CT USER GUIDES

SOFTWARE FOR
SAFE WORLD

SURESOFT

User Guides

2023.12 — Last update: Dec 14, 2023

Suresofttech

Copyright © 2023 Suresofttech

Table of Contents

1. 소스 코드 변경 후 테스트 재설정

테스트를 설계한 이후에 대상 소스 코드가 변경될 수 있습니다. CT 2023.12은 소스 코드의 변경을 감지하고 영향을 받는 테스트의 수정을 도와주는 [테스트 재설정] 기능을 제공합니다.

> ✻ RTV 프로젝트와 RTV 타깃 프로젝트의 경우 [테스트 재설정] 기능을 사용하기 전에 [RTV 소스 파일 새로 고치기] 기능을 이용하여 변경된 소스 코드를 반영해야 합니다.

CT 2023.12은 소스 코드의 변경을 여섯 가지로 구분하여 감지합니다.

- 테스트 대상 함수의 클래스 이름이 변경된 경우
- 테스트 혹은 스텁의 대상 함수 이름이 변경된 경우
- 테스트에서 사용하는 전역 변수의 이름 또는 타입이 변경된 경우
- 클래스 코드의 대상 클래스의 멤버 함수 이름이 변경된 경우
- 테스트 대상 함수의 리턴 타입 또는 파라미터의 이름이나 개수가 변경된 경우
- 결함 주입 대상 함수의 코드가 변경된 경우

CT 2023.12에서 감지하는 경우는 본 문서의 자동으로 변경을 감지하는 경우 페이지를, 감지하지 않는 경우는 자동으로 변경을 감지하지 않는 경우 페이지를 참고하시기 바랍니다.

