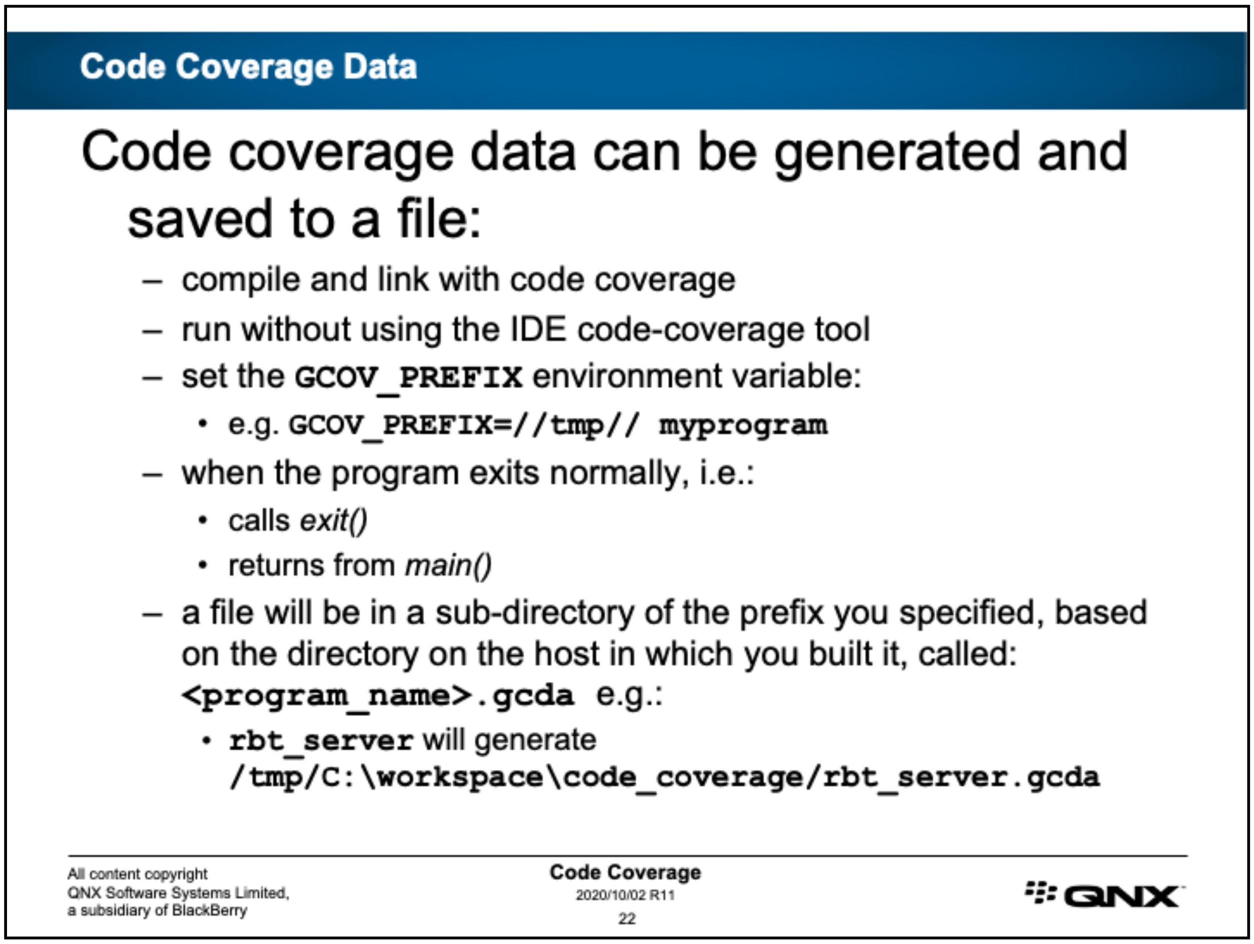


Code Coverage

Topics:

- Code Coverage Overview
- Setup for Using Code Coverage
- Analyzing Results
- Improving Code Coverage
- → Importing Code Coverage Data
- Conclusion

NOTES:



Code Coverage Data

Code coverage data can be generated and saved to a file:

- compile and link with code coverage
- run without using the IDE code-coverage tool
- set the **`GCOV_PREFIX`** environment variable:
  - e.g. **`GCOV_PREFIX=//tmp// myprogram`**
- when the program exits normally, i.e.:
  - calls *exit()*
  - returns from *main()*
- a file will be in a sub-directory of the prefix you specified, based on the directory on the host in which you built it, called: **`<program_name>.gcda`** e.g.:
  - **`rbt_server`** will generate **`/tmp/C:\workspace\code_coverage/rbt_server.gcda`**

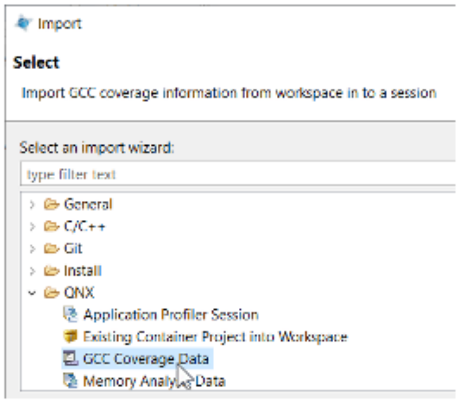
NOTES:

Importing Code Coverage Data

To import this code coverage data:

– select the project where you built the program

– then File->Import…->



Import

Select

Import GCC coverage information from workspace in to a session

Select an import wizard:

type filter text

– name it something descriptive, click Next a couple time then…

NOTES:

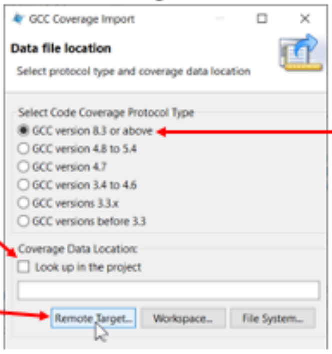
Importing Code Coverage Data

You can import directly from the target:

unselect this to manually find the data

select coverage protocol

then click Remote Target



GCC Coverage Import

Data file location

Select protocol type and coverage data location

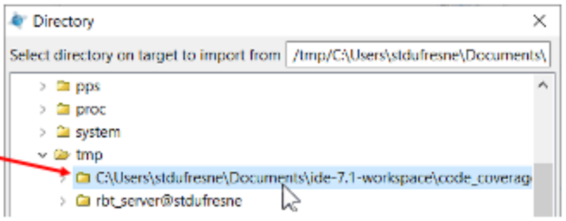
Select Code Coverage Protocol Type

- [✓ GCC version 8.3 or above]
- GCC version 4.8 to 5.4
- GCC version 4.7
- GCC version 3.4 to 4.6
- GCC versions 3.3.x
- GCC versions before 3.3

Coverage Data Location:

- [ ] Look up in the project

to find the directory created on the target



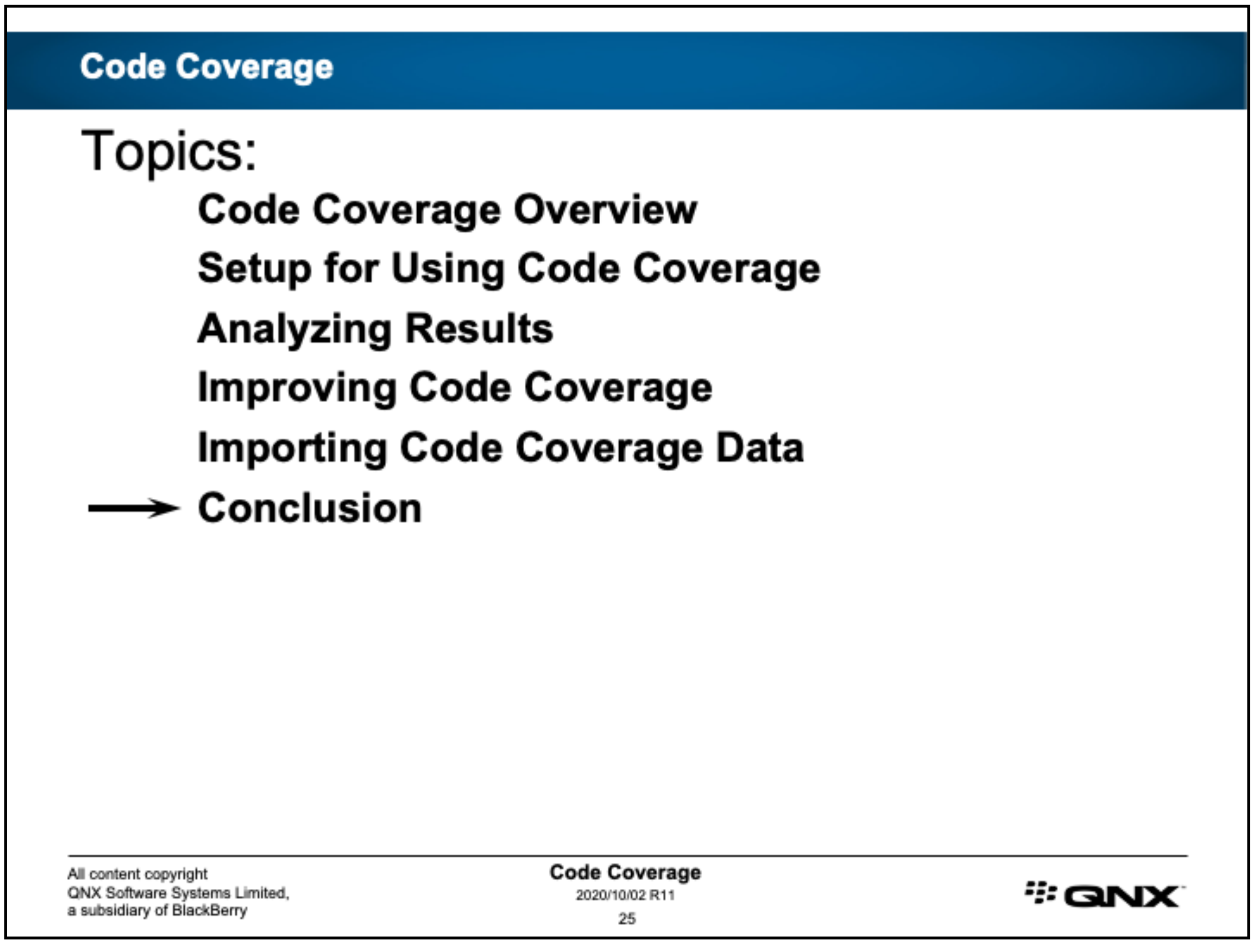
Directory

Select directory on target to import from /tmp/C:\Users\stdufresne\Documents\

pps
proc
system
tmp
C:\Users\stdufresne\Documents\ide-7.1-workspace\code_coverag
rbt_server@stdufresne

– and examine your code coverage

NOTES:

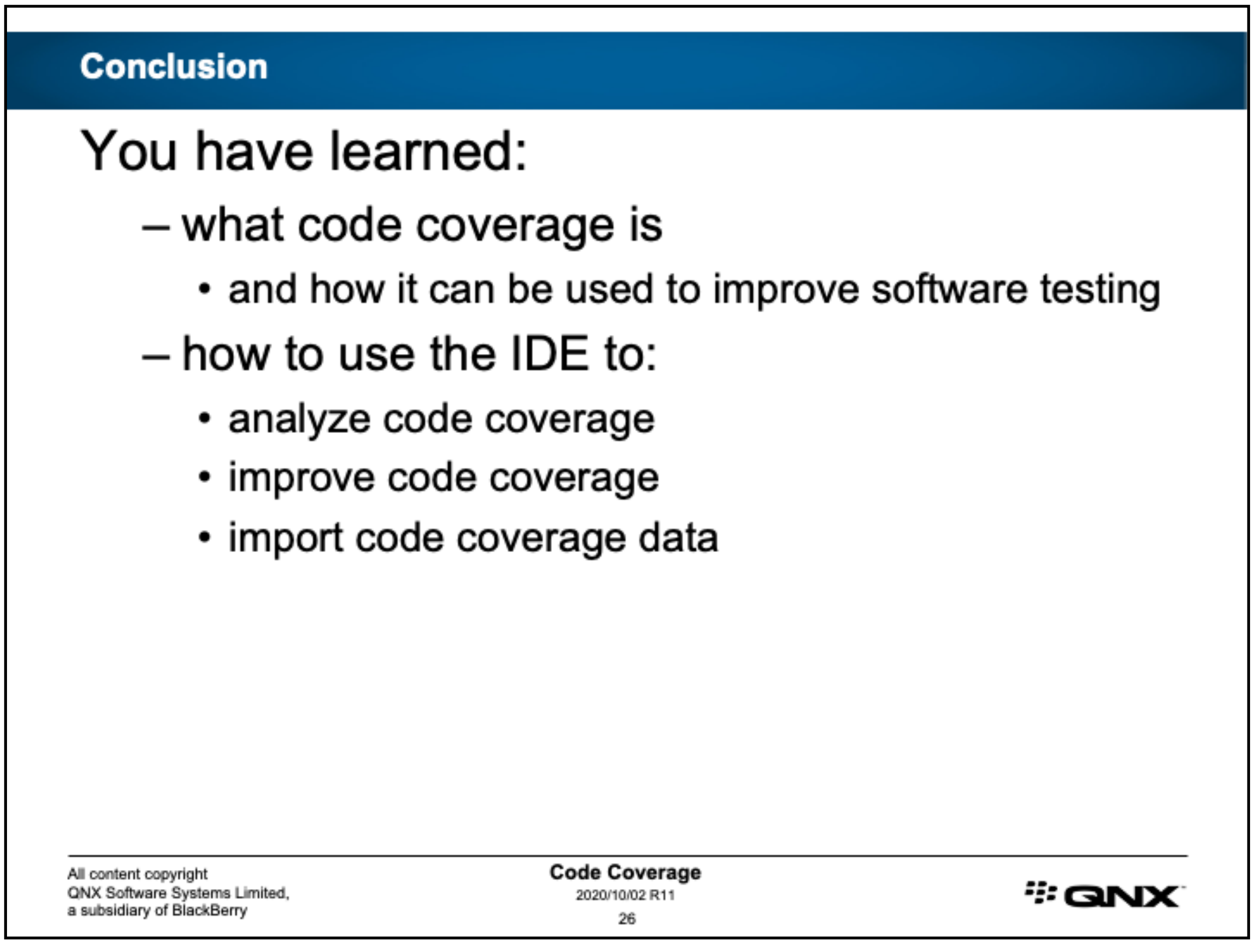


Code Coverage

Topics:

- Code Coverage Overview
- Setup for Using Code Coverage
- Analyzing Results
- Improving Code Coverage
- Importing Code Coverage Data
- → Conclusion

NOTES:



Conclusion

You have learned:

- what code coverage is
  - and how it can be used to improve software testing
- how to use the IDE to:
  - analyze code coverage
  - improve code coverage
  - import code coverage data

NOTES: