CODESCROLL
CONTROLLER TESTER

Controller Tester Troubleshooting Guides

3.6 — Last update: Dec 14, 2021

Suresofttech

Table of Contents

1. Controller Tester 문제 해결 가이드

Controller Tester를 사용하며 발생할 수 있는 일반적인 문제에 대한 해결 가이드 문서입니다.

1.1. 가변 배열 멤버에 값을 할당하는 방법

c99에서 추가된 가변 배열 멤버에 값을 할당하는 방법에 대해 소개합니다.

```
#include <stdio.h>

typedef struct _line {
        int size;
        int data[];
}line;

void funfun(line * abc){
        if(abc->data[0] == 1 ){
                printf("You can't touch this");

        }
        else{
                printf("MC Hammer");
        }

}
```

위와 같은 코드에서 참인 경우의 if 분기가 실행되도록 테스트를 설계하려면, 일반적인 테스트 데이터 입력 방법로는 값을 넣을 수 없습니다.
가변 배열 멤버(Flexible array member)이기 때문입니다.

따라서, 테스트 정보 탭의 호출 전 코드를 이용하여 아래와 같이 값을 넣어야 합니다.

```
size_t this_length = 5;
abc = (line *)malloc (sizeof (line) + this_length);
abc->data[0] = 1;
```

❗ malloc이 없는 환경에서는 위의 방식을 사용하실 수 없습니다.

1.2. 메모리가 부족할 때 힙 메모리 늘리는 방법

Eclipse RCP 기반으로 Java 언어로 작성되어 jvm 에서 실행되는 Controller Tester는 메모리가 부족할 때, 잦은 GC(Garbage collection)로 인해 성능이 저하되거나 메모리 부족으로 프로그램이 정상 동작하지 않을 수 있습니다.
이 경우, Controller Tester 64bit 패키지를 사용하거나(최대 힙 메모리 기본값 8G), 수동으로 힙 메모리 설정을 조정할 수 있습니다.

수동으로 힙 메모리 설정을 조정하는 방법은 아래와 같습니다.

1. {제품 설치 경로}\CodeScroll.ini 파일을 편집기로 엽니다(관리자 권한으로 수정 필요).
2. -Xmx 값을 증가시키면서 Controller Tester를 다시 실행합니다. 너무 큰 값을 입력하면 Controller Tester가 실행이 안됩니다. 조금씩 값을 낮추거나 늘리면서 최대값을 찾아서 설정합니다.

> Xmx의 최대값은 PC의 여유 메모리 상태에 따라 달라집니다. 일반적으로 4GByte 메모리를 사용하고 Controller Tester만 실행할 경우 약 1000MByte ~ 2000MByte 범위의 값을 가지게 됩니다.

1.3. 비정상 종료로 인해 프로젝트 DB 파일 (*.csp)이 손상된 경우

Controller Tester는 프로젝트 정보를 SQLite Database 파일에 저장합니다.

```
{워크스페이스 경로}\{프로젝트 이름}\.csdata\{프로젝트 이름}.csp
```

전원이 차단되거나 Windows 오류로 인해, 작업 도중 Controller Tester가 비정상 종료되는 경우에 DB가 손상될 수 있습니다.
이러한 경우, 손상된 DB 파일을 아래와 같은 방법으로 복구할 수 있습니다(항상 가능한 것은 아닙니다).

1. 아래 경로에서 명령 프롬프트 실행
    - `{제품 설치 경로}\plugins\com.codescroll.gp.core_1.0.2.201202152119\lib`
2. 명령 프롬프트에서 아래 명령 실행
    - `$ sqlite3 corrupted.db ".dump" | sqlite3 new.db`

1.4. QNX 소프트웨어를 테스트할 때 testrun.exe가 비정상 종료되는 경우

C++ 코드를 테스트할 때 전역변수가 class 타입이면, 해당 class의 생성자에서 비정상 종료되는 경우가 발생할 수 있습니다.

이 경우, 아래의 내용을 확인해야 합니다.

1. __get_errno_ptr() 함수가 스텁으로 만들어져 있다면, 스텁 바디에 아래 내용을 추가합니다.
    - `int* x = (int *)calloc(1, sizeof(int)); return x;`
2. debug 유틸이나 shm(공유 메모리) 관련 유틸을 사용하고 있다면, 파일 경로가 하드코딩되어 있지 않은지 확인이 필요합니다. 하드코딩되어 있다면 파일에 접근하는 시스템 함수들(fprintf 등)을 스텁으로 생성해야 합니다.
3. 초기화 시점에 exit 코드를 수행하는지 확인해야 합니다. 만약, exit 코드를 수행한다면 exit 코드를 스텁으로 생성해야 합니다.
4. 생성자 스텁에 throw가 있는지 확인해야 합니다. throw가 있다면 제거해야 합니다.

1.5. 커버리지 뷰에 특정 함수가 표시되지 않는 경우

커버리지 뷰에 함수가 표시되지 않는 경우와 해결방법은 아래와 같습니다.

커버리지 뷰에 함수가 표시되지 않는 경우	해결방법
해당 함수가 속한 소스 파일이 분석 제외 대상인 경우	프로젝트 특성의 [분석 대상] 또는 환경 설정의 [분석 제외 대상]에서 해당 소스 파일 또는 포함된 경로를 제거
해당 함수 또는 해당 함수가 속한 소스 파일이 커버리지 측정 제외 대상으로 선택된 경우	프로젝트 특성의 [커버리지 측정 제외]에서 해당 함수 또는 소스 파일을 제거
해당 함수가 실행이 되지 않은 경우	테스트가 정상적으로 실행이 됐는지 확인 또는 테스트 케이스 설계 내용 확인

1.6. 프로젝트 분석 실패 후 오류 메시지가 출력되지 않는 경우

프로젝트 생성 후 유닛 테스트를 생성할 때(분석), 별다른 오류 메시지 없이 실패하는 경우에는 아래의 내용을 확인하시기 바랍니다.

메시지 없이 분석 실패하는 경우	해결방법
Controller Tester가 관리자 권한으로 실행되지 않은 경우	관리자 권한으로 다시 실행
생성된 프로젝트의 경로 또는 테스트 대상 소스 파일의 경로 또는 이름이 매우 긴 경우	워크스페이스 경로 또는 테스트 대상 소스 파일의 경로를 가능한 짧은 경로로 수정
환경변수의 PATH 중에서 MYSQL과 관련된 경로가 있고, 해당 경로에 “&“가 포함되어 있는 경우	“&“가 있는 경우에 MYSQL 경로를 삭제한 후에 Controller Tester를 다시 시작

1.7. Windows 디스플레이 배율이 100%가 아닐 때, 화면이 잘리는 문제

Windows 디스플레이 배율을 100%가 아닌 값으로 설정한 경우, 아래와 같이 설정하면 화면이 잘리거나 이미지가 깨지는 문제를 해결할 수 있습니다.
(Controller Tester 3.1 버전부터 자동으로 반영됨)

1. 실행 파일(CodeScroll.exe) 또는 바로가기 파일을 오른쪽 클릭하고 [속성] 클릭
2. [호환성] 탭에서 높은 DPI 설정 변경 클릭
3. [높은 DPI 조정 재정의]에서 [높은 DPI 조정 동작을 재정의합니다.]를 선택
4. [조정한 사람]을 시스템으로 선택하고 확인

> 위와 같이 설정하면, Windows 시스템에서 자동으로 스케일을 조정하기 때문에 화면이 약간 뿌옇게 표시될 수 있습니다.

1.8. Windows.h 파일을 찾지 못하는 문제

Windows SDK 6.1을 설치하고 Visual Studio 2008 컴파일러를 사용할 때, 특정 환경에서 아래와 같은 오류 메시지가 출력되는 경우가 있습니다.

```
Cannot open include file: 'windows.h': No such file or directory
```

이 오류는 Visual Studio 빌드 환경을 설정하는 배치 파일에 누락된 내용이 있어서 발생합니다. 해결 방안은 아래와 같습니다.

C:\Program Files\Microsoft SDKs 디렉터리에서 windows.h 헤더 파일을 검색하여 찾습니다.
예) `C:\Program Files\Microsoft SDKs\Windows\v6.1\Include`

C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\bin\vcvars32.bat 파일을 에디터로 열어서 INCLUDE와 LIB, LIBPATH 변수에 SDKs\windows\v6.0A\include, SDKs\windows\v6.0A\lib 경로를 추가합니다.

```
예) 26라인  @set INCLUDE 에 추가할 경로: C:\Program Files\Microsoft SDKs\Windows\v
6.1\Include

@set INCLUDE=C:\Program Files\Microsoft SDKs\Windows\v6.1\Include;%VCINSTALLDI
R%\ATLMFC\INCLUDE;%VCINSTALLDIR%\INCLUDE;%INCLUDE%\
```

1.9. 로그에 INFO [ut.hio]: runTest:testrun exit code(105)가 출력되고 유닛 테스트 실행이 안되는 경우

{프로젝트 이름}_full.log에 INFO [ut.hio]: runTest:testrun exit code(105) 라는 메시지가 출력되고, 유닛 테스트 실행이 되지 않는 경우가 있습니다.

이 경우에는
`{워크스페이스}\{프로젝트 이름}\.csdata\build\testrun.log`
파일에 아래와 같은 오류 메시지가 있는지 확인합니다.
bc. error:add virtual address : memory alloc error [E000000-E1FFFFFF]
error:fail to init iohandle(105)

이 오류는 환경 설정의 [가상 메모리 주소]에 설정한 주소 범위가 유효하지 않을 때 발생합니다.

환경 설정의 [가상 메모리 주소]에서 현재 시스템에서 사용할 수 있는 메모리 주소 범위로 수정하시기 바랍니다.

1.10. “툴체인 정보 자동 추출“이 실패하는 경우

툴체인 추가 마법사에서 [툴체인 정보 자동 추출] 버튼을 눌렀을 때 실패하는 경우가 있습니다.
이 때는 아래와 같이 조치하면 문제를 해결할 수 있습니다.

- 툴체인 추가 마법사의 [툴체인 정보]에서 [환경 정보]에 컴파일러를 실행하기 위한 환경 정보 파일 (*.bat)을 입력합니다.
  - 예) Visual Studio cl 컴파일러의 경우 cl.exe 파일과 같은 경로에 있는 vcvars32.bat 파일을 입력합니다.
- 보안 프로그램(알약 등)에서 tce.exe 실행 파일을 차단했는지 확인하고, 검사 예외 목록에 추가합니다.
- 윈도우 환경 변수 중 ComSpec 변수에 아래의 기본값 외에 다른 값이 추가되어 있는지 확인하고, 있다면 기본값으로 수정합니다.
  - ComSpec 변수의 기본값: ComSpec=C:\Windows\system32\cmd.exe

1.11. 오류 뷰에서 메시지가 비정상적으로 표시될 경우

Controller Tester를 사용하는 환경에 따라 출력되는 오류 메시지의 인코딩이 달라져서 메시지가 비정상적으로 표시될 수 있습니다.
이러한 경우, 아래와 같은 방법으로 문제를 해결할 수 있습니다.

`{제품 설치 경로}\plugins\com.codescroll.gp.rcp.helios_1.0.0.201909240351\plugin_customization.ini`
파일을 편집기로 엽니다.

85라인의

```
#log file encoding (log plugin)
com.codescroll.gp.log/log.file.encoding=euc-kr
```

에서 euc-kr 을 현재 환경에 맞는 인코딩으로 변경한 뒤에 Controller Tester을 다시 시작합니다.

1.12. C++20 항목 포함된 소스 코드로 테스트를 수행하는 경우

현재 Controller Tester에서 사용하고 있는 분석기에서 C++20 항목 및 일부 신규 헤더를 지원하고 있지 않기 때문에, 해당 항목을 포함하는 소스 코드로 테스트를 수행하려는 경우 별도의 작업이 필요합니다.

1. *Controller Tester 글로벌 경로* \1.1\parserConfig\ *vs2019 툴체인 이름*.conf 편집
    - 마지막 줄에 “ms_c++20” 추가
2. *Controller Tester 설치경로* \plugins\com.codescroll.ut_3.3.2\config\cl.flag.txt 파일 편집
    - 항목의 마지막 줄에 “&/std:c++latest” 추가
    - 이 값은 신규 항목이 포함되지 않는 소스를 분석/수행 시 다시 지워줘야 합니다.

1.13. 테스트 실행 시 발생할 수 있는 에러

1. 기본 생성자를 사용할 수 없는 에러

예제코드

```
class A{
public:
      A(int a, int b){}
      testFunction();
}
```

- Controller Tester에서는 클래스 멤버 함수에 대한 테스트 생성시 기본 생성자로 인스턴스를 생성한 후에 함수를 호출하여 커버리지를 측정합니다. 하지만 위의 예제 코드를 보면 class A의 기본 생성자가 없어 인스턴스 생성시 에러가 발생하게 됩니다. 이와 같은 에러가 발생하면 테스트를 수정하여 적절한 생성자로 인스턴스를 생성하도록 수정해야합니다.

2. instance를 생성하지 못하는 에러

- 추상 클래스의 함수를 테스트로 생성하는 경우 추상 클래스로 인스턴스를 생성하고 생성된 인스턴스에서 함수를 호출하려고 하여 에러가 발생할 수 있습니다. 이런 경우 '테스트 정보'에서 해당 클래스를 찾은 후 생성자에서 '사용자 코드'로 바꾸어줍니다. 이 후 사용자 코드에서 추상 클래스를 상속받고 있는 클래스로 인스턴스를 생성하도록 수정해주시면 됩니다.

1.14. 통합 테스트 가져오기 후 전역 변수를 찾지 못하는 경우

기존에 잘 수행되던 통합 테스트 데이터를 가져오기 후 테스트 실행을 했을 때 특정 전역 변수에 대해 undefined reference 에러가 발생할 수 있습니다.
에러가 발생한 테스트의 ‘테스트 정보’에서 ‘전역 변수’ 클릭시 Not Found element 메세지가 출력이 된다면 코드 수정으로 인해 전역 변수의 TU가 달라졌을 수 있습니다.
테스트의 ‘설정’ 탭에서 연관 파일을 전역 변수의 TU가 바뀐 소스 파일로 변경해주시면 테스트가 정상 동작하는 것을 확인할 수 있습니다.

1.15. 테스트 실행시 “C2118 : 첨자가 음수입니다 “ 에러 발생시

visual studio 툴체인 혹은 변환 툴체인 사용시 발생할 수 있는 에러 메세지입니다.
모듈 특성의 ‘컴파일 플레그’에 추가 되어 있는 `/Zp` 옵션을 제거해주시면 됩니다.

1.16. Visual Studio 2015 툴체인 사용시 SDK 버전 문제

Visual Studio 2015 툴체인 사용시 SDK 버전이 8.1과 10.0.10240.0 이상이 같이 설치되어 있는 경우 테스트 실행이 정상적으로 수행되지 않을 수 있습니다.
Visual Studio 2015의 vcvars32.bat 환경 설정 스크립트가 최신 버전 sdk로 설정이 되기 때문에 발생하는 문제입니다.
모듈 특성의 '컴파일 플레그'에 /I 옵션으로 최신 sdk 버전의 시스템 헤더를 추가해주어야합니다.

ex) sdk 최신 버전이 10.0.18362.0인 경우
/I"C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\include"
/I"C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\atlmfc\include"
/I"C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\ucrt"
/I"C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\um"
/I"C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\shared"
/I"C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\winrt"
/I"C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\cppwinrt"

1.17. Controller Tester가 설치되지 않는 경우

설치 파일이 일반적인 디스크가 아닌 드라이브에 있어서 인스톨러가 접근할 수 없거나 보안 프로그램의 영향 등으로 설치가 실패하는 경우

1. 설치 파일(.msi 혹은 .exe)을 시스템 드라이브로 옮깁니다.
2. 파일을 우클릭한 후, 관리자 권한으로 실행합니다.

설치 파일이 전송 중 손상된 경우

설치 파일을 다시 받아서 설치합니다.

그 외

위 두 가지 방법을 시도해도 동일한 오류가 발생하는 경우에 다음과 같이 해보시기 바랍니다.

1. 명령 프롬프트(cmd.exe)를 열고 실행 파일이 있는 경로로 이동합니다.
2. 다음 명령어를 수행하여 설치 마법사를 동일하게 진행합니다.
   - .msi 파일로 설치하는 경우
     `msiexec /I "설치_파일_이름.msi" /L*V "install.log"`
   - .exe 파일로 설치하는 경우
     `"설치_파일_이름.exe" /L*v "install.log" /i`

3. 설치 마법사를 진행하면 해당 경로에 install.log 파일이 생성됩니다. install.log 파일에 다음과 같은 로그가 있는 경우 Windows Installer 실행 관련 오류입니다.
   ```
   MainEngineThread is returning 1603
   ```

4. 이 경우 우선 컴퓨터를 재부팅합니다.
5. 작업 관리자를 연 뒤 msiexec.exe 프로세스가 실행중이라면 종료시킨 뒤 Controller Tester를 다시 설치하시기 바랍니다.
6. 그래도 문제가 해결되지 않는다면 아래 Microsoft의 문제 해결 가이드대로 수행하시기 바랍니다.
   https://support.microsoft.com/ko-kr/help/17588/windows-fix-problems-that-block-programs-being-installed-or-removed
   https://support.microsoft.com/ko-kr/help/834484/you-receive-an-error-1603-a-fatal-error-occurred-during-installation

7. Microsoft의 문제 해결 가이드대로 수행하여도 문제가 해결되지 않는 경우 아래 내용을 기술지원 연락처로 보내주시기 바랍니다.
   - 설치를 시도한 패키지의 상세 정보
     - ex) Controller Tester (Host) 3.4.2 ×64
   - 명령 프롬프트로 설치하여 생성된 install.log 파일
   - 특정 PC에서만 발생하는지 여부
   - 문제가 발생한 PC의 환경
   - (Controller Tester 외에 사용하는 자사 제품이 있는 경우) 다른 제품을 설치할 때도 동일한 현상이 발행했는 지

기술 지원 연락처

- help@suresofttech.com

1.18. 대용량 제어 흐름 그래프를 이미지 파일로 저장하지 못하는 경우

그래프를 이미지 형태로 내보낼 때, Controller Tester는 임시로 메모리로 저장하여 내보냅니다. 이미지 크기가 클 경우(가용 메모리인 512M을 초과할 경우), Not enough memory 오류가 발생합니다. 이 경우 3가지 방법으로 그래프 이미지를 얻을 수 있습니다.

