Controller Tester Troubleshooting Guides

3.3 — Last update: Aug 21, 2020

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(sizeof(int),1); 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\v6.1\Include

@set INCLUDE=C:\Program Files\Microsoft SDKs\Windows\v6.1\Include;%VCINSTALLDIR%\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 실행시 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.
프로젝트의 ‘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 최신 sdk 버전이 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.
프로젝트의 ‘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”

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 주석 처리한 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\co
m.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 { ToolchainName }.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 { ToolchainName
}.info
```

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

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

- ‘타깃 로그 가져오기’를 할 때 실패하거나 오류가 발생하는 경우
- 타깃 로그 버퍼 사이즈를 초과한 경우

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],0CSOS#
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<4294967296,2CSTC#
CSOS<2s_aDoJobCnt[0],0CSOS#
```

- 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

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.

3. Controller Tester 고객지원용 가이드

3.1. 타깃에서 빌드 스텁 사용 방법 (타깃 전용 프로젝트는 사용불가)

> **아래의 가이드는 테스트에 영향을 줄 수 있기 때문에 사업지원팀과 개발팀에 먼저 문의를 주시기 바랍니다.**

먼저 기본적으로 타깃에서는 빌드 스텁을 사용하지 못하는 것이 맞습니다.
하지만 실제로 호출되는 함수가 테스트 데이터로 인해 인터럽트를 발생시키는 경우는 테스트가 정상적으로 수행이 되어도 결과를 받을 수가 없습니다.
이런 경우 아래의 3가지 방법 중 하나로 빌드 스텁을 사용할 수 있도록 처리해주어야 합니다.

1. 타깃 빌드 시점에 처리하는 방법

이 방법은 실제 개발자한테 문의를 해야 하는 방법입니다.
타깃 테스트 내보내기 시 환경 설정에서 빌드 스텁 사용으로 체크하신 후 makefile 혹은 타깃 IDE에서 함수 정의가 있는 라이브러리나 소스 파일을 빌드시 제외시키는 방법입니다.

2. cs_replace_code 옵션을 사용해서 처리하는 방법

cs_replace_ code 옵션으로 강제로 함수 호출부 이름을 바꾸는 방법입니다. 실제 함수의 호출부를 바꾸는 방법이기 때문에 권장하는 방법은 아닙니다.

예제 코드

```
void function();
int main(){
   function();// stub 처리가 필요한 함수
   return 0;
}
```

툴체인.conf에 아래 옵션 추가
cs_replace_code=functionfunction_stub

3. —preinclude 혹은 -include 옵션을 통한 스텁 처리

시스템 스텁을 사용자 스텁으로 처리하여 사용하는 방법입니다.

예제코드

```
int main(){
   function();// stub 처리가 필요한 함수
   return 0;
}
```

임의의 헤더파일을 만든 후에 아래와 같이 함수 정의를 추가합니다.

```
#ifndef _STUB_HEADER_
#define _STUB_HEADER_
void function(){
}
#endif
```

Controller Tester 프로젝트에서 예제코드가 있는 소스파일 우클릭 > 특성 > 컴파일 플래그에 다음과 같이 매크로로 치환을 합니다.
-Dfunction=function_stub

Controller Tester 프로젝트 우클릭 후 [커버리지 측정 제외] > [함수 단위 제외 목록]에 function_stub 함수를 추가해주시면 됩니다.

> **!** 3번의 방법에서 임의로 만든 헤더 파일에 시스템 헤더를 추가하여 사용하는 경우에는 헤더 파일이 전처리에 여러 번 포함되어 호스트 실행시 multiple definition 에러가 발생할 수 있습니다.

3.2. Code Composer Studio 환경에서 타깃 로그가 정상적으로 출력되지 않는 경우

signal 에러도 발생하지 않았는데 타깃 테스트 로그가 정상적으로 출력되지 않는 경우가 있을 수 있습니다.
Code Composer Studio에서 제공하는 pragma가 영향을 줄 수 있습니다.

테스트 대상 함수에 아래와 같은 코드가 있는지 확인을 해보시기 바랍니다.

```
#pragma CODE_SECTION("example_function", "ramfuncs")
void example_function(){
}
```

위와 같은 코드가 있을 경우 함수가 ram영역에 저장이 되고, 실행은 flash 영역에서 일어나기 때문에 해당 함수를 호출하기 위해서는 flash영역에 load하는 코드가 필요하게 됩니다.
아래의 코드가 원본에 존재하는지 확인 후 로그 인터페이스의 cs_io_initialize()에서 해당 함수를 호출하도록 수정해야 합니다.

```
Memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (Uint32)&RamfuncsLoadSize);
```

수정 후에도 문제가 발생한다면 툴체인.conf에 아래의 옵션을 추가해 보시기 바랍니다.
cs_replace_code=#pragma CODE_SECTION//#pragma CODE_SECTION

3.3. vector autosar에서 만든 makeFile을 사용하는 빌드 스크립트 사용시 clean build가 되지 않는 문제

vector autsar에서 만든 makeFile은 incremental build를 수행하기 때문에 clean 명령 없이 빌드가 가능합니다.
이때 빌드 대상 파일이 변경됐는지 여부를 확인할 때 dep/소스파일.c.d 파일을 통해 확인하는 것으로 보입니다.
그래서 타깃 테스트를 수행할 때 대상 소스 코드를 수정해도 dep/소스파일.c.d 파일은 변경이 되지 않아 rebuild가 되지 않는 문제가 있을 수 있습니다.
이런 경우 배치 파일에 테스트 대상 소스파일의 c.d파일들을 지우는 명령을 추가해야 합니다.