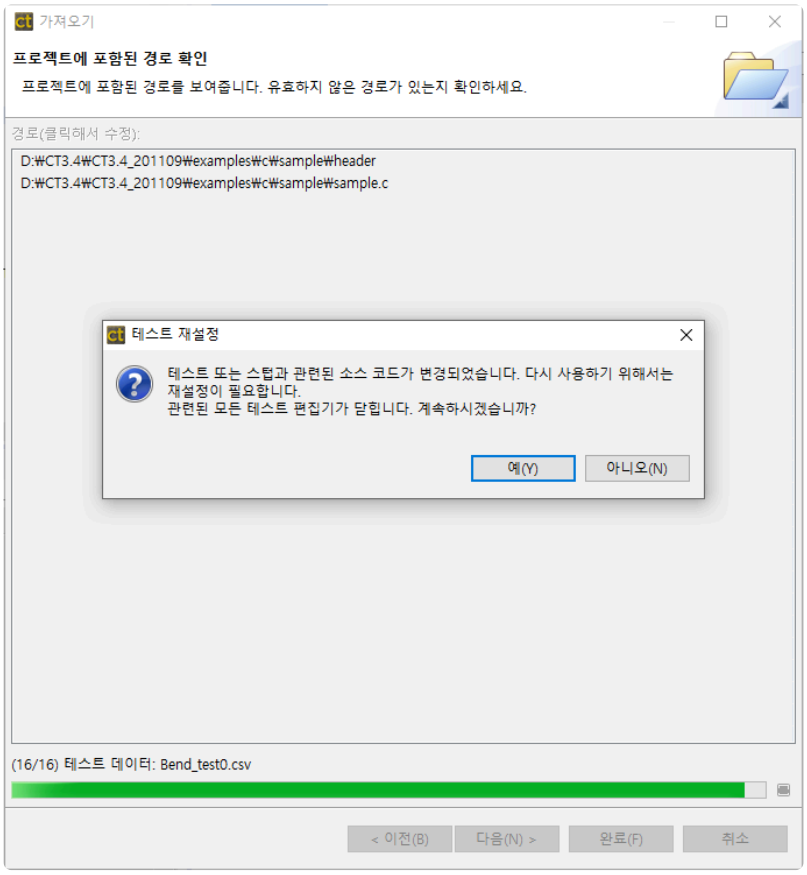
1.1. [테스트 재설정] 실행하기

자동으로 [테스트 재설정] 기능을 이용하는 방법

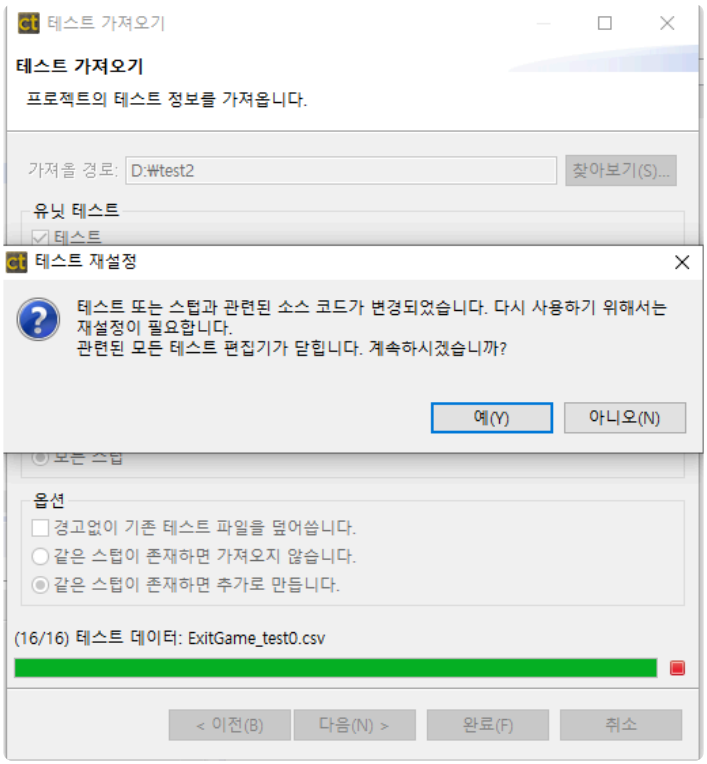
다음 네 가지의 경우에 CT 2023.12가 변경을 자동으로 감지합니다.

- [프로젝트 가져오기] 기능을 사용할 때, 가져온 프로젝트 정보와 현재 프로젝트 정보가 다른 경우
- [테스트 가져오기] 기능을 사용할 때, 가져온 테스트와 현재 프로젝트 정보가 다른 경우
- 프로젝트를 분석한 후 소스 코드 변경이 감지된 경우
- 결함 주입이 불가능한 위치에 결함 주입 코드를 작성한 후 프로젝트를 분석한 경우

[프로젝트 가져오기] 기능을 사용할 때, 가져온 프로젝트 정보와 현재 프로젝트 정보가 다른 경우



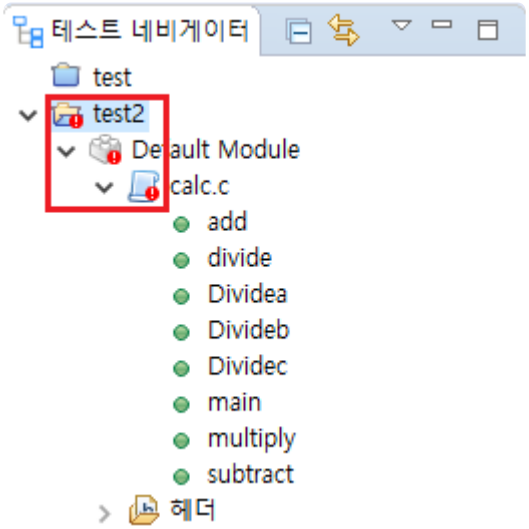
[테스트 가져오기] 기능을 사용할 때, 가져온 테스트와 현재 프로젝트 정보가 다른 경우



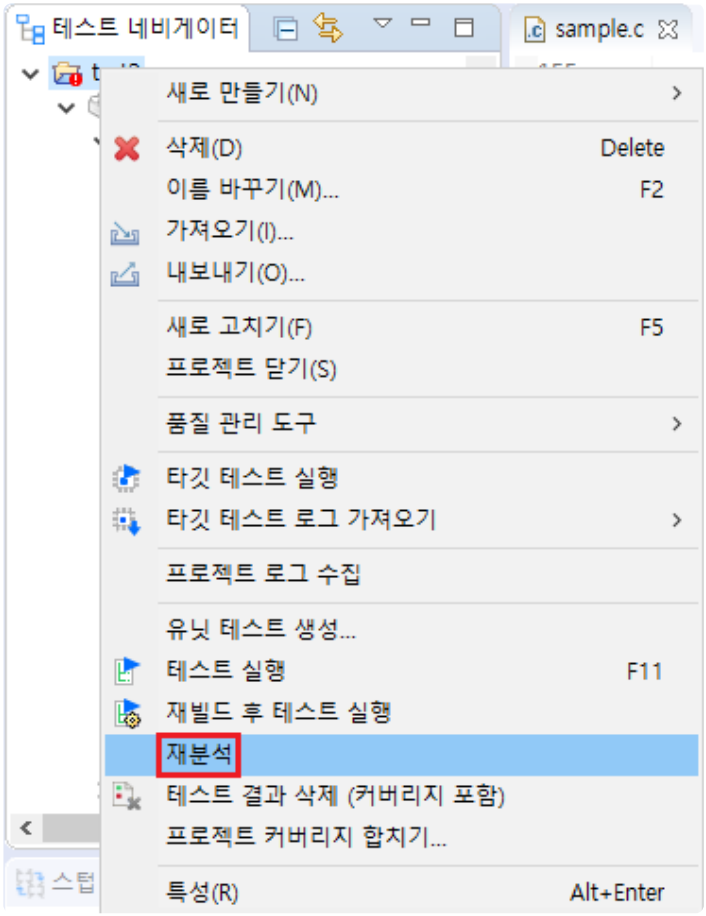
프로젝트를 분석한 후 소스 코드 변경이 감지된 경우

프로젝트를 분석하거나 재분석을 한 후 소스 코드의 변경이 감지되면 [테스트 재설정] 기능을 사용할 수 있습니다.

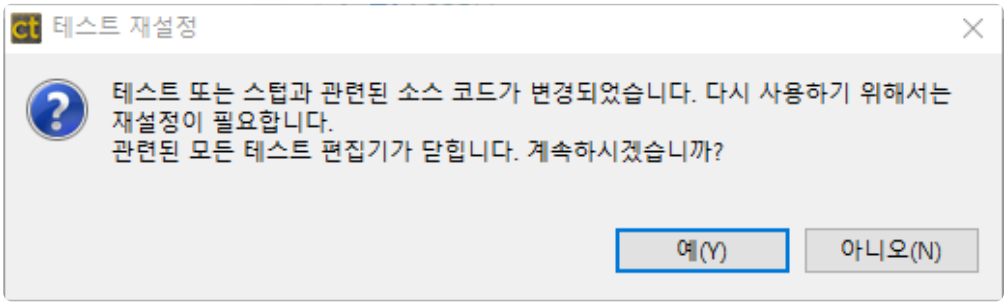
1. 소스 코드가 변경되면 [테스트 네비게이터 뷰]에 변경 여부가 표시됩니다.



2. 프로젝트 컨텍스트 메뉴의 [재분석]을 선택하거나 테스트 실행을 하여 소스 코드를 분석합니다.



3. [테스트 재설정] 대화상자의 [예(Y)] 버튼을 클릭하면 재설정 대화상자가 뜹니다.

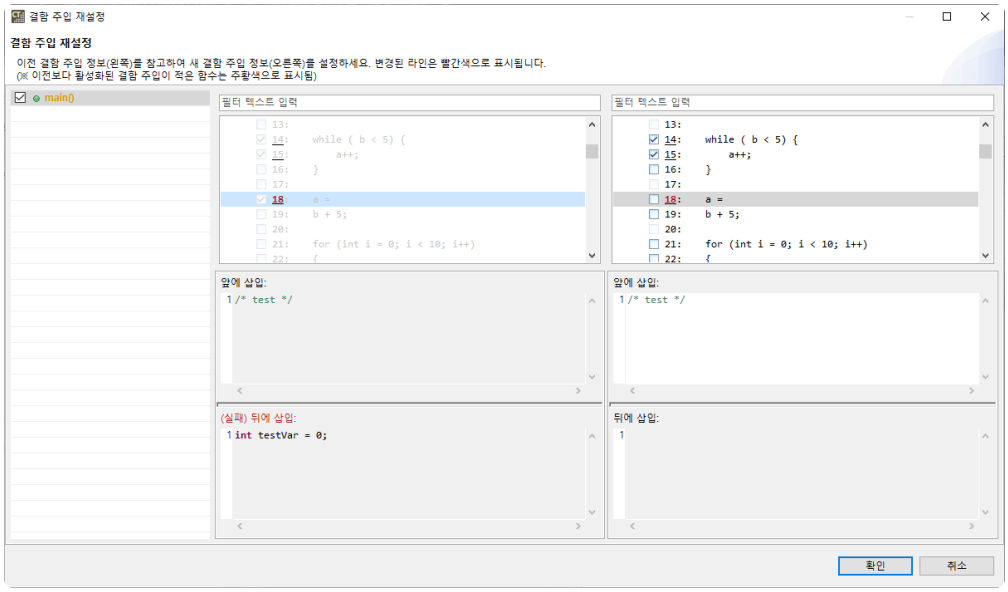


결함 주입이 불가능한 위치에 결함 주입 코드를 작성한 후 프로젝트를 분석한 경우

프로젝트를 재분석할 때, 아래 조건을 만족하는 결함 주입 정보가 존재하면 결함 주입 재설정 대화상자가 뜹니다.

- 결함을 주입할 수 없는 위치의 라인을 활성화하고 결함 주입 코드를 작성한 경우

결함 주입 재설정 대화상자에서 결함을 주입할 수 없는 위치와 결함 주입 정보를 확인할 수 있습니다.



결함 주입 재설정 대화 상자를 통해 이전에 작성했던 결함 주입 정보를 재사용할 수 있습니다.

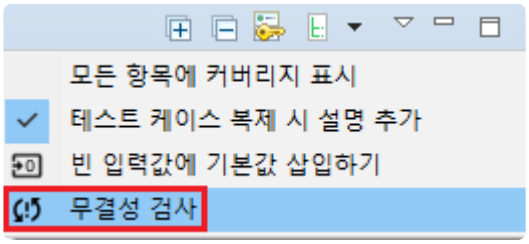
수동으로 [테스트 재설정] 기능을 이용하는 방법

[클래스 재설정] 다이얼로그에서 [아니오(N)]를 선택하면 아래의 방법으로 [클래스 재설정] 기능을 실행할 수 있습니다.

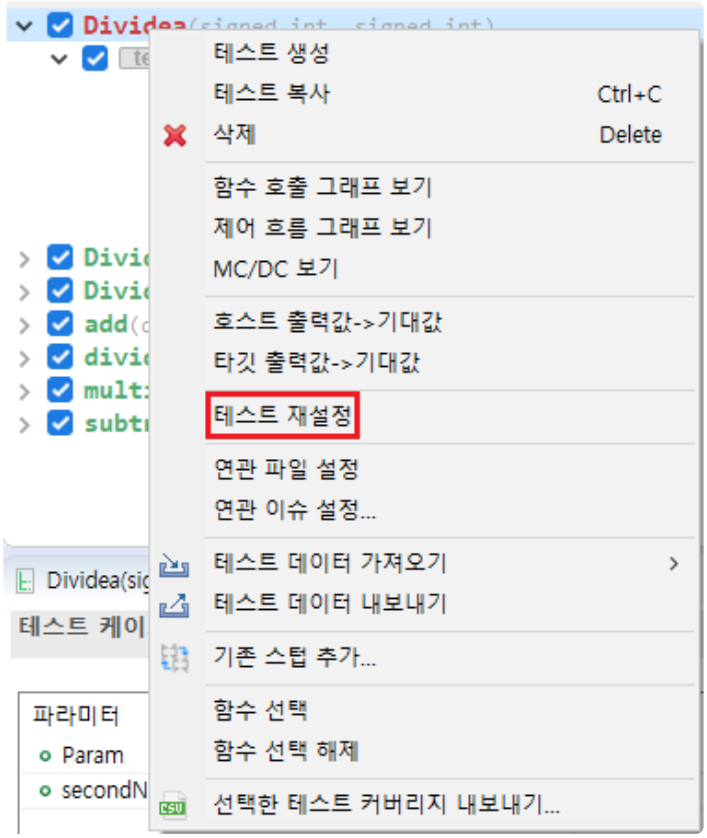
- [클래스 팩토리 뷰]의 풀다운 메뉴에서 [클래스 재설정]을 선택합니다.

[테스트 재설정] 다이얼로그에서 [아니오(N)]를 선택하거나 테스트 재설정 중에 취소를 눌러 종료하면 아래의 세 가지 방법으로 [테스트 재설정] 기능을 실행할 수 있습니다.

- [유닛 테스트 뷰]의 풀다운 메뉴에서 [무결성 검사]를 선택합니다.



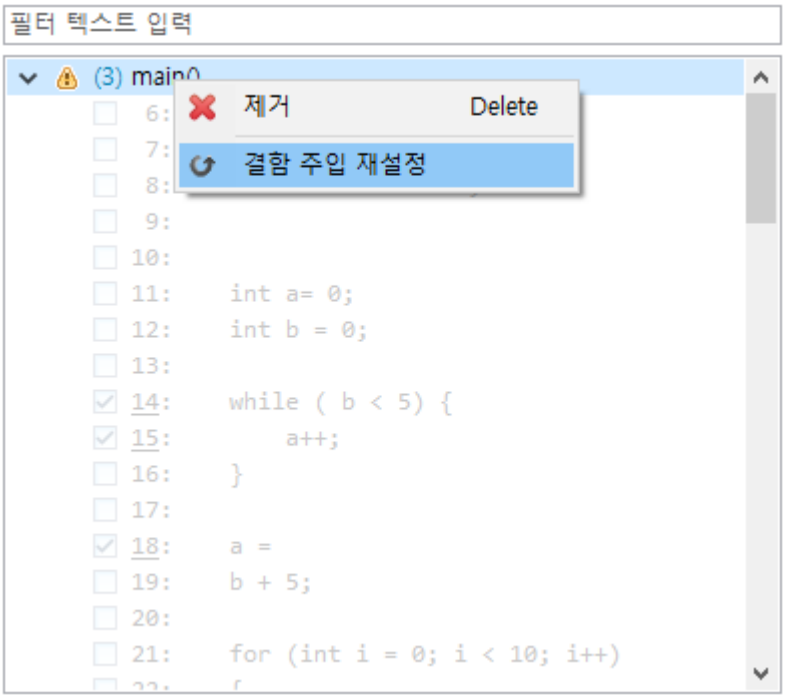
- [유닛 테스트 뷰]의 함수나 테스트의 컨텍스트 메뉴에서 [테스트 재설정]을 선택하여 [테스트 재설정] 기능을 사용합니다.



[테스트 재설정]의 경우, 기존 테스트를 참고하여 새 테스트를 설계할 수 있습니다.

- [결함 주입 뷰]의 풀다운 메뉴나 컨텍스트 메뉴로 [결함 주입 재설정]을 실행할 수 있습니다.
  - [결함 주입 뷰] 우측 상단의 메뉴에서 [결함 주입 재설정] 기능을 사용합니다.

  - [결함 주입 뷰]에서는 재설정이 필요한 결함 주입 함수를 재설정 필요 상태 로 표시합니다. [결함 주입 재설정]은 재설정이 필요한 결함 주입 함수를 더블클릭하거나 우클릭하여 실행할 수 있습니다.



재설정 필요 상태의 결함 주입 함수는 결함 주입 정보를 수정할 수 없습니다.