- 설치 경로의 CodeScroll.ini 파일의 -Xmx 설정을 사용하는 시스템의 최대값으로 바꾼다.
  - `-Xmx1200m`
- 이미지 파일로 내보내지 않고 시스템 클립보드에 복사한 뒤, 다른 프로그램에서 붙여넣는다.
  - 이 방법은 이미지로 내보내는것 보다 큰 사이즈도 가능합니다.
- 그래프 포맷(ex. ygf)으로 내보낸 뒤에, yEd 같은 그래프 편집기에서 해당 그래프를 열고 클립보드에 복사하거나 pdf 형태로 내보낸다.

1.19. 워크스페이스가 망가진 경우

워크스페이스가 망가진 경우 아래와 같은 과정을 통해 문제를 해결할 수 있습니다.

1. 새 워크스페이스를 생성하고 기존과 동일한 이름의 프로젝트를 생성한 후 Controller Tester를 종료합니다.
2. 윈도우 탐색기에서 기존 프로젝트 디렉토리의 모든 내용을 복사한 후 새로운 프로젝트의 디렉토리에 붙여넣습니다.
3. Controller Tester를 다시 실행합니다.

1.20. VPES 업로드 시 발생할 수 있는 문제

VPES에 업로드할 때 다음과 같은 문제가 발생할 수 있습니다.
프로젝트에 변경 사항이 있습니다. 프로젝트를 수행한 후 다시 시도해주세요.
이 경우 프로젝트를 재분석한 후 다시 시도하시기 바랍니다.

1.21. 매트릭 뷰 관련 오류

매트릭 뷰에서 오류가 발생하는 경우 다음과 같이 해보시기 바랍니다.

1. Controller Tester을 종료합니다.
2. `C:\Users\사용자_이름\AppData\Roaming\CodeScroll\1.1\metric_messages` 디렉토리를 삭제합니다.
3. Controller Tester을 실행합니다.

1.22. 지역 변수의 값을 확인하고 테스트 케이스 결과에 반영하는 방법

소스 코드의 특정 지점에서 지역 변수의 값에 따라 테스트 케이스의 성공/실패 여부를 결정하기 위해 결함 주입(기존의 코드 삽입)과 테스트 매크로를 사용할 수 있습니다. 이 방법을 이용하면 제한적으로 지역 변수에 대한 검증을 할 수 있습니다.

결함 주입 뷰에서 지역 변수의 값을 확인하고자 하는 위치에 다음과 같은 코드를 삽입합니다.

```
if(CS_TESTCASENO() == 1) {
        CS_ASSERT(temp == 0);
}
```

코드를 삽입하고 테스트를 실행하면 1번 테스트 케이스에서 해당 부분의 temp의 값이 0이 아닐 경우 테스트 케이스가 실패하게 됩니다.

1.23. 함수 호출 커버리지와 함수 커버리지의 결과가 맞지 않는 경우

Controller Tester에서 함수 호출 커버리지는 100%이지만 함수 커버리지가 100%가 아닌 경우가 있습니다. 이 차이는 함수 커버리지와 함수 호출 커버리지를 계산하는 방법에 차이가 있기 때문에 발생합니다. 함수 커버리지의 경우, 테스트 수행 중에 해당 함수가 실제로 호출 되었는지를 계산합니다. 함수 호출 커버리지는 해당 함수를 호출하는 코드 블록(제어 흐름 그래프의 노드)의 수행 여부를 계산합니다.

ex) `if ( a()==1 || b()==1 )`

위와 같은 코드가 있는 경우 함수 호출 커버리지와 함수 커버리지가 맞지 않는 경우가 생깁니다. `a()==1`이 true인 경우, `b()`를 호출하지 않고 넘어갑니다. 이 때, `a()`와 `b()`가 같은 코드 블록이기 때문에 함수 호출 커버리지는 `2/2`로 표시되지만 실제로 호출된 것은 `a()`뿐이므로 함수 커버리지는 `1/2`로 표시됩니다.

1.24. 실수형 변수의 기대값, 출력값 관련 문제

실수형 변수의 기대값과 출력값이 달라도 테스트 케이스가 실패하지 않는 경우

Controller Tester에서 실수형 변수의 허용 오차는 소수점 아래 6번째 자리로 설정되어 있습니다. 예를 들어 기대값이 3.14159265이고 출력값이 3.14159274인 경우 Controller Tester는 기대값과 출력값이 같다고 판단하여 테스트 케이스가 성공하게 됩니다. 아래의 방법으로 허용 오차를 바꿀 수 있습니다.

- `%프로젝트 경로%\.csdata\ut.ini` 파일을 열고 `FLOAT_TOLERANCE`의 값을 바꿉니다.
  - 위 예시에서는 `FLOAT_TOLERANCE=0.0000001`로 수정하면 됩니다.

float 변수의 기대값과 출력값이 같아도 테스트 케이스가 실패하는 경우

Controller Tester가 float 변수를 처리할 때 double로 캐스팅하므로 오차가 발생할 수 있습니다. 이러한 문제는 두 가지 방법으로 해결될 수 있습니다.

1. float로 선언된 변수를 double로 선언한다.
2. codescroll_flt 타입을 float으로 바꾼다.
   - '환경 설정' > '툴체인' > '설정 디렉터리 열기'에서 사용중인 툴체인의 .info 파일을 열어 다음과 같이 수정하면 됩니다.
3. 2번 수정 후 '환경 설정' > '툴체인' > 해당 툴체인 선택 > '툴체인 편집' > '적용'

```
// 수정 전
#typeName,valueKind,min,max,size,csType
long double,float,2.22507e-308,1.79769e+308,8,codescroll_flt

// 수정 후
#typeName,valueKind,min,max,size,csType
float,float,2.22507e-308,1.79769e+308,8,codescroll_flt
```

상수를 이용하여 실수형 변수의 값을 연산할 때 연산 결과가 바르지 않은 경우

계산식에서 상수를 이용하면 Controller Tester는 실수를 정수로 바꾸어 계산합니다. 상수를 이용한 계산을 할 경우 실수를 정수로 변환하지 않도록 옵션을 바꿔야 합니다.

- '환경 설정' > '툴체인' > '설정 디렉터리 열기'에서 사용 중인 툴체인의 .ini 파일을 열어 다음과 같이 수정하면 됩니다.
  `leave_float_literal = 0` -> `leave_float_literal = 1`

```
// 예제 코드
float sampleFunction(){
    float test_a = (1.0F/300.F);
    float test_b = (1.0/300.0);
    return 0;
}
```

```
//leave_float_literal = 0 옵션이 켜진 경우
float sampleFunction(){
    float test_a = (1/300);
    float test_b = (1/300);
    return 0;
}
```

이와 같은 문제는 RTV 프로젝트나 RTV 타깃 프로젝트를 이용하는 경우에 발생합니다.

1.25. COVER의 커버리지 가져오기가 실패하는 경우

COVER와 Controller Tester에서 사용하는 컴파일 플래그가 다른 경우

커버리지를 측정할 때 전처리 파일을 기준으로 측정합니다. 컴파일 플래그가 다른 경우 전처리 파일이 달라질 수 있습니다. 이 경우 COVER와 Controller Tester의 컴파일 플래그를 동일하게 맞추고 테스트를 재수행한 후 커버리지 내보내기/가져오기를 합니다.

```
void testFunction(){
    int a;
    int b;
//Controller Tester에서는 CT_FLAG라는 매크로를 추가하고 커버에서는 추가하지 않은 경우나 혹은 그 반대의 경우
#ifdef CT_FLAG
     callFunction();
#endif
}
```

동일한 함수의 bodyhash가 다르게 계산된 경우

COVER와 Controller Tester에서 같은 함수의 bodyhash의 값이 다르게 계산되어 커버리지 가져오기가 실패하는 경우가 있습니다. 이 경우 ut.ini 파일에 `EXCLUDE_BODYHASH_CVR_IMPORT=true` 옵션을 추가하면 bodyhash의 값이 다르더라도 커버리지를 가져올 수 있습니다.

- ut.ini 경로 : `%프로젝트 경로%\.csdata\ut.ini`

1.26. 상수 주소를 사용할 때 signal error 가 발생하는 경우

```
int * ptr;
int * pre_ptr = (int *) (0x60000000U);
ptr = (int *)pre_ptr ; // 포인터 캐스팅 사용
```

위와 같이 상수 주소 값을 포인터에 할당하는 코드가 있을 경우, 테스트를 실행했을 때 signal error가 발생할 수 있습니다.
이 때 프로젝트에 사용한 툴체인의 .ini 파일에 use_memory_map=1 옵션이 설정되어 있다면, 프로젝트에서 사용할 가상 메모리 주소를 설정해줌으로써 signal error를 해결할 수 있습니다.
프로젝트에서 가상 메모리 주소를 설정하기 위해서는 아래의 방법을 사용할 수 있습니다.

- [환경설정] > [테스트] > [가상 메모리 주소] 에서 사용하고자 하는 가상 메모리 주소를 추가한 후, [프로젝트] > [특성] > [테스트] > [가상 메모리 주소] 에서 추가한 가상 메모리 주소를 설정해줍니다.

위 방법으로 가상 메모리 주소를 설정한 후, 프로젝트를 재분석하고 테스트를 실행하면 signal error 가 발생하지 않는 것을 확인할 수 있습니다.

> 툴체인의 .ini 파일은 [환경설정] > [툴체인] > [설정 디렉터리 열기] 를 실행하여 확인할 수 있습니다. 옵션을 작성할 때, [CONVERTER_OPTION] 하위에 작성해야 합니다.

> 일반 포인터 변수에 대해서 포인터 캐스팅을 사용하는 경우, use_memory_map와 convert_pointer_cast_variable 옵션을 전부 1로 켜야 합니다.

1.27. 부동 소수점 타입의 변수를 비교 연산할 경우 결과가 다르게 나오는 문제

부동 소수점 타입의 변수를 비교 연산할 경우 잘못된 결과가 나올 수 있습니다.

```
int main()
{
    float x = 0.03f;
    float y = 0.1f;
    y += 0.06;
    y += 0.01;
    if (x == y)
        printf("x == y\n");
    else
        printf("x != y\n");
}
```

위 예제는 부동 소수점을 비교할 때 발생하는 알려진 이슈로 x와 y를 비교 연산한 결과가 false가 나오는 경우가 대표적인 예 입니다.
해당 이슈는 도구의 버그가 아니라 도구가 일부 부동 소수점 비교 연산을 직접 하는 경우 발생할 수 있는 버그를 발견한 이슈로 보셔야 할 것 같습니다.

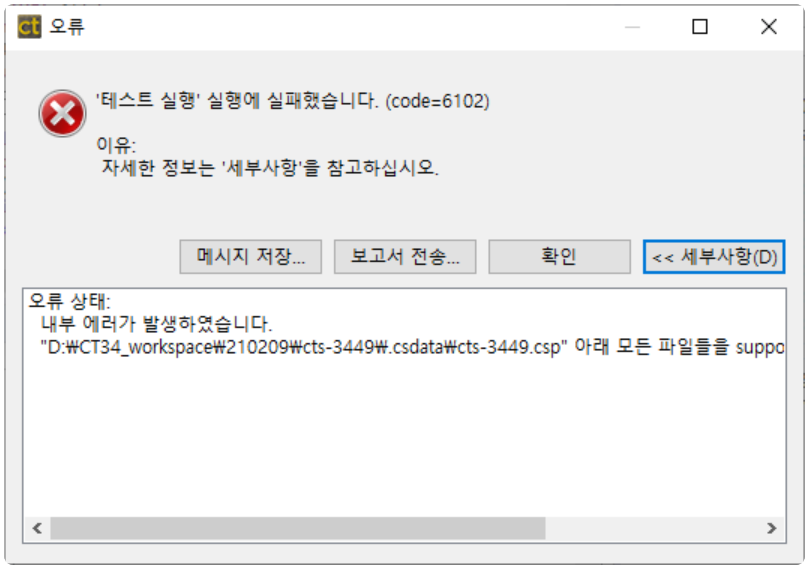
```
int compareValues(float val1, float val2){
        float diff = val1 - val2;
        float tolerance = 0.0001f;
        int result = 0;
        if (diff < tolerance){
                result = 0;
        }else {
                result = 1;
        }
         return result;
}
int main()
{
    float x = 0.03f;
    float y = 0.1f;
    y += 0.06;
    y += 0.01;
    if (compareValues(x,y) == 0)
        printf("x == y\n");
    else
        printf("x != y\n");
}
```

이러한 경우 위 예제와 같이 간단한 비교 함수를 작성하여 오차 허용 범위 내에서 값을 비교하여 해결할 수 있습니다.

자세한 내용은 아래 링크를 참조 부탁 드립니다.
https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/
http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm
http://devmachine.blog.me/220119534107

1.28. 테스트 실행 시 빌드 성공 후 테스트 수행이 실패하는 경우

테스트 실행 시 빌드는 성공했지만 테스트 실행에 실패했을 때, 다음과 같은 창이 나오고 [오류 뷰]에 오류 메세지가 나오지 않는 경우에는 engine.log를 확인해볼 수 있습니다.



- engine.log 경로 : `프로젝트_경로\.csdata\log\engine.log`

engine.log 파일에 아래와 같은 오류 메세지가 출력된 경우에 다음과 같은 방법을 통해 문제를 해결할 수 있습니다.

```
[CEM] CFGFunction::load:not found function(qualifiedName, idFunctionInfo, func
tionKey)
```

1. `프로젝트_경로\.csdata\pa.ini` 파일에서 TYPE_MAX_BUFFER_SIZE 값을 큰 값(ex. 5000)으로 변경한다.
2. TYPE_MAX_BUFFER_SIZE를 변경한 후 프로젝트를 재분석 후 테스트를 수행한다.

engine.log에 해당 에러가 없거나 TYPE_MAX_BUFFER_SIZE 값을 변경하여도 해결되지 않는 경우 기술 지원 연락처로 연락 바랍니다.

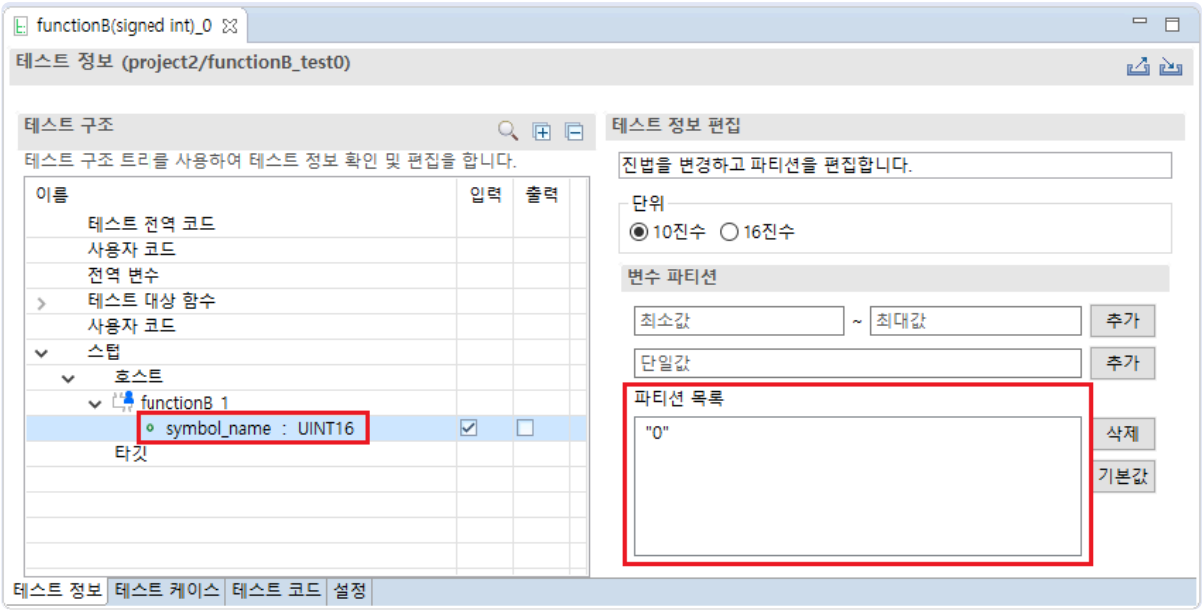
- 기술 지원 연락처 : help@suresofttech.com

1.29. 스텁 변수의 파티션 목록에 “0“만 존재할 경우

CS_XXX_INPUT 매크로에 첫 번째 인자는 타입, 두 번째 인자는 심볼 이름을 넣어 심볼을 사용할 수 있습니다.
만약, 첫 번째 인자에 기본 타입(int float, unsigned short 등)이 아닌 사용자 정의 타입을 다음과 같이 넣을 경우,

```
CS_UINT_INPUT(UINT16, "symbol_name");
```

`UINT16`은 기본 타입이 아니기 때문에, 해당 타입에 대한 파티션 정보를 처리하지 못하여 “0“만 존재하게 됩니다.



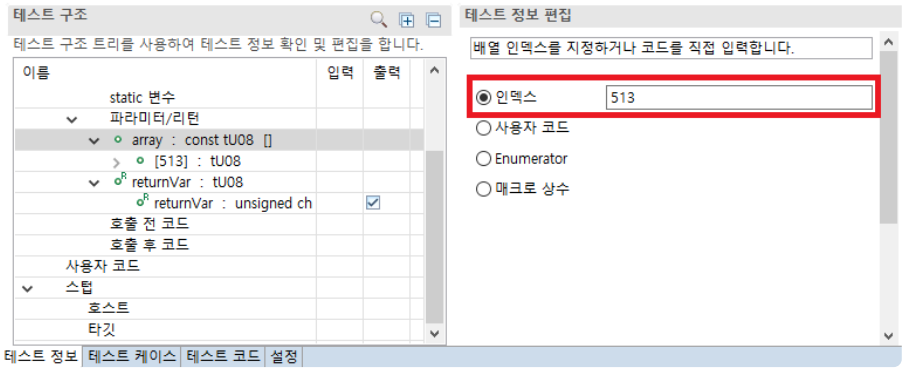
1.30. 스텁의 입력데이터 정보가 삭제되는 경우

테스트와 스텁을 연결한 후, CS 매크로에서 심볼명이나 타입을 수정하면 기존에 편집했던 스텁 관련 입력 데이터는 삭제됩니다.
테스트에 연결된 스텁의 심볼명이나 타입이 변경된 경우, 스텁 정의를 다시 읽어 입출력 심볼들을 처리하기 위해 기존에 읽었던 심볼을 삭제합니다.

1.31. 가변 배열을 인자로 받는 함수에서 signal error가 발생하는 경우

```
static tU08 getCheckSum(tU08 const array[])
{
    tU16  index;
    tU16 sum = 0U;
    tU08 checksum;
    for(index = 0; index < 511U; index++)
    {
        sum = sum + array[index];
    }
    checksum = (tU08)(sum & 0xffU);
    return checksum;
}
```

위처럼 가변 배열을 인자로 받는 함수의 테스트를 실행할 때, signal error가 발생하는 경우가 있습니다.
이 때는 함수 내에서 인자로 받은 배열에 인덱스로 접근을 하고 있는지 확인을 해봐야 합니다.
테스트 정보에서 지정한 배열 길이 보다 큰 인덱스로 접근하는 경우, [테스트 편집기] > [테스트 구조] > 가변 배열의 [테스트 정보 편집창] 에서 인덱스 값을 조정해야 합니다.



인덱스 값을 조정한 후 테스트를 재실행하면 signal error가 사라진 것을 확인할 수 있습니다.

> 함수 내에서 접근하는 인덱스 값보다 충분히 큰 값으로 인덱스 값을 조정해야 정상적인 실행 결과를 얻을 수 있습니다.

1.32. 구조체의 사이즈가 Controller Tester와 원본 프로그램에서 다르게 출력되는 경우

```
#pragma pack(1)
typedef struct TEST_STRUCTURE {
   /* Byte 0 */
     UINT8 uiSEQ;
   /* Byte 1 */
     BIT1 biCharDir :1;
     UINT32 uiReserved1 :7;
   /* Byte 2 */
     UINT8 uiPulseCount_Ch1;
   /* Byte 3 */
     UINT8 uiPulseCount_Ch2;
   /* Byte 4~7 */
     UINT32 uiPulseCount_Access;
   /* Byte 8 */
     UINT32 uiSwVer :6;
     UINT32 uiStatus :2;
   /* Byte 9~11 */
     UINT16 uiSizeIndex;
     UINT8 uiReserved2;
   /* Byte 12~15 */
     UINT32 uiC;
};
```

위와 같이 `#pragma pack` 지시문이 포함되어 있는 코드의 경우, Controller Tester에서의 구조체의 사이즈가 원본 프로그램과는 다르게 표시될 수 있습니다.
위 지시문은 구조체 메모리 정렬 옵션인데, 컴파일러 별로 pack 지원 여부 및 동작 방식이 다를 수 있기 때문입니다.
따라서, Controller Tester에서 변환 툴체인을 사용하는 경우 원본 프로그램 빌드할 때 사용한 컴파일러와 pack 동작 방식이 일치하는 visual studio 컴파일러 혹은 gcc 컴파일러를 선택하여 수행하면 위 문제를 해결할 수 있습니다.

1.33. IAR 환경에서 undefined referenced 에러가 발생하는 경우

IAR 환경에서 anonymous union 또는 anonymous struct가 사용될 경우 발생할 수 있습니다.
IAR 컴파일러에서는 anonymous union 또는 anonymous struct의 멤버 변수들을 전역변수처럼 접근해서 사용할 수 있습니다.
이로 인해 Controller Tester에서 분석 시 특정 변수들을 찾을 수 없는 에러가 발생합니다.
IAR의 특수한 문법에 대한 의미를 살리기 위해서는 anonymous union과 anonymous struct의 멤버 변수들을 전역 변수로 추가해야 합니다.

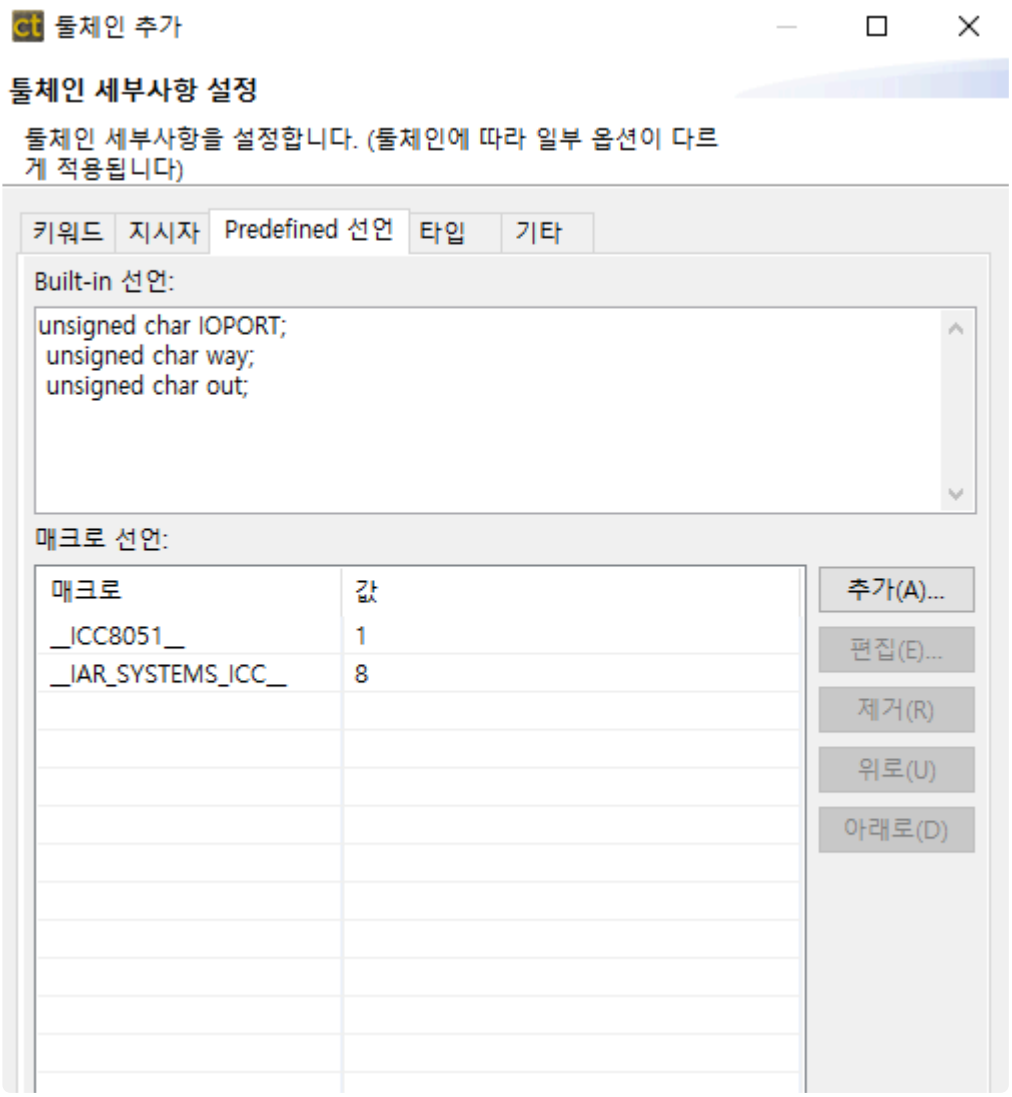
전역변수를 추가하는 방법은 아래와 같습니다.

- ‘환경설정’ > ‘툴체인’ > 해당 툴체인 선택 후 ‘편집’ > ‘Predefined 선언’ > ‘Built-in 선언’ 에 변수 선언 추가

IAR anonymous union 사용 예제

```
__no_init volatile
union
{
 unsigned char IOPORT;
 struct
 {
 unsigned char way: 1;
 unsigned char out: 1;
 };
} @ 0x1000;
/* The variables are used here. */
void Test(void)
{
 IOPORT = 0; // union의 멤버 변수를 전역 변수 처럼 사용
 way = 1; // struct의 멤버 변수를 전역 변수 처럼 사용
 out = 1; // struct의 멤버 변수를 전역 변수 처럼 사용
}
```

툴체인에 추가한 전역 변수


툴체인 추가
툴체인 세부사항 설정
툴체인 세부사항을 설정합니다. (툴체인에 따라 일부 옵션이 다르게 적용됩니다)
키워드
지시자
Predefined 선언
타입
기타
Built-in 선언:
unsigned char IOPORT;
unsigned char way;
unsigned char out;
매크로 선언:
매크로
값
__ICC8051__
1
__IAR_SYSTEMS_ICC__
8
추가(A)...
편집(E)...
제거(R)
위로(U)
아래로(D)

1.34. const 전역 변수가 존재하여 테스트 코드에서 에러가 발생하는 경우

우선 툴체인.ini 파일에 있는 아래의 옵션을 변환하여 전역 변수의 const를 제거할 수 있습니다.

```
global_variable_nonconstant = 1
```

하지만 테스트 실행 시 기본적으로 생성되는 테스트 코드가 컴파일이 불가능하여 컨버팅 단계에서 에러가 발생합니다.

테스트 코드를 컴파일이 가능한 형태로 수정하여 테스트를 실행시키면 정상적으로 빌드가 가능합니다.

```
void Rte_MemCpy(void *destination, void *source, unsigned long num);

typedef unsigned char CanMsgData[8];
typedef unsigned char uint8;
CanMsgData data;
const CanMsgData Rte_C_CanMsgData_0 = {
  0U, 0U, 0U, 0U, 0U, 0U, 0U, 0U
};

void Rte_MemCpy(void *destination, void *source, unsigned long num)
{
}
void main(void)
{
 Rte_MemCpy(data, Rte_C_CanMsgData_0, sizeof(CanMsgData));
}
```

위와 같은 코드에서는 기본적으로 생성되는 테스트 코드는 아래와 같습니다.

```
/*Input*/
Rte_C_CanMsgData_0[0] = CS_UINT_INPUT(unsigned char,"Rte_C_CanMsgData_0[0]");
data[0] = CS_UINT_INPUT(unsigned char,"data[0]");
```

해당 Rte_C_CanMsgData_0는 const 전역 변수로 선언이 되어있기 때문에, 에러가 발생합니다.
`error: expression must be a modifiable lvalue`

해당 전역 변수의 테스트 정보를 사용자 코드로 바꾸어 다음과 같이 컴파일 가능한 형태로 수정해줍니다.

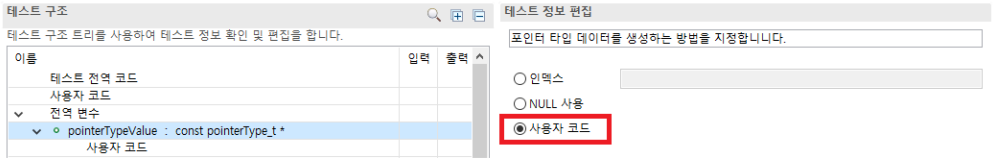
```
unsigned char * Rte_C_CanMsgData_Temp = (unsigned char *)Rte_C_CanMsgData_0;
// 주소를 가리키고 있으면 컴파일 시점에서는 const를 판별하지 못하여 컨버팅에서 컴파일이 정상적으로 되는 코드라고 판단하여 정상 동작함
// 런타임에서 const를 체크할 때는 global_variable_nonconstant 옵션으로 인해 이미 cons
```

```
t가 제거된 상태로 런타임 에러에 걸리지 않음
Rte_C_CanMsgData_Temp[0] = CS_INT_INPUT(unsigned char, "Rte_C_CanMsgData_Tem
p[0]");
Rte_C_CanMsgData_Temp[1] = CS_INT_INPUT(unsigned char, "Rte_C_CanMsgData_Tem
p[1]");
Rte_C_CanMsgData_Temp[2] = CS_INT_INPUT(unsigned char, "Rte_C_CanMsgData_Tem
p[2]");
Rte_C_CanMsgData_Temp[3] = CS_INT_INPUT(unsigned char, "Rte_C_CanMsgData_Tem
p[3]");
Rte_C_CanMsgData_Temp[4] = CS_INT_INPUT(unsigned char, "Rte_C_CanMsgData_Tem
p[4]");
```

1.35. static const 포인터 변수에 접근할 때 signal error가 발생하는 경우

```
static const pointerType_t* pointerTypeValue;

extern pointerType_t* externPointerTypeFunction(void);

static void getMessageFunction(tU08 timeTick){
        MessageInfo msg;
        tS08 tick= (tS08)timeTick;

        while(tick >= 0){
                tick = tick-4;
                if(tick==0){
                        msg = (*pointerTypeValue->message)(); // signal error
발생
                        break;
                }
        }
}
```

static const 포인터 변수에 접근할 때, static const 포인터 변수가 초기화되어 있지 않으면 signal error가 발생할 수 있습니다.
이 때는 Controller Tester에서 해당 함수의 테스트 편집창을 열고 테스트 구조 트리에서 초기화되지 않은 static const 포인터 변수를 찾아, 데이터 생성 방법을 [사용자 코드]로 지정합니다.



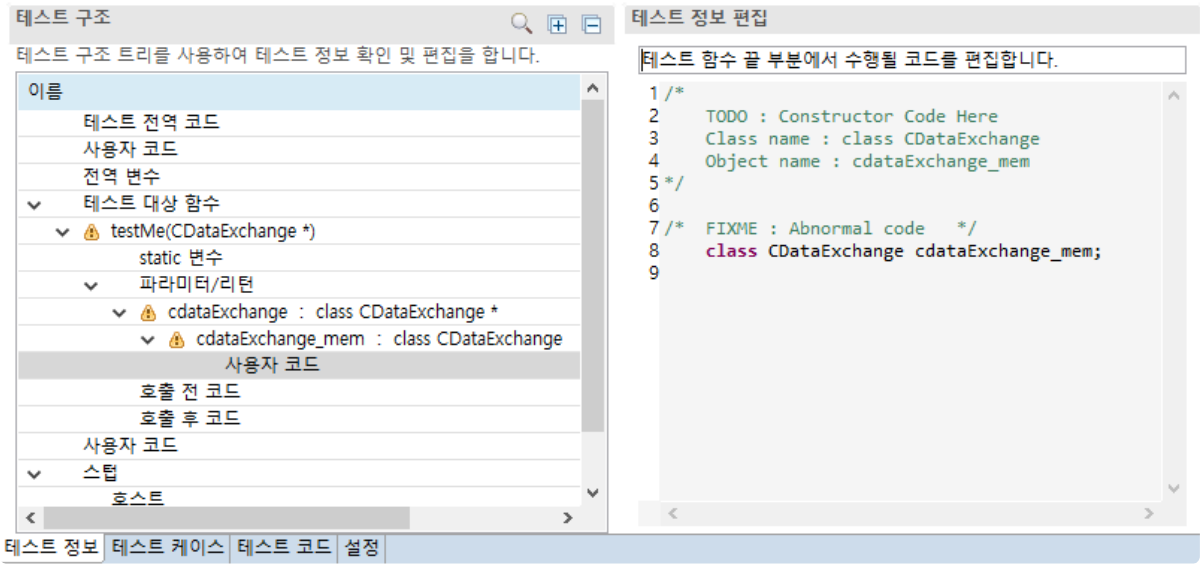
사용자 코드 작성창에서 동일한 타입의 새로운 객체를 초기화한 후, 코드에 선언된 static const 변수가 가리킬 수 있도록 합니다.
아래는 위 코드에 대해 작성한 사용자 코드의 예시입니다.

```
pointerType_t temp1 = {0x0,0x0,externPointerTypeFunction, };
pointerType_t* temp2 = &temp1;
pointerTypeValue = temp2;
```

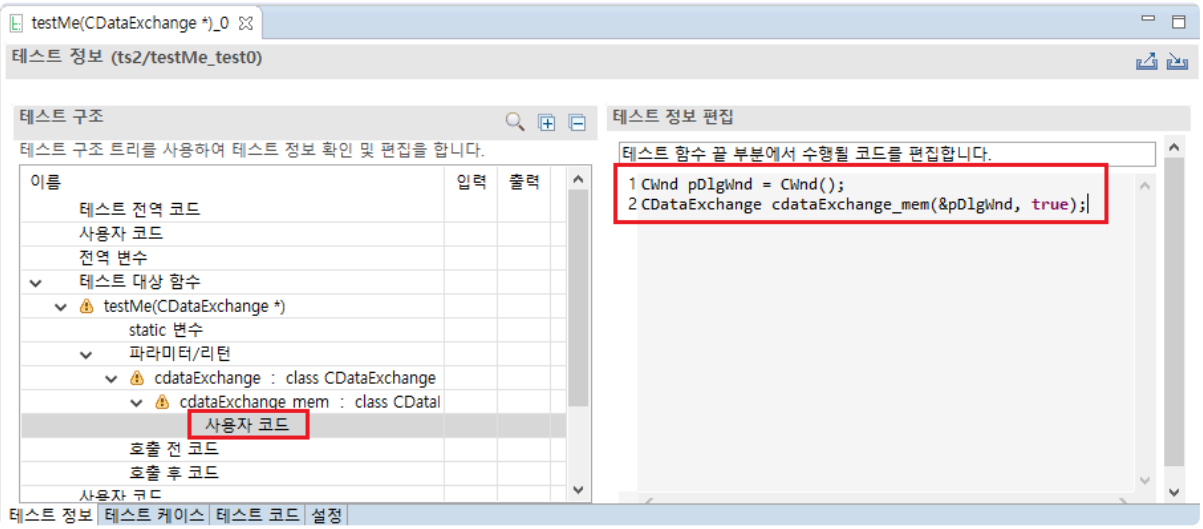
테스트 편집창을 저장하고 테스트를 재실행하면 signal error를 해결할 수 있습니다.

1.36. “C2512: 사용할 수 있는 적절한 기본 생성자가 없습니다.” 에러가 발생하는 경우

```
#include <afxwin.h>
void testMe(CDataExchange *cdataExchange)
{
        return;
}
```



Controller Tester에서 분석 대상이 아닌 클래스의 생성자가 사용될 경우 자동으로 기본 생성자가 선택됩니다.
위 예제를 수정 없이 실행하면 기본 생성자가 선택되어 “error C2512: 사용할 수 있는 적절한 기본 생성자가 없습니다.” 메시지를 출력하며 컴파일 에러가 발생합니다.



이런 경우 위 예제와 같이 해당 클래스의 적당한 생성자를 직접 사용자 코드에 입력해서 해결 할 수 있습니다.

! CT3.4 버전 이후 부터는 일부 클래스에 대해서 기본생성자를 생성해주기 때문에 해당 문제가 발생하지 않을 수 있습니다.

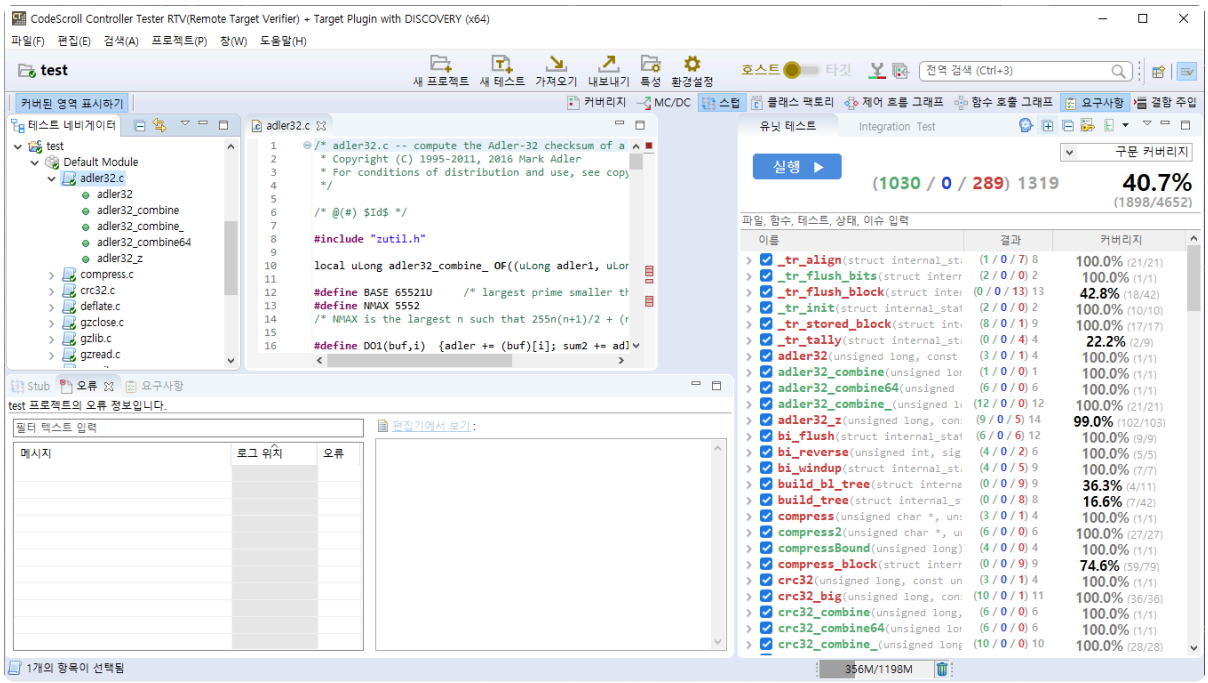
1.37. 테스트 케이스 별로 다르게 동작하는 스텁 코드 디자인하는 방법

테스트에 스텁을 연결한 후 테스트를 실행할 때, 테스트 케이스 별로 실행 결과를 다르게 할 수 있습니다. 다음은 스텁에서 반환하는 결과를 테스트 케이스 별로 다르게 작성한 스텁 코드의 예시입니다.

```
int result = 0;
if(CS_TESTCASENO() == 1) { // 테스트 케이스 별로 다른 동작을 할 수 있도록 수정
        result = some_function1();
} else if(CS_TESTCASENO() == 2) {
        result = some_function2();
}
return result;
```

1.38. Controller Tester의 퍼스펙티브가 깨진 경우

Controller Tester에서 뷰를 옮겨가며 테스트를 진행하다보면 퍼스펙티브가 아래 그림처럼 이상해지는 경우가 있습니다.

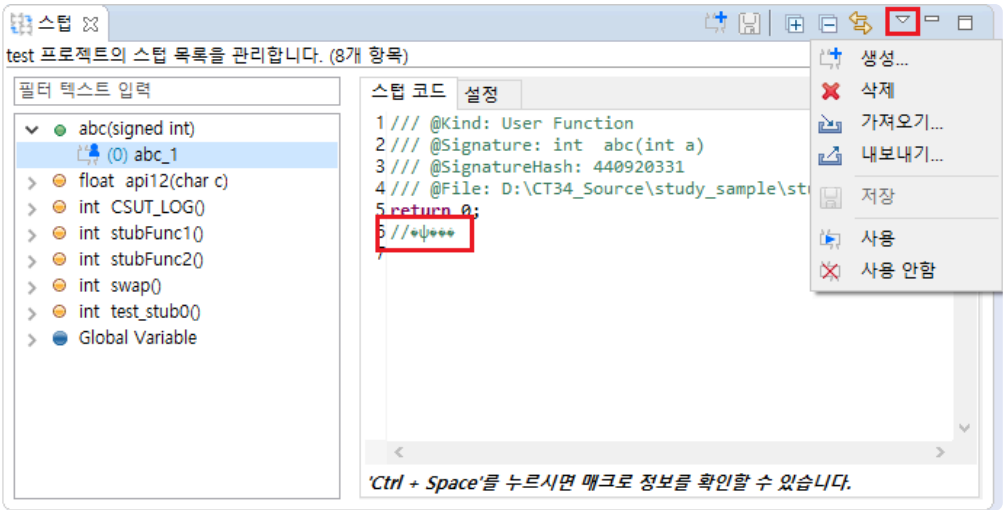


[창] – [퍼스펙티브 재설정]을 통해 퍼스펙티브를 되돌릴 수 있습니다. [퍼스펙티브 재설정]을 하여도 해결되지 않으면 아래 방법을 사용하여 퍼스펙티브를 초기화하세요.

1. Controller Tester를 종료합니다.
2. 워크스페이스_경로\.metadata\.plugins\org.eclipse.e4.workbench\workbench.xmi 파일을 삭제합니다.
3. Controller Tester를 다시 실행합니다.

1.39. Controller Tester 3.4 전에 내보낸 스텁 코드를 3.4 이후에 가져올 때 한글 주석이 깨지는 문제

Controller Tester 3.4 전에 스텁 뷰에서 내보낸 스텁 코드를 Controller Tester 3.4 이후에 가져오기할 때, 한글이 깨지는 현상이 있습니다.



Controller Tester 3.3까지는 스텁 코드의 인코딩이 ANSI이고 Controller Tester 3.4 이후부터는 UTF-8이기 때문입니다. Controller Tester 3.4 이전에 내보낸 스텁 코드를 가져올 경우 외부 편집기를 사용하여 인코딩을 UTF-8로 변경한 후 가져와야 합니다.

1.40. 레지스터 변수가 매크로로 선언되어 있어서 테스트 케이스에서 값을 입력할 수 없는 경우

```
#define IN_IG_ON (int8_t)(PORTDbits.RD0)
#define IN_INTER_LOCK_SNSR (int8_t)(PORTDbits.RD2)
#define IN_DIFF_LOCK_SNSR_2ND (int8_t)(PORTDbits.RD3)
#define IN_DIFF_LOCK_SNSR_3RD (int8_t)(PORTDbits.RD12)

if((IN_IG_ON == OFF) && (getHazardSwitch() == OFF) && (getBrakeSwitch() == OF
F))
{

}
```

위와 같은 코드에서 getHazardSwitch()나 getBrakeSwitch() 함수의 리턴값은 스텁을 이용하여 return 값을 변경할 수 있지만 IN_IG_ON의 값은 매크로로 선언되어 있어서 테스트 케이스에서 값을 제어할 수 없습니다.

이와 같은 경우에는 레지스터 변수 사용을 함수로 묶어 스텁으로 처리할 수 있도록 고객사에서 코드를 수정해야 합니다.

1.41. xls 형식의 보고서를 html 형식으로 변환하는 방법

스크립트를 실행하여 CT에서 내보낸 xls 형식의 보고서를 html 형식으로 변환할 수 있습니다.

1. xls_to_html.zip 파일 압축 해제 후 xls_to_html.vbs 파일을 편집기로 열기
2. ctTestResultXLS에 변환할 보고서의 디렉터리 경로 입력
3. ctTestResultHTML에 html형식의 보고서가 생성될 디렉터리 경로 입력

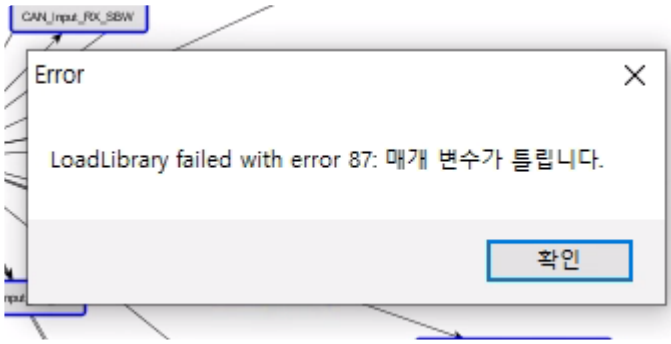
> ! 디렉터리 경로는 미리 생성되어 있어야 하며, 경로 끝에는 \을 포함하여 작성해야 합니다.

4. cmd에서 xls_to_html.bat을 실행하거나 Start xls_to_html.vbs를 입력하여 스크립트 실행

xls_to_html.zip

1.42. “LoadLibrary failed with error 87: 매개 변수가 틀립니다.” 에러가 발생하는 경우

Controller Tester 사용 중 “LoadLibrary failed with error 87: 매개 변수가 틀립니다.“와 같은 경고창이 뜨는 경우가 있습니다.



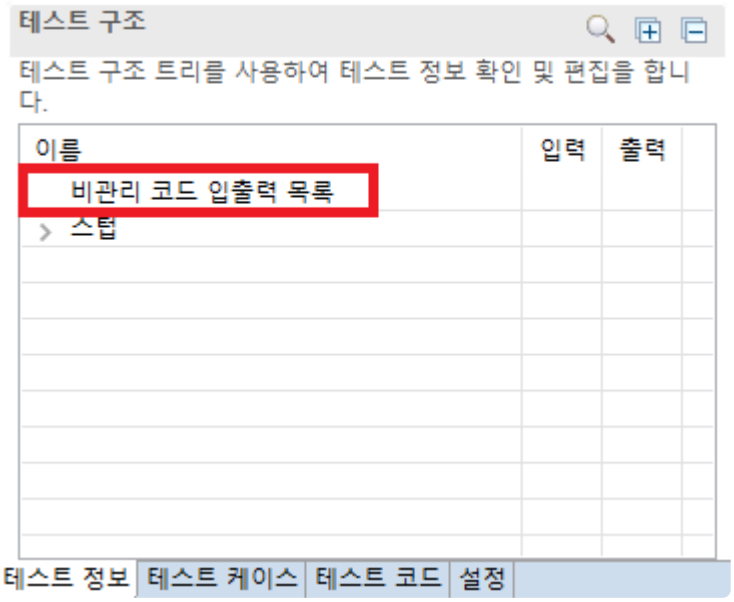
이와 같은 오류는 해당 PC에 설치된 그래픽 카드 드라이버에 문제가 있거나, 원격 접속을 사용할 때 호스트 PC와 원격 PC의 그래픽 카드에 충돌이 있을 경우 발생합니다.
문제를 해결하기 위해서는 다음과 같은 방법들을 시도할 수 있습니다.

- 그래픽 카드 드라이버를 최신으로 업데이트합니다.
- 만일 현재 드라이버가 최신이라면, 이전 버전으로 되돌립니다.
- 두 개 이상의 그래픽 카드가 켜져 있을 경우, 문제가 되는 그래픽 카드를 비활성화합니다.
- 원격 접속을 사용하는 경우, 호스트 PC의 그래픽 카드를 비활성화하고 원격 접속을 시도합니다.
- Controller Tester을 완전히 삭제하고 다시 설치합니다.

1.43. 테스트 편집기에서 비관리 코드 입출력 목록 표현 안되는 문제 (CT 3.4 이전 버전)

Controller Tester 3.3 이하 버전에서는 다음과 같은 경우에 테스트 편집기의 테스트 정보 창에 변수 목록이 표시되지 않을 수 있습니다.

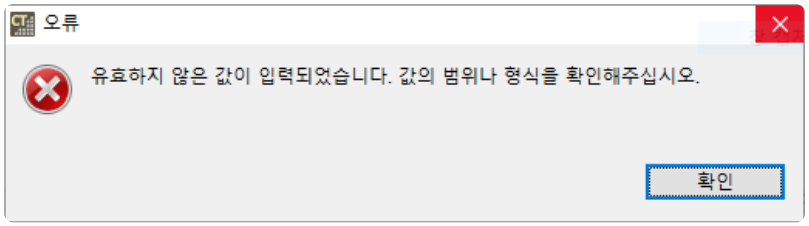
- 비관리 테스트 코드 또는 스텁 코드에 입출력 매크로를 사용하여 많은 변수를 추가했을 때
- 위와 같은 형태의 테스트를 [테스트 가져오기] 혹은 [테스트 코드 파일로부터 테스트 가져오기] 했을 때
- 위와 같은 형태의 테스트 코드 또는 스텁 코드를 일부 수정 후 저장했을 때



Controller Tester 3.3 이하 버전에서는 테스트 정보 창에서 표현할 수 있는 변수 개수가 최대 4096개로 제한이 있기 때문입니다.
제한하는 것보다 많은 수의 변수를 사용해야 한다면, Controller Tester를 3.4 이상 버전으로 올려 사용하거나 지원팀(help@suresofttech.com)에 문의주시기 바랍니다.

1.44. 유효하지 않은 값을 테스트 데이터에 입력하는 방법

Controller Tester에서 유효하지 않은 값을 테스트 케이스의 입력값 또는 기대값에 입력하는 경우 오류가 발생합니다.



- Ex 1. `unsigned int` 타입의 변수에 음수 값을 입력
- Ex 2. `char` 타입의 변수에 -128 미만 또는 127 초과의 값을 입력하는 경우

유효하지 않은 값을 테스트 데이터에 입력하는 방법

예시에서 사용한 변수는 `b`이고 `char` 타입입니다.

1. 테스트 뷰에서 유효하지 않은 값을 입력할 함수를 우클릭하여 테스트 데이터를 내보냅니다.
2. 내보낸 테스트 데이터 파일(.csv)을 엽니다.
3. 파일 형식에 맞춰 값을 입력하고 저장합니다.

> Controller Tester 3.5부터는 테스트 데이터 가져오기 시, 테스트 케이스를 추가하는 정책으로 변경되었습니다. 특정 테스트 케이스의 데이터만 가져오려는 경우, 기존 테스트 케이스를 삭제한 후 가져오시기 바랍니다.

- Controller Tester 3.5 이후

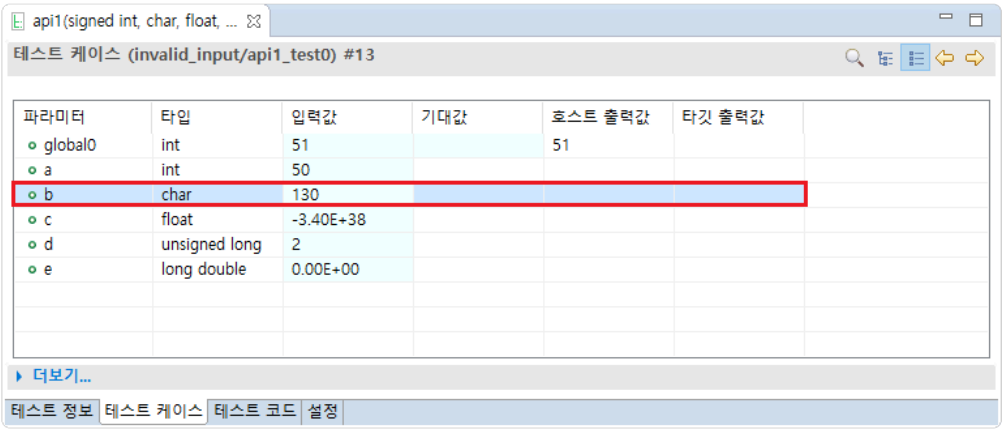
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	CodeScroll Unit Tester(Controller Tester) Test Data													
2	test name:api1_test0													
3		<input>	a	b	c	d	e	global0	<expect>	global0	<output>	global0	<toutput>	global0
4	1		50	130	-3.40E+38	2	0.00E+00	51				51		

- Controller Tester 3.4 이전

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	CodeScroll Unit Tester(Controller Tester) Test Data													
2	test name:api1_test0													
3		<input>	a	b	c	d	e	global0	<expect>	global0	<output>	global0	<toutput>	global0
4	1		50	0	-3.40E+38	2	0.00E+00	51				51		
5	2		1	11	-1.18E-38	9	-1.79769E-	0				0		
6	3		10	9	-3.40E+38	49	1.79769E+	11				11		
7	4		49	-1	-1.18E-38	11	0.00E+00	2				2		
8	5		51	-128	-3.40E+38	4.29E+09	1.79769E+	49				49		
9	6		9	127	-1.18E-38	50	0.00E+00	-2.1E+09				-2.1E+09		
10	7		-2.1E+09	50	1.18E-38	10	-1.79769E-	-1				-1		
11	8		11	10	3.40E+38	1	0.00E+00	1				1		
12	9		2	49	0	0	0	9				9		
13	10		-1	2	0	1	-1.79769E-	2.15E+09				2.15E+09		
14	11		0	51	3.40E+38	0	0.00E+00	50				50		
15	12		2.15E+09	1	1.18E-38	51	0	10				10		
16	13		50	130	-3.40E+38	2	0.00E+00	51				51		

4. 테스트 뷰에서 해당 함수를 우클릭하여 저장한 테스트 데이터를 가져옵니다.

5. 테스트 편집기의 테스트 케이스 탭에서 가져온 테스트 데이터를 확인할 수 있습니다.

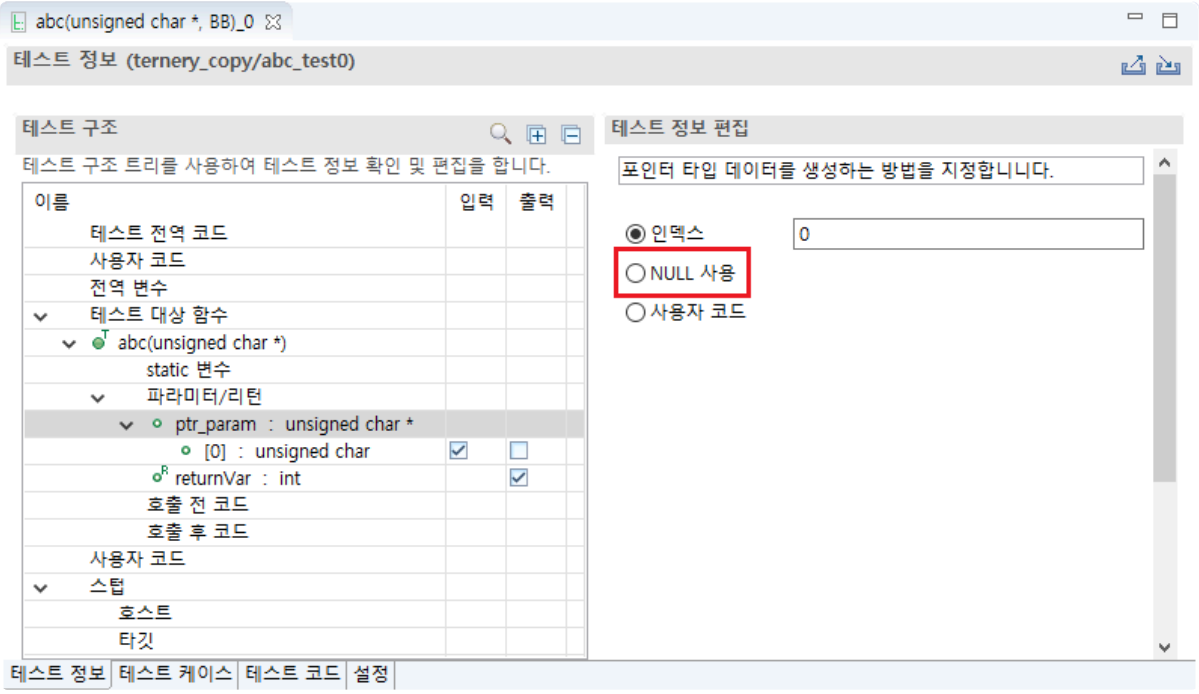


파라미터	타입	입력값	기대값	호스트 출력값	타깃 출력값
global0	int	51		51	
a	int	50			
b	char	130			
c	float	-3.40E+38			
d	unsigned long	2			
e	long double	0.00E+00			

1.45. 포인터형 파라미터에 NULL 값을 입력하는 방법

테스트에서 포인터형 파라미터에 NULL 값을 입력하는 방법

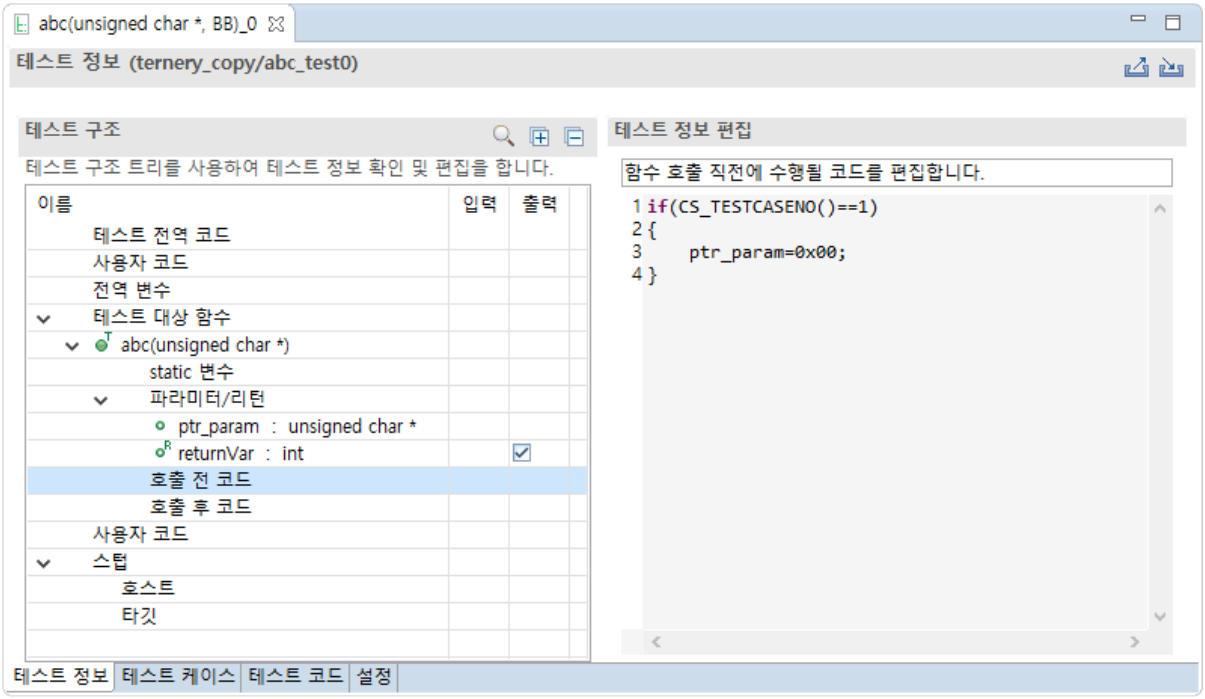
[테스트 편집기] – [테스트 정보] 탭 – [테스트 구조] 트리에서 해당 파라미터를 선택한 후 테스트 정보 편집에서 [NULL 사용]을 선택하고 저장합니다.



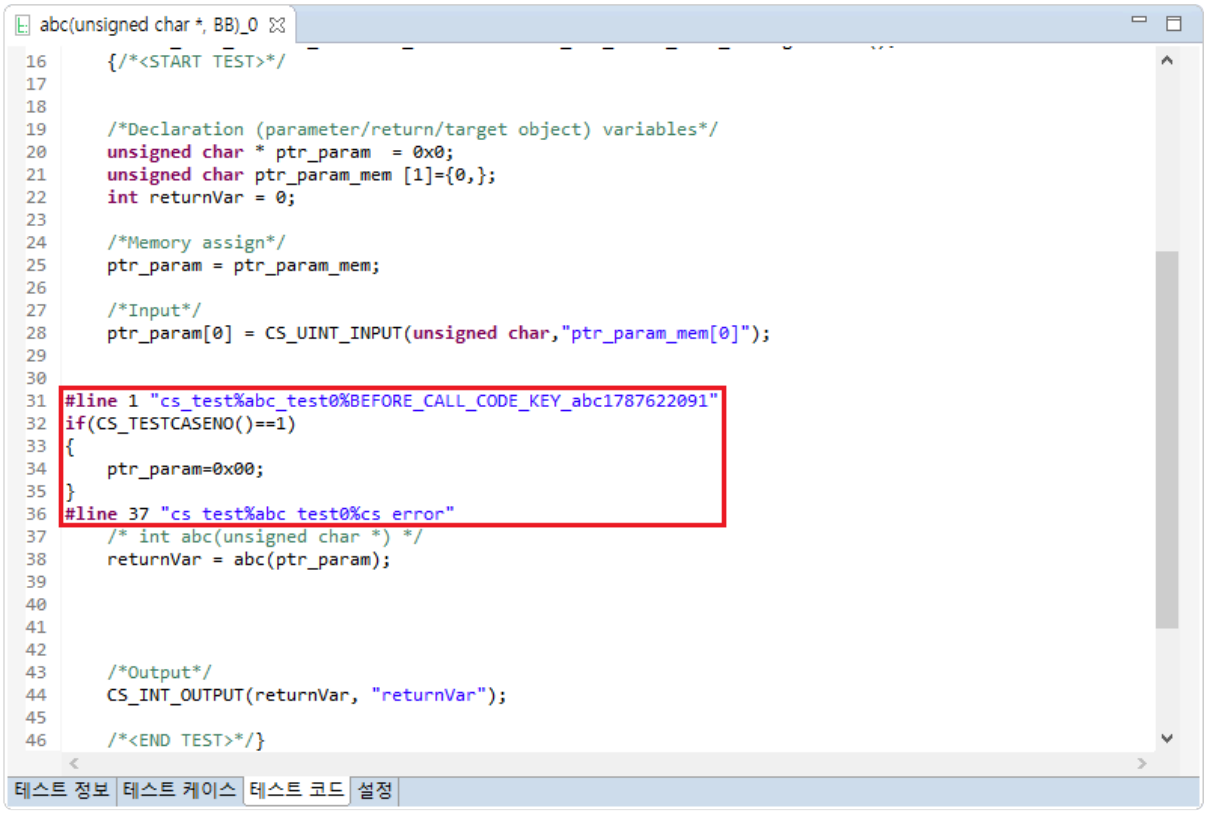
특정 테스트 케이스에서 포인터형 파라미터에 NULL 값을 입력하는 방법

테스트 케이스 번호를 반환하는 테스트 매크로(`CS_TESTCASENO()`)를 이용하여 특정 테스트 케이스의 파라미터에 NULL 값을 입력할 수 있습니다.

1. [테스트 편집기] – [테스트 정보] 탭 – [테스트 구조] 트리에서 다음 중 하나를 선택합니다.
    - [호출 전 코드]
    - 해당 파라미터를 선택한 후, 테스트 정보 편집에서 [사용자 코드]
2. 특정 테스트 케이스에서 NULL 포인터를 인자로 넘길 수 있도록 코드를 입력합니다.
    - 예시 코드는 테스트 케이스가 1번일 때, NULL 포인터를 인자로 넘겨 테스트하는 코드입니다.

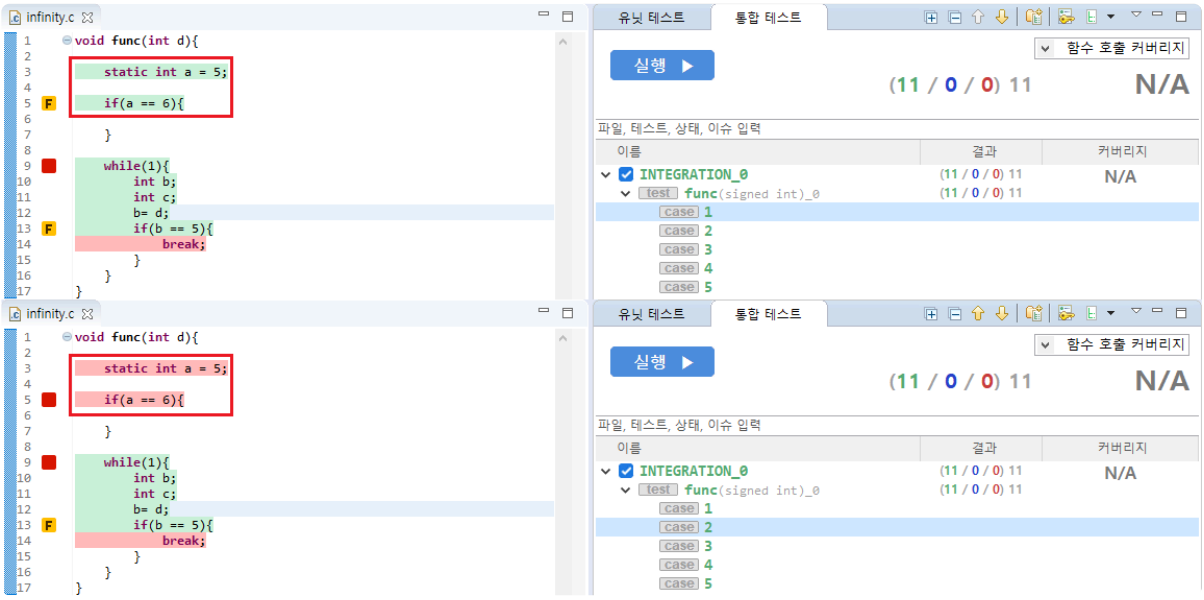


3. 입력한 코드는 [테스트 코드] 탭에서 확인할 수 있습니다. 테스트 코드를 확인하여 적절히 코드를 입력하면 됩니다.



1.46. 변환 툴체인을 사용할 때, 무한 루프를 제거한 통합 테스트에서 커버리지가 비정상적으로 표시되는 문제

변환 툴체인을 사용하여 호스트 시험을 진행하는 경우, 툴체인.ini 파일에서 remove_infinity_loop = 1로 설정하여 무한 루프를 제거하고 통합 테스트를 진행하면 첫번째 테스트 케이스의 커버리지는 정상적으로 표시되지만 두번째 테스트 케이스부터 커버리지가 비정상적으로 표시됩니다.

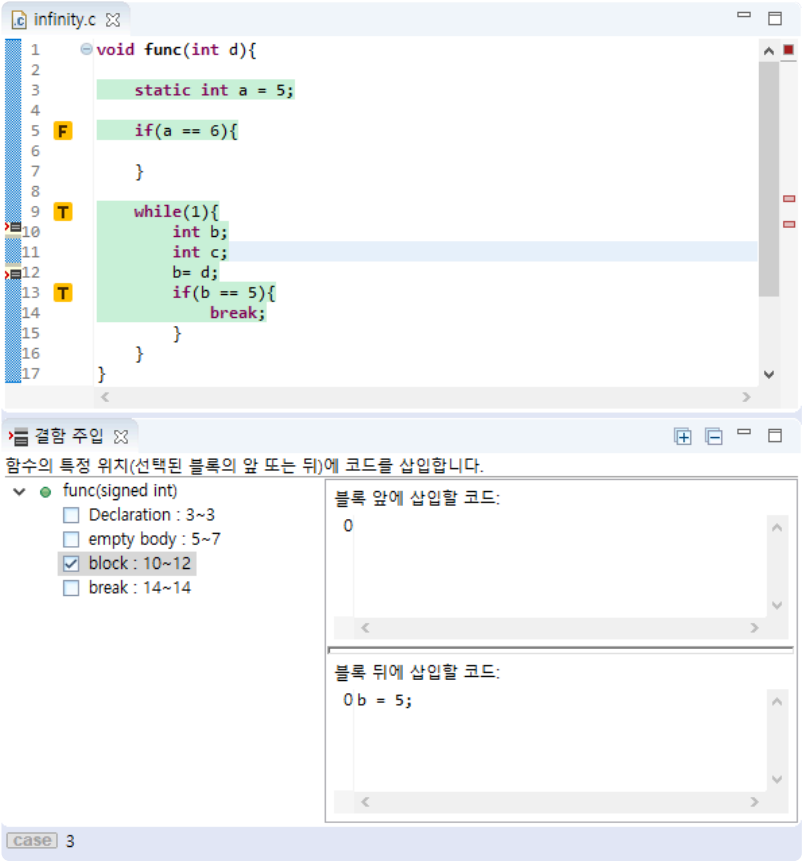


통합 테스트에서 while(1) 형식의 무한 루프문을 테스트 케이스 수만큼 반복할 수 있도록 Controller Tester가 제공하는 기능입니다.

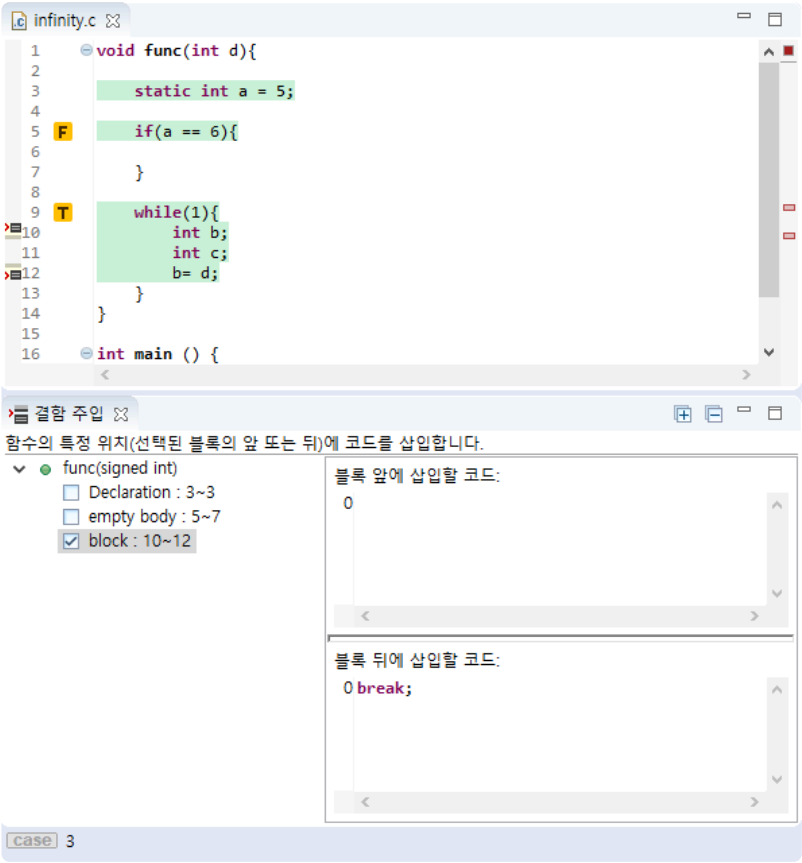
사용자가 통합 테스트에서 무한 루프를 제거하고자 할 때는 결함 주입 기능을 이용하여야 합니다.

1. 툴체인.ini 파일에서 remove_infinity_loop = 0으로 설정합니다.

2. 반복문 내부에 결함을 주입하여 반복문을 벗어납니다.
   - 반복문 내부에 무한 루프를 벗어날 수 있는 조건문이 있는 경우
     - 해당 조건문을 통해 무한 루프를 벗어나도록 적절히 결함을 주입합니다.



- 반복문 내부에 무한 루프를 벗어날 수 있는 조건문이 없는 경우
  - 반복문 끝에 `break;`를 넣어 무한 루프를 벗어납니다.



1.47. 테스트 내보내기 시 파일명이 잘리는 경우

테스트 내보내기 시 파일 이름이 테스트 코드 파일 이름 길이 제한보다 길 경우 파일명이 잘려나올 수 있습니다.
문제를 해결하기 위해서는 다음과 같은 방법들을 시도할 수 있습니다.

1. 도구를 종료합니다.
2. `workspace\project\.csdata\ut.ini` 파일에서 TEST_FUNCTION_NAME_LENGTH 값을 100 정도로 설정한 다음
글자가 잘리는 테스트를 새로 생성하여 내보내기를 수행합니다.
3. 도구를 실행합니다.

> ! 내보내기 시 내보내는 위치 + 파일 명이 윈도우에서 허용하는 최대 경로 길이 보다 짧게 설정해야합니다.

1.48. 변환툴체인을 사용하는 호스트 시험에서 'invalid use of void expression' 에러가 발생하는 경우

'invalid use of void expression' 오류는 void 함수에 대한 호출을 변수에 할당하려고 할 때 발생합니다. void 함수가 다른 데이터 유형으로 피연산될 때도 발생합니다.

원본 소스 파일에서 해당 에러가 발생하지 않더라도, Controller Tester 테스트 과정에서 해당 에러가 발생할 수 있습니다.

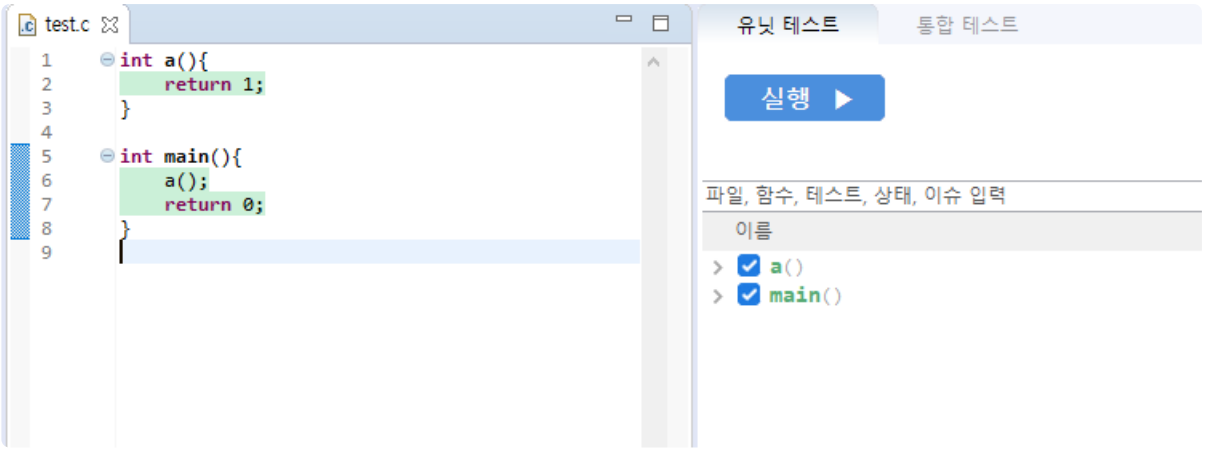
```
typedef unsigned int uint8_t;

typedef volatile uint8_t register8_t;

typedef struct PORT_struct
{
    register8_t DIR;
    ...
} PORT_t;

static static void PORTE_set_port_dir(void)
{
        uint8_t i;
        *((uint8_t *)&(*(PORT_t *) 0x0480) + 0x10 + i) |= 1 << 3;
}
```

위의 예시는 type punning을 사용하고 있고, 상수를 포인터로 캐스팅하여 메모리 주소에 직접 접근하고 있습니다.
이러한 경우에 가상 메모리 옵션을 사용 중일 경우, 변환 시 void *CS_UT_get_host_addr(unsigned int addr, unsigned int size); 함수로 해당 함수의 내용을 감싸게 됩니다.
CS_UT_get_host_addr 함수가 void 형이므로 정수로 피연산될 때 해당 에러가 발생합니다.

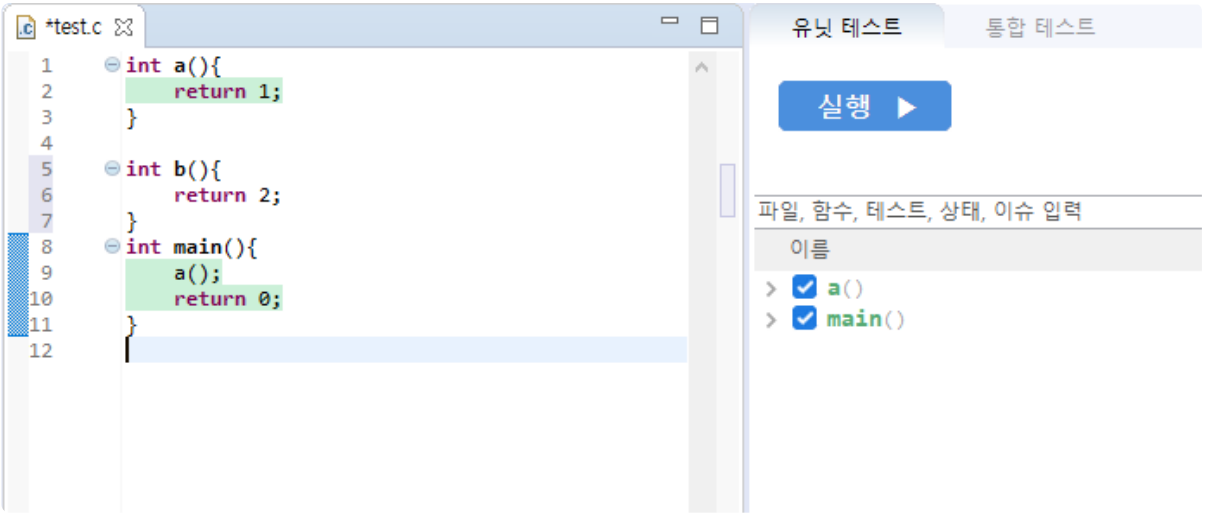
> %appdata%\Roaming\CodeScroll\1.1\parserConfig 의 툴체인.ini 파일에서 add_cast_on_memorymap_operation = 1 옵션을 추가하면 변환 시 CS_UT_get_host_addr 함수가 피연산자의 타입으로 캐스팅 됩니다.

1.49. 재활용 시나리오에서 신규 함수만 테스트가 생성이 필요할 때

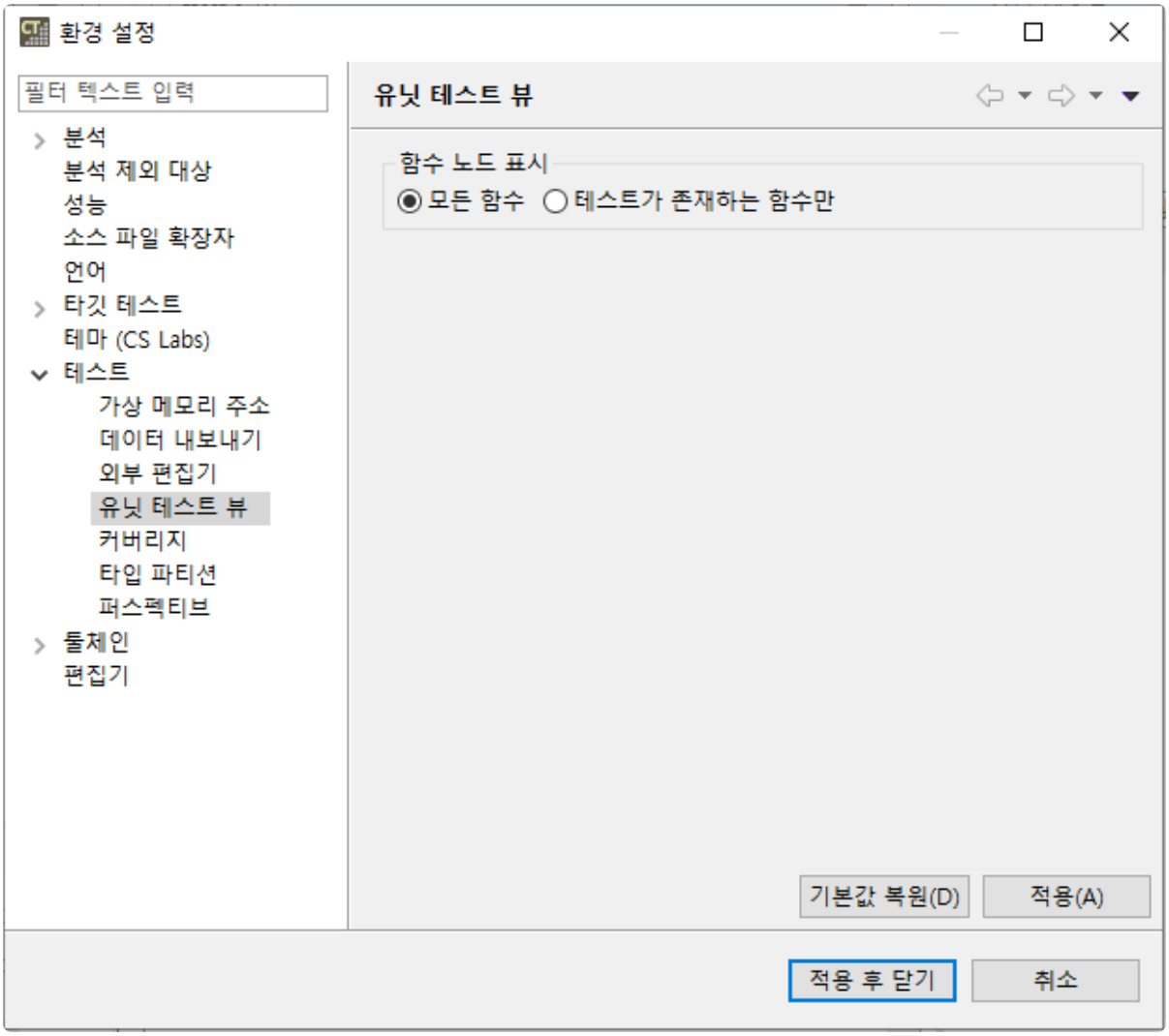
다음과 같은 방법으로 새 기능에 대한 테스트를 생성할 수 있습니다.



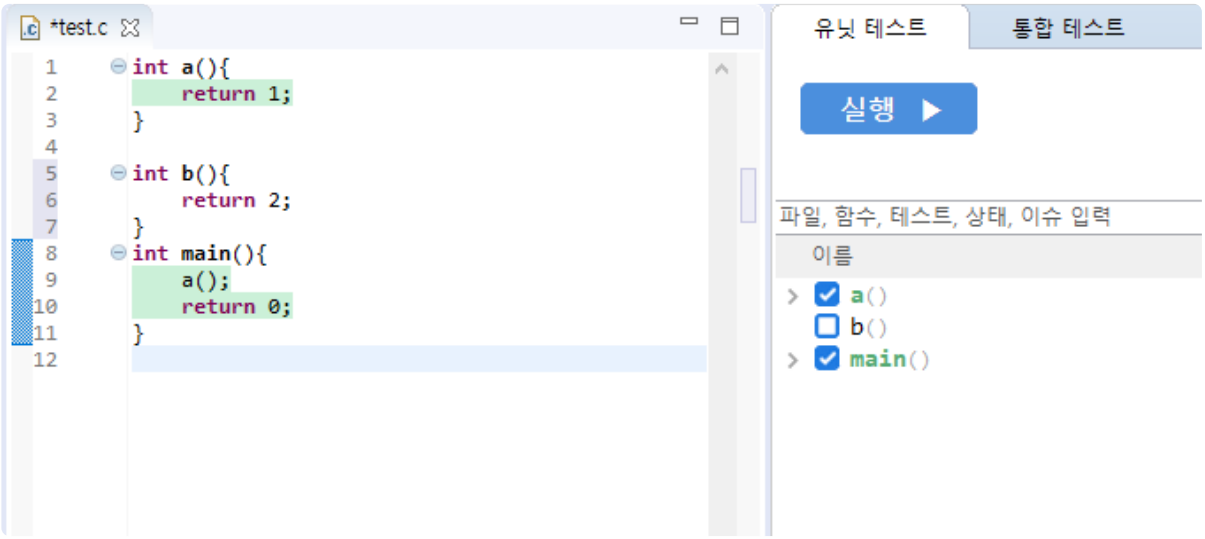
1. 이 예제에서는 a, main 함수에 대한 테스트가 완료되었습니다.



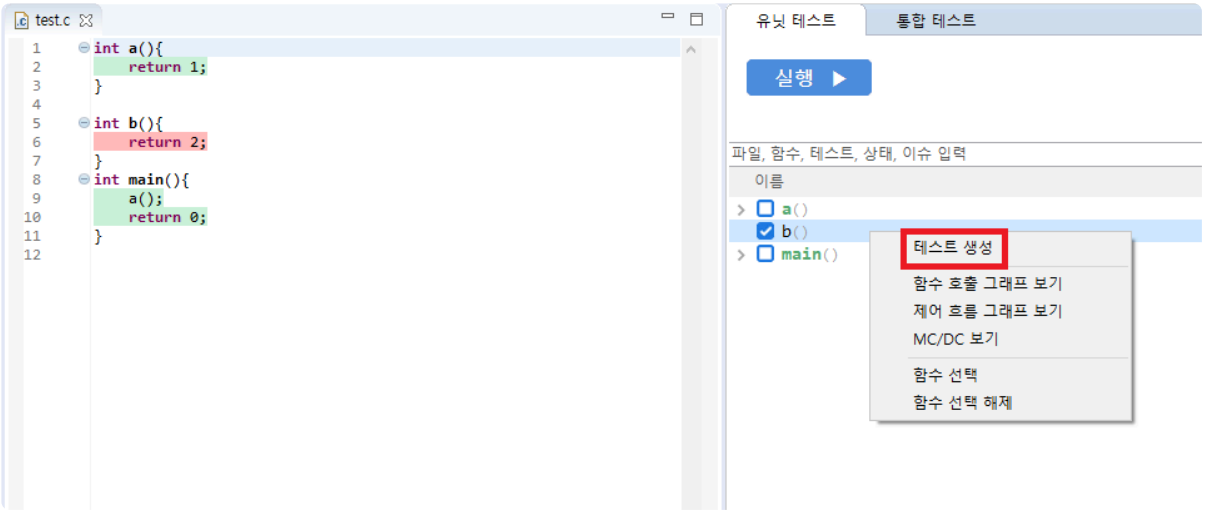
2. 새로운 함수 b가 소스에 추가되었습니다.



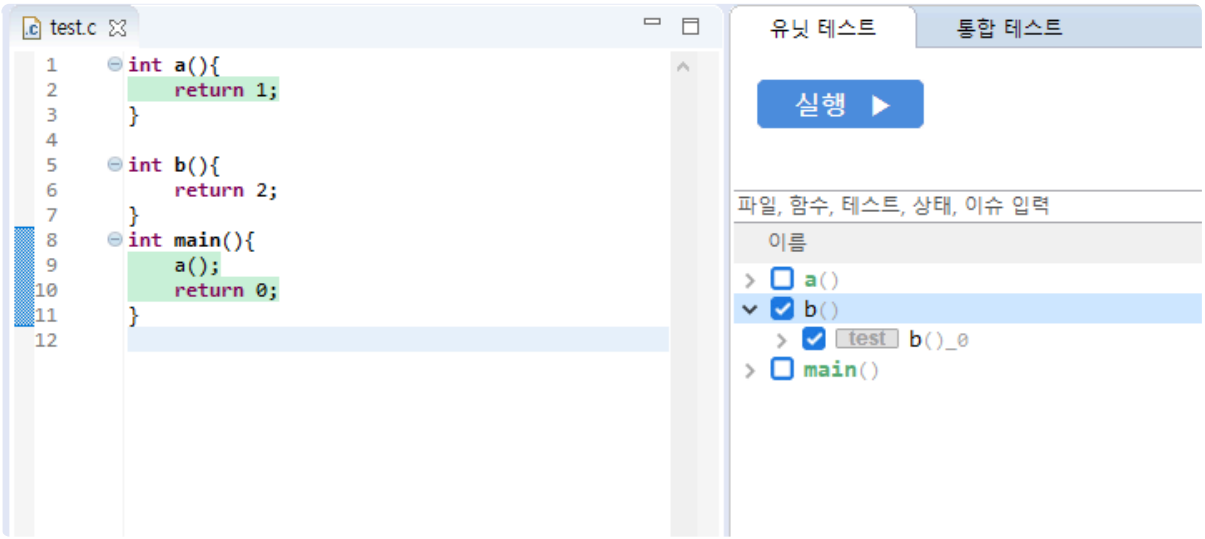
3. [환경설정] – [테스트] – [유닛 테스트 뷰] 페이지에서 함수 노드 표시 옵션을 [모든 함수]로 선택합니다.



4. 테스트를 생성하지 않은 함수들을 유닛 테스트 뷰에서 볼 수 있습니다.



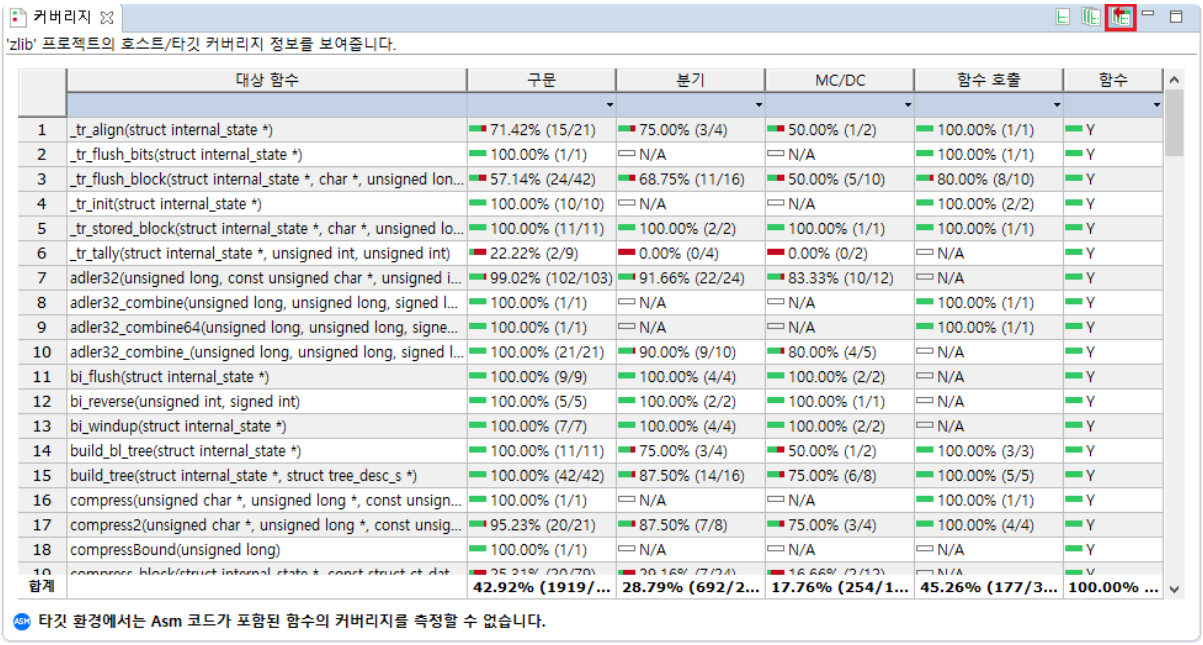
5. 유닛 테스트 뷰에서 체크되지 않은 항목(하위 테스트가 없는 항목)만, Ctrl + 클릭을 통해 복수 선택한 뒤 컨텍스트 메뉴에서 [테스트 생성]을 합니다.



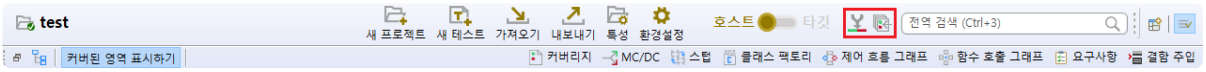
6. 위의 과정을 거치면 소스에 새로 추가된 함수에 대한 테스트를 생성할 수 있습니다.

1.50. 테스트 케이스 선택 시 커버된 영역의 색상이 변경되지 않는 경우

커버리지 표시 옵션이 전체 커버리지 보기(외부 커버리지 포함) 상태일 때에는 테스트 또는 테스트 케이스 별 커버리지가 표시되지 않습니다.
전체 커버리지 보기(외부 커버리지 포함)를 해제한 뒤에 테스트 또는 테스트 케이스 별 커버리지를 확인하시기 바랍니다.



	대상 함수	구문	분기	MC/DC	함수 호출	함수
1	_tr_align(struct internal_state *)	71.42% (15/21)	75.00% (3/4)	50.00% (1/2)	100.00% (1/1)	Y
2	_tr_flush_bits(struct internal_state *)	100.00% (1/1)	N/A	N/A	100.00% (1/1)	Y
3	_tr_flush_block(struct internal_state *, char *, unsigned lon...	57.14% (24/42)	68.75% (11/16)	50.00% (5/10)	80.00% (8/10)	Y
4	_tr_init(struct internal_state *)	100.00% (10/10)	N/A	N/A	100.00% (2/2)	Y
5	_tr_stored_block(struct internal_state *, char *, unsigned lo...	100.00% (11/11)	100.00% (2/2)	100.00% (1/1)	100.00% (1/1)	Y
6	_tr_tally(struct internal_state *, unsigned int, unsigned int)	22.22% (2/9)	0.00% (0/4)	0.00% (0/2)	N/A	Y
7	adler32(unsigned long, const unsigned char *, unsigned i...	99.02% (102/103)	91.66% (22/24)	83.33% (10/12)	N/A	Y
8	adler32_combine(unsigned long, unsigned long, signed l...	100.00% (1/1)	N/A	N/A	100.00% (1/1)	Y
9	adler32_combine64(unsigned long, unsigned long, signe...	100.00% (1/1)	N/A	N/A	100.00% (1/1)	Y
10	adler32_combine_(unsigned long, unsigned long, signed l...	100.00% (21/21)	90.00% (9/10)	80.00% (4/5)	N/A	Y
11	bi_flush(struct internal_state *)	100.00% (9/9)	100.00% (4/4)	100.00% (2/2)	N/A	Y
12	bi_reverse(unsigned int, signed int)	100.00% (5/5)	100.00% (2/2)	100.00% (1/1)	N/A	Y
13	bi_windup(struct internal_state *)	100.00% (7/7)	100.00% (4/4)	100.00% (2/2)	N/A	Y
14	build_bl_tree(struct internal_state *)	100.00% (11/11)	75.00% (3/4)	50.00% (1/2)	100.00% (3/3)	Y
15	build_tree(struct internal_state *, struct tree_desc_s *)	100.00% (42/42)	87.50% (14/16)	75.00% (6/8)	100.00% (5/5)	Y
16	compress(unsigned char *, unsigned long *, const unsign...	100.00% (1/1)	N/A	N/A	100.00% (1/1)	Y
17	compress2(unsigned char *, unsigned long *, const unsig...	95.23% (20/21)	87.50% (7/8)	75.00% (3/4)	100.00% (4/4)	Y
18	compressBound(unsigned long)	100.00% (1/1)	N/A	N/A	N/A	Y
19	compress_block(struct internal_state *, const struct st_dat...	25.31% (20/79)	29.16% (7/24)	16.66% (2/12)	N/A	Y
합계		42.92% (1919/...	28.79% (692/2...	17.76% (254/1...	45.26% (177/3...	100.00% ...



1.51. CT2.9에서 CT3.0로 버전 변경 시 프로젝트가 열리지 않는 문제

문제를 해결하기 위해서는 아래의 2가지 방법을 시도할 수 있습니다.

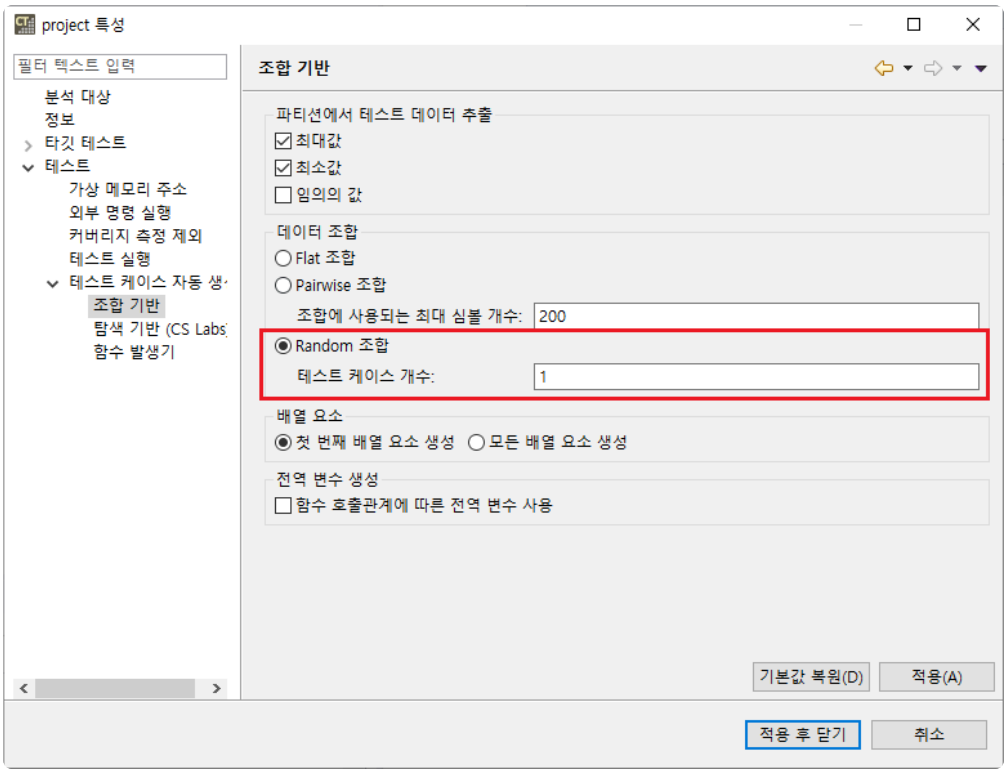
- Controller Tester에서 프로젝트 가져오기
  1. 새로 워크스페이스 생성
  2. CT 3.0 버전에 있는 [가져오기] – [일반] – [기존 프로젝트를 워크스페이스로] 기능을 사용하여 프로젝트 가져오기.

- 파일 시스템에서 프로젝트 복사하기
  1. 새 워크스페이스에서 기존 프로젝트 이름과 동일한 빈 프로젝트를 생성
  2. 파일 시스템에서 기존 워크스페이스의 프로젝트 디렉터리를 그대로 복사
  3. Controller Tester에서 [프로젝트 닫기]를 한 후 다시 열기

위의 방법으로도 문제가 해결되지 않는다면, 설치된 3.0 패키지 형상에 문제가 있을 가능성이 있으니, 패키지 재설치 후에 다시 시도해보시기 바랍니다.

1.52. 테스트 생성 시 테스트 케이스가 1개만 생성되도록 설정하는 방법

[프로젝트 특성] > [테스트] > [테스트 케이스 자동 생성] > [조합 기반]에서 데이터 조합 옵션을 변경하여 테스트 케이스의 개수가 1개만 생성되도록 설정할 수 있습니다.



1.53. 테스트 실행 후 통합 커버리지가 0으로 표시되거나 실행 결과가 보이지 않는 문제

[프로젝트 특성] > [테스트] > [가상 메모리 주소] 에서 가상 주소를 설정한 프로젝트의 경우, 가상 주소의 범위가 테스트 실행 환경의 시스템 메모리 가용 범위를 넘기면 (ex. 0×0-0×60000400) 아래와 같은 문제가 발생할 수 있습니다.

- 커버리지 결과가 정상적으로 표시되던 테스트의 통합 커버리지가 0으로 표시됨
- 테스트의 실행 결과가 실행 후에도 표시되지 않음

설정한 가상 주소의 범위를 줄인 후 다시 테스트를 실행해보시기 바랍니다.

1.54. (Ver.3.6 이후) 테스트 데이터를 ‘덮어쓰기’로 가져오고 싶을 때

Controller Tester 3.5부터는 테스트 데이터 가져오는 방식이 ‘이어쓰기’로 설정되어, 기존 데이터를 덮어쓰지 않습니다.
Controller Tester 3.6부터 테스트 데이터 가져오는 방식을 설정할 수 있습니다. 기본값은 ‘이어쓰기’이며, 아래는 가져오는 방식을 ‘덮어쓰기’로 설정하는 방법입니다.

1. `{제품 설치 경로}\plugins\com.codescroll.gp.rcp.helios_xx\plugin_customization.ini` 파일을 연다.
2. com.codescroll.ut/import.testdata.append 옵션을 false로 바꾸고 저장한다.

Controller Tester을 실행하면 테스트 데이터를 ‘덮어쓰기’ 방식으로 가져올 수 있습니다.

2. Controller Tester Target Plug-in 문제 해결 가이드

2.1. 타깃 테스트 코드 내보내기 후 빌드 이슈

2.1.1. entry point 함수 이름이 main이 아닌 경우 (CT 3.2 이전 버전)

Controller Tester Target Plugin은 원본 main 함수를 cs_renamed_main 함수로 이름을 대체하여, 타깃에서 소프트웨어를 실행하면 Controller Tester에서 정의한 main 함수가 수행되도록 합니다.
원본의 entry point 함수 이름이 main이 아닌 경우에는 cs_tfx.c와 cs_build_macro.h 파일을 수정해야 합니다.
cs_tfx.c 파일은 Controller Tester 설치 경로에 있지만, cs_build_macro.h는 '타깃 테스트 코드 내보내기'를 할 때마다 생성되는 파일이기 때문에 패치를 적용해야 문제를 해결할 수 있습니다.

entry point 함수 이름 문제가 의심되는 경우, 아래의 코드를 수정하여 확인할 수 있습니다.

cs_tfx.c

```
#if defined main /* normal */
#undef main // entry point 함수 이름으로 교체
#endif
```

cs_build_macro.h

```
#if !defined CS_START_FROM_IUT
#define main        __cs_renamed_main //entry point 함수명으로 교체
```

2.1.2. 타깃 테스트 결과를 저장할 때 사용하는 함수의 multiple definition 오류 (CT 3.2 이전 버전)

타깃 테스트 결과는 타깃에서 테스트를 실행하면 '타깃 로그' 형태로 출력됩니다. 타깃 로그는 memset 함수를 사용하여 데이터를 저장하기 때문에 memset 함수를 사용하기 위해 cs_tfx.h 파일에 extern 키워드로 선언되어 있습니다.
'타깃 테스트 코드 내보내기'를 할 때 원본 소스 코드에 cs_tfx.h 파일이 include 됩니다. 이 때, 테스트 대상 소스 코드에서 memset이나 strncpy 함수가 include 되어있는지 여부에 따라 multiple definition 오류가 발생할 수 있습니다.

```
ex) multiple definition of `memset' error
```

해당 오류가 발생한 경우에, Controller Tester 설치 경로의 cstfx.h 파일에서 multiple 오류가 발생한 라인을 주석처리한 뒤에 다시 '타깃 테스트 코드 내보내기'를 하면 문제를 해결할 수 있습니다.

```
cstfx.h 경로 예시: C:\Program Files\CodeScroll Controller Tester 3.1\plugins\com.codescroll.ut_3.1.2\target\lib\controller_nc\cs_tfx.h
```

2.1.3. ‘sprintf’ has not been declared 혹은 CS_FLT_OUTPUT 오류 (CT 3.2이전 버전)

테스트 대상 소스 코드에서 sprintf 함수를 사용할 수 없는 경우가 있습니다.
이 경우에는 Controller Tester 설치 경로의 cstfx.h 파일에 아래 코드를 추가하면 문제를 해결할 수 있습니다.

```
cstfx.h 경로 예시: C:\Program Files\CodeScroll Controller Tester 3.1\plugins\com.codescroll.ut_3.1.2\target\lib\controller_nc\cs_tfx.h
```

cs_tfx.h 파일의 210라인에

```
int sprintf(char* str, const char* format, ...);
```

추가

2.1.4. 타깃 로그 인터페이스 설정

타깃 로그 인터페이스 파일 이름은 cs_io_implementation.c 입니다. '타깃 테스트 코드 내보내기'를 수행하면 소스 파일마다 타깃 로그 인터페이스 파일을 include 합니다.
타깃 로그 인터페이스의 cs_io_putbyte 함수를 사용하여 char 단위로 로그를 출력하며, 타깃 로그 인터페이스 구현에 따라 Ethernet, UART, JTAG 등 다양한 방식으로 타깃 로그를 출력할 수 있습니다.
사용하는 통신방식에 따라 적절한 초기화 로직을 cs_io_initialize 함수에 구현해야 합니다. 특히, UART를 사용할 때 baudRate나 dataBits 등의 설정을 맞춰주지 않으면, 타깃 로그가 잘못 출력되는 문제가 발생할 수 있습니다.
타깃 시험을 위해 정확하게 설정하려면, 타깃과 소프트웨어에 대한 이해가 필요하므로, 타깃 개발 담당자에게 문의해야 합니다.