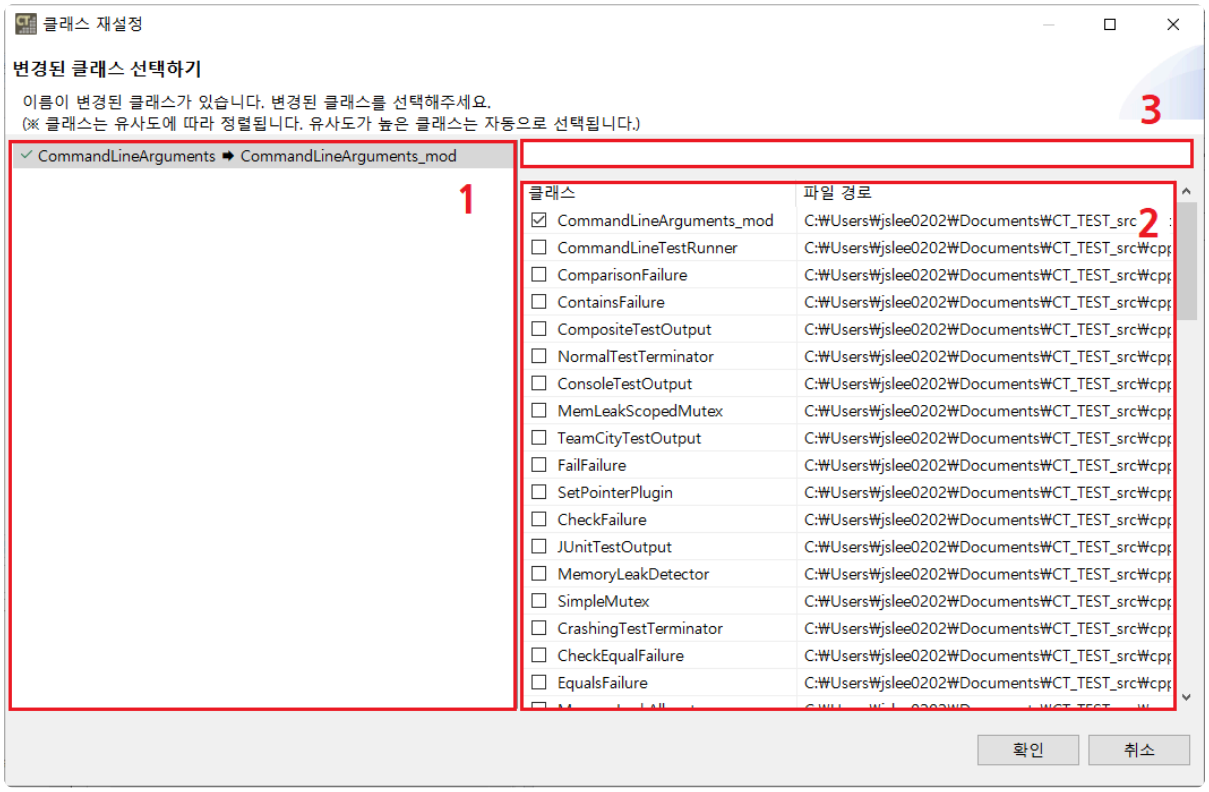
1.2. 자동으로 변경을 감지하는 경우

소스 코드를 변경하고 재분석을 하거나 테스트를 실행하면 무결성 검사를 통해 소스 코드의 변경을 감지합니다. CT 2023.12이 감지하는 소스 코드의 변경 유형은 다음과 같습니다.

- 테스트 혹은 클래스 코드의 대상 클래스 이름이 변경된 경우
- 테스트 혹은 스텁의 대상 함수 이름이 변경된 경우
- 테스트에 사용하는 전역 변수의 이름 또는 타입이 변경된 경우
- 클래스 코드의 대상 함수 이름이 변경된 경우
- 테스트 대상 함수의 리턴 타입 또는 파라미터의 이름이나 개수가 변경된 경우
- 결함 주입 대상 함수의 코드가 변경된 경우