2.1.5. signal 오류 확인

타깃 환경에서 테스트가 실행되는 도중에 오류가 발생했는지 여부는 타깃 로그가 출력되는 형태를 보면 확인할 수 있습니다.
타깃 로그를 확인하면 테스트 케이스 수행 중에 오류가 발생했는지, 아니면 테스트 케이스가 수행되기 전에 오류가 발생했는지를 확인할 수 있습니다.
정상적으로 테스트 실행이 완료되면 타깃 로그 마지막에 “CSET“가 출력됩니다. 만약 “CSTC“가 출력된 뒤에 로그가 끊겼다면, 해당 테스트 케이스를 프로젝트 DB에서 찾아야 합니다.

```
프로젝트 DB 파일: [프로젝트 특성] - [정보] - [위치]의 프로젝트 경로 아래 .csdata 디렉터리의 { ProjectName }.csp 파일
테스트케이스 출력 포멧: CSTC<TEST_ID, TEST_CASE_NO>
```

타깃 로그 출력 예시

```
CSTR<374069146,130789013,1,0,1>CSTR#
CSST<test,1574932569>CSST#
CSTC<133143986176,1>CSTC#
CSOS<1returnVar,0>CSOS#
CSES<1>CSES#
CSTB<1>CSTB#
CSET< test,1574932569>CSET#
```

2.1.6. 타깃 소프트웨어의 UART 통신 관련된 함수를 테스트할 때 문제

UART 통신 설정을 변경하는 테스트 케이스가 수행되면, 이후 타깃 로그가 정상적으로 출력되지 않을 수 있습니다.
(예시: baudrate 설정을 변경하는 테스트 케이스)
이 경우에는 테스트 케이스 편집기의 설정에서 비관리 코드로 전환한 뒤에, 해당 테스트 케이스가 수행되기 전 설정값을 백업하고 테스트 케이스 수행 후 복원하도록 테스트 케이스를 수정(코드로 구현)해야 합니다.

2.1.7. CodeWarrior로 빌드할 때, cs_tfx.c 와 같은 테스트 관련 파일들을 찾지 못하는 문제

CodeWarrior의 프로젝트 설정에서 system path에 아래의 경로가 있는지 확인하고, 없다면 추가해야 합니다.

```
- {CT_Workspace}/.metatdata/.plugins/com.codescroll.ut.embedded/{ ProjectName
}/TestFixture/cs
- {CT_Workspace}/.metatdata/.plugins/com.codescroll.ut.embedded/{ ProjectName
}/TestFixture
```

2.1.8. CodeWarrior에서 float 타입에 대해 타깃 출력값이 나오지 않거나 0으로 출력되는 경우

타깃 테스트에서 실수형 타입의 변수값을 출력할 때 sprintf 함수를 통해 결과값을 가지고 옵니다.
그런데 CodeWarrior에서 제공하는 라이브러리에서 실수형 타입에 대해 sprintf 함수를 사용하지 못하는 경우가 있습니다.
이런 경우에 원본 프로젝트에서 링크하는 라이브러리를 바꾸어 주거나 추가해주어야합니다.
현재 파악된 실수형 타입에 대해 sprintf 함수를 사용할 수 있는 라이브러리는 **libc99_E200z650.a** 입니다.
해당 라이브러리로 교체 후에 cs_tfx.h 파일에 내용을 다음과 같이 수정해주시면 됩니다.

수정 전

```
#define TFX_ftoa_writeBytes(value) \
do {\
        sprintf(buf, "%g", value);\
        TFX_writeBytes(buf);\
} while(0)
```

수정 후

```
 #define TFX_ftoa_writeBytes(value) \
do {\
        sprintf(buf, "%f", value);\
        TFX_writeBytes(buf);\
} while(0)
```

2.1.9. codescroll_int32와 codescroll_uint32 타입의 undefined error

프로젝트에 설정된 툴체인의 info 파일에 codescroll_int32와 codescroll_uint32가 있는지 확인하고, 없다면 해당 타입을 추가해야 합니다. info 파일을 수정한 뒤에는 '타깃 테스트 코드 내보내기'를 다시 수행해야 합니다.

```
툴체인 info 파일 경로: [환경설정] - [툴체인] - [설정 디렉터리 열기]에서 { ToolchainNam
e }.info
```

2.1.10. codescroll_int, codescroll_uint 타입에서 cannot use ‘long’ 또는 undefined type to ‘long’ error가 발생하는 경우

타깃 소프트웨어에서 long 타입을 지원하지 않는 경우에 이 같은 오류가 발생할 수 있습니다. 프로젝트에 설정된 툴체인의 info 파일에 codescroll_int와 codescroll_uint 타입을 int 타입으로 수정하면 됩니다.
수정 후에는 Controller Tester를 재시작해야 합니다.

```
툴체인 info 파일 경로: [환경설정] - [툴체인] - [설정 디렉터리 열기]에서 { ToolchainNam
e }.info
```

```
ex) unsigned long long,unsigned,0,9223372036854775807,8,codescroll_uint
        -> unsigned int,unsigned,0,9223372036854775807,8,codescroll_uint
```

2.1.11. Code Composer Studio에서 cs_io_putbyte의 address를 찾지 못하는 경우

```
SEVERE: Unable to get address for symbol: cs_io_putbyte
org.mozilla.javascript.WrappedException: Wrapped com.ti.ccstudio.scripting.env
ironment.ScriptingException: Unable to get address for symbol: cs_io_putbyte
```

해당 에러가 날 경우에는, 타깃 환경 설정의 실행 탭의 target_binary_path의 경로와 실제 빌드한 바이너리가 다를 수 있습니다.

이러한 경우에는 target_binary_path를 실제 빌드한 바이너리의 경로로 변경해야 정상동작 합니다.

console 로그에서 해당 원인임을 찾아보는 방식은 다음과 같습니다.

```
Finished building target의 .out 파일과 loadProgram: ENTRY sFileName: 의 .out 파일
이 같은 지 확인
```

2.1.12. GreenHills AdaMulti에서 declaration is incompatible with “void cs_io_putbyte” 에러 발생시

AdaMulti에서 빌드시 **declaration is incompatible with “void cs_io_putbyte(codescroll_byte)”** 에러가 출력된다면 codescroll_byte 타입이 char 가 아닌 signed char, unsigned char 로 설정되어 발생한 경우입니다.
해당 프로젝트 분석시 사용했던 툴체인.info 파일에 아래와 같은 형태에서 codescroll_byte 타입을 수정해야 합니다.**(툴체인.info 파일은 ‘환경 설정’ > ‘툴체인’ > ‘설정 디렉터리 열기’ 에서 확인할 수 있습니다.)**

수정 전

```
#typeName,valueKind,min,max,size,csType
signed char,signed,-128,127,1,codescroll_byte
```

수정 후

```
#typeName,valueKind,min,max,size,csType
char,signed,-128,127,1,codescroll_byte
```

수정 후 Controller Tester 의 툴체인 편집창에서 툴체인 선택 후 ‘편집’ > ‘완료’ > ‘적용 후 닫기’ 를 해야 수정된 내용이 반영됩니다.

2.2. TRACE32 관련 이슈

- symbol not found error “ct_target_log”
- target reset failed
- cmm 스크립트 실행 중 ‘&binary_path’ symbol not found 혹은 Data.Load.Elf “{file_path}” /LPATH 위치에서 오류가 발생했을 때

2.2.1. symbol not found error “ct_target_log”

타깃에서 테스트를 수행한 결과가 ct_target_log에 저장됩니다.
ct_target_log를 찾을 수 없는 오류가 발생한 경우, 아래 2가지 내용을 확인해야 합니다.

1. 타깃 테스트 코드가 정상 빌드되었는지 확인
   - 테스트 코드가 정상적으로 적용되지 않았거나 테스트 코드로 빌드한 바이너리가 타깃에 배포되지 않은 경우(원본 소프트웨어만 빌드), 테스트 결과를 저장할 변수를 찾을 수 없다는 오류가 발생할 수 있습니다.

2. cmm 스크립트에서 GO.HLL 명령어의 위치
   - BREAK.SET 명령으로 breakpoint를 지정한 후 디버깅을 시작하는 형태로 cmm 스크립트를 작성해야 합니다.

ex)

```
BREAK.SET cs_io_initialize /Program /cmd " … " /RESUME
BREAK.SET cs_write_log /Program /cmd "…"  /RESUME
BREAK.SET cs_io_finalize /Program /cmd "…" /RESUME /cmd "…" /RESUME

GO.HLL <- BREAK를 설정한 뒤에 입력
```

2.2.2. target reset failed

아래 2가지 내용을 확인해야 합니다.

1. cmm 스크립트에서 System.CPU {CPU_NAME}의 CPU_NAME이 제대로 설정됐는지 확인
2. 타깃 바이너리 크기가 커서 타깃에 로드가 안되는지 확인

2.2.3. cmm 스크립트 실행 중 ‘&binary_path’ symbol not found 혹은 Data.Load.Elf “{file_path}” /LPATH 위치에서 오류가 발생했을 때

Trace32를 이용해서 타깃 테스트를 할 때, Controller Tester는 start.cmm 스크립트에서 target.cmm 스크립트를 호출합니다.
start.cmm 스크립트에서 target.cmm 스크립트를 호출할 때, 파라미터로 “&binary_path“를 넘겨줍니다. 빌드가 정상적으로 되지 않았거나 파라미터로 넘겨주는 경로가 잘못된 경우에 해당 오류가 발생할 수 있습니다.
이 경우에는 빌드가 정상적으로 되었는지 확인하고, start.cmm 스크립트에서 경로를 확인해야 합니다.

2.3. 타깃 로그(테스트 결과) 가져오기 이슈

2.3.1. ‘타깃 로그 가져오기’를 할 때 실패하거나 오류가 발생하는 경우

타깃 로그 파일을 열어 확인해야 합니다. 정상적으로 로그가 출력된 경우 ‘CSET<…>#CSET’ 로그가 마지막에 출력이 되어 있습니다. 만약 이 로그가 출력되지 않았으면 타깃에서 테스트 수행 중에 signal error가 발생했을 가능성이 높습니다.
문제를 해결하려면, 테스트케이스를 하나씩 실행하며 signal 오류를 발생시킨 테스트케이스를 찾아서 수정해야 합니다.

2.3.2. 타깃 로그 버퍼 사이즈를 초과한 경우

cs_io_putbyte 함수의 char 값을 char * buff[BUFF_SIZE] 와 같은 고정된 크기의 버퍼에 값을 저장하는 경우에 테스트 결과가 많으면 버퍼 크기를 초과하는 문제가 발생할 수 있습니다.
문제를 해결하려면 “타깃 로그 인터페이스“에서 BUFF_SIZE를 수정한 뒤에 다시 타깃 테스트 코드를 내보내야 합니다.

2.3.3. Code Composer Studio에서 테스트 빌드 후 실행시 스크립트에서 exception이 발생하는 경우

CCS Caused by: com.ti.ccstudio.scripting.environment.ScriptingException: Could not open session. Found devices matching: .* 에러 발생시

Controller Tester에서 Code Composer Studio 디버거를 사용하여 타깃 테스트를 실행할 때, Javascript로 작성된 실행 스크립트를 사용합니다.
스크립트에서는 사용 가능한 디버깅 프로브로부터 디버깅 세션을 열고 타깃 바이너리를 실행합니다.
기본적으로 디버깅 프로브 및 디버깅 세션은 1개인데, 타깃 환경 설정에 따라 디버깅 프로브가 2개 이상 존재하는 경우가 있으며 이 때에 위와 같은 에러가 발생합니다.
사용하고자 하는 디버깅 프로브를 특정하면 오류를 해결할 수 있습니다.
디버깅 프로브는 프로젝트 특성 > 타깃 테스트 > 타깃 환경 > 실행 탭의 **debug_probe** 옵션에서 설정할 수 있습니다.

> 어떤 디버깅 프로브를 선택해야 할지 모를 때, Code Composer Studio IDE에서 디버깅 모드로 대상 프로젝트를 실행 후 debug view에 표시되는 디버깅 프로브 값을 debug_probe 옵션으로 설정합니다.

invalid target memory page 에러 발생시

Code Composer Studio 프로젝트를 빌드하면 .map 파일이 생성됩니다. .map 파일은 변수들이 할당된 주소, 페이지 위치에 대한 정보를 가지고 있습니다.
Controller Tester에서 제공하는 Code Composer Studio 실행 스크립트에서는 data 영역에 타깃 테스트 결과값을 저장하고 이 값을 가져오도록 구현되어 있습니다.
하지만 data영역이 아닌 program 영역에 저장되는 경우 스크립트 실행시 메모리 영역에 접근할 수 없다는 에러 메시지를 출력하는 경우가 있습니다.
프로젝트 특성 > 타깃 테스트 > 타깃 환경 > 실행 탭의 **char_format** 옵션을 확인해보시기 바랍니다.

2.3.4. 타깃 디버그 정보 로그를 가져오기시 실패하는 경우

타깃 테스트시 디버거가 없는 환경에서 '디버그 정보 확인' 기능을 사용하면 디버깅 정보를 로그를 통해 확인할 수 있습니다.
타깃 테스트 환경에서 '디버그 정보 확인'을 통해 테스트 코드를 내보내어 빌드 및 타깃 실행을 하게 되면 커버리지 측정값이 아닌 디버깅 정보가 출력되게 됩니다.
이 때, 기본 로그 출력 사이즈가 1000으로 지정되어 있는데 로그 정보가 1000글자를 넘어서면 그대로 실행이 끝나게 되어 디버깅 로그가 잘리는 현상이 발생할 수 있습니다.
프로젝트별로 생성되는 ut.ini 파일에서 **TARGET_DEBUG_PROBE_INDEX** 옵션을 통해 로그 출력 사이즈를 조절할 수 있습니다.

ut.ini 파일 위치 : Controller Tester_프로젝트_경로\.csdata\ut.ini
옵션 : TARGET_DEBUG_PROBE_INDEX=1000

2.3.5. ‘타깃 로그 가져오기’ 실패시 타깃 로그로 확인하는 방법

테스트 실행 결과는 아래와 같은 패턴으로 출력됩니다.

```
CSTR<212668380,3136998082,2,0>CSTR#
CSST<ct_test_log,425747>CSST#
CSTC<4294967296,2>CSTC#
CSOS<2s_aDoJobCnt[0],0>CSOS#
CSES<1>CSES#
CSTB<110011000000100101100000>CSTB#
CSET<ct_test_log,425747>CSET#
```

- 2번 테스트 케이스 실행시 시그널 에러가 발생한 경우

```
CSTR<212668380,3136998082,2,0>CSTR#
CSST<ct_test_log,425747>CSST#
CSTC<4294967296,2>CSTC#// CSTC 태그 사이에 2번째에 있는 번호가 테스트 케이스 번호를 나타냄
```

```
CSTR<212668380,3136998082,2,0>CSTR#
CSST<ct_test_log,425747>CSST#
CSTC<4294967296,2>CSTC#
CSOS<2s_aDoJobCnt[0],0>CSOS#
```

- 테스트 케이스의 특정 변수가 null인데 출력값을 체크한 경우

```
CSTR<212668380,3136998082,2,0>CSTR#
CSST<ct_test_log,425747>CSST#
CSTC<4294967296,2>CSTC#
CSOS<2s_aDoJobCnt[0], //s_aDoJobCnt[0] 변수값이 null이라 값 출력 중 에러 발생
```

- 테스트 실행중 무한 루프로 인해 커버리지 결과가 출력되지 않는 경우

```
CSTR<212668380,3136998082,2,0>CSTR#
CSST<ct_test_log,425747>CSST#
CSTC<4294967296,2>CSTC#
CSOS<2s_aDoJobCnt[0],0CSOS#
CSES<1>CSES#
CSTB<11001100000010010110  //커버리지 비트맵이 출력중에 무한 루프로 인해 더 이상 출력되지 않음
```

2.4. 기타 팁

- .map 파일
- TRACE32 디버깅
- 호스트/타깃 환경의 byte order 때문에 출력값이 다른 경우

2.4.1. .map 파일

타깃 소프트웨어를 빌드한 뒤에 바이너리가 생성되는 경로에 .map 파일이 있는 경우가 있습니다. 해당 파일에는 타깃 바이너리에 포함된 심볼 정보가 기록되어 있습니다. main 함수가 정상적으로 치환되었는지 확인하거나 실행 파일에 포함된 함수 심볼을 확인할 수 있습니다.

2.4.2. TRACE32 디버깅

TRACE32를 사용하는 경우에 테스트용 바이너리를 디버깅할 수 있습니다.

1. TRACE32를 실행하고 cmm 스크립트 파일(start.cmm)을 엽니다.
2. “Debug“를 실행하고, 스크립트의 Go.HLL 이후에 breakpoint 설정한 뒤에 “Continue“를 실행합니다.
3. Break가 걸렸을 때, [view]-[Var]를 선택하면 현재 타깃에서 수행 중인 변수와 함수 정보를 볼 수 있습니다.

2.4.3. 호스트/타깃 환경의 byte order 때문에 출력값이 다른 경우

호스트 환경과 타깃 환경의 byte order가 다르면 호스트 출력값과 타깃 출력값이 달라져서 테스트가 실패하는 경우가 있습니다.

호스트/타깃의 출력값이 다른 경우

소스 코드에서 메모리를 직접 다루는 경우, 호스트 환경과 타깃 환경의 byte order에 따라 출력값이 달라질 수 있습니다. 동일한 기대값으로 테스트를 할 경우 호스트 테스트 또는 타깃 테스트가 실패하게 됩니다. 이때, 기대값에 수직 바(|)를 사용하여 동일한 테스트 케이스로 호스트/타깃 테스트를 진행할 수 있습니다.

- 예시
  - 호스트 출력값 : `0x1122`
  - 타깃 출력값 : `0x2211`
  - 입력할 기대값 : `0x1122|0x2211`

소스 코드 로직이 byte order에 완전히 종속된 경우

호스트 시험에서 메모리 사용 방법에 의존성이 있는 로직은 정상적인 결과를 얻을 수 없습니다. 메모리 사용 방법에 의존성이 있는 경우는 다음과 같습니다.

- union을 사용하여 메모리 세부 정보를 직접 접근하고 사용하는 경우
- 컴파일러에 따라 구현이 달라질 수 있는 비트 필드에 대한 패킹

위와 같은 형태의 코드는 이식성을 고려해볼때 지양해야 합니다. 테스트 결과를 확인할 때에도, 메모리 값을 직접 확인하기보다 자료형의 값을 확인하도록 설계하는 것이 좋습니다.

만약, 소스 코드를 수정하기 어려운 상황이라면, 이와 같은 코드에 대한 테스트는 실제 타깃 환경에서 수행하거나 호스트와 타깃 테스트를 구분해서 설계하고 관리해야 합니다.