테스트 혹은 클래스 코드의 대상 클래스 이름이 변경된 경우

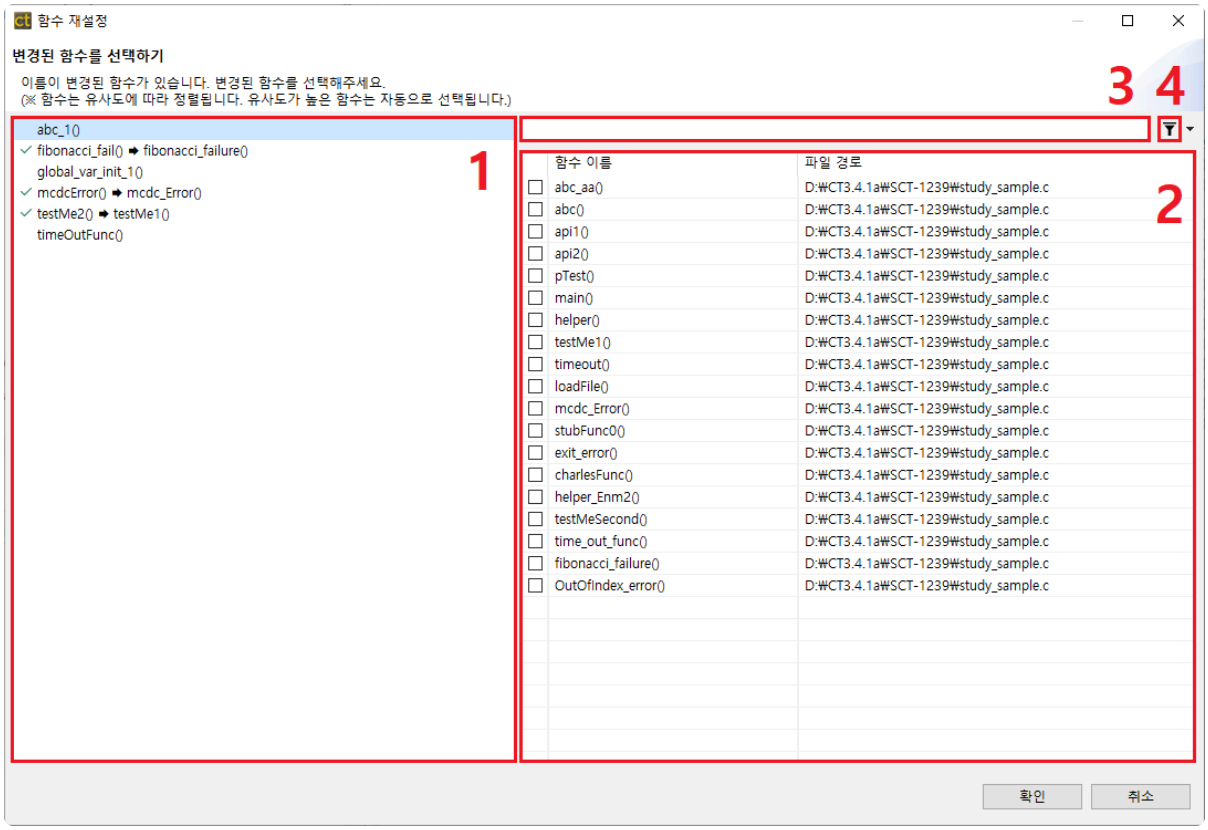
테스트 혹은 클래스 코드의 대상 클래스 이름이 변경된 경우, [클래스 재설정] 대화 상자가 뜹니다.



1. 왼쪽 영역은 변경이 감지된 클래스의 목록입니다. 설정이 완료된 함수는 ✓ 표시됩니다.
2. 오른쪽 영역은 현재 소스 코드의 클래스 목록입니다
    - 클래스 이름의 유사도에 따라 정렬됩니다.
    - 유사도가 높은 클래스는 자동으로 연결됩니다.
3. 클래스의 이름을 검색할 수 있습니다. (*: 임의의 문자열, ?: 임의의 문자)

테스트 혹은 스텁의 대상 함수 이름이 변경된 경우

테스트 혹은 스텁의 대상 함수 이름이 변경된 경우, [함수 재설정] 대화 상자가 뜹니다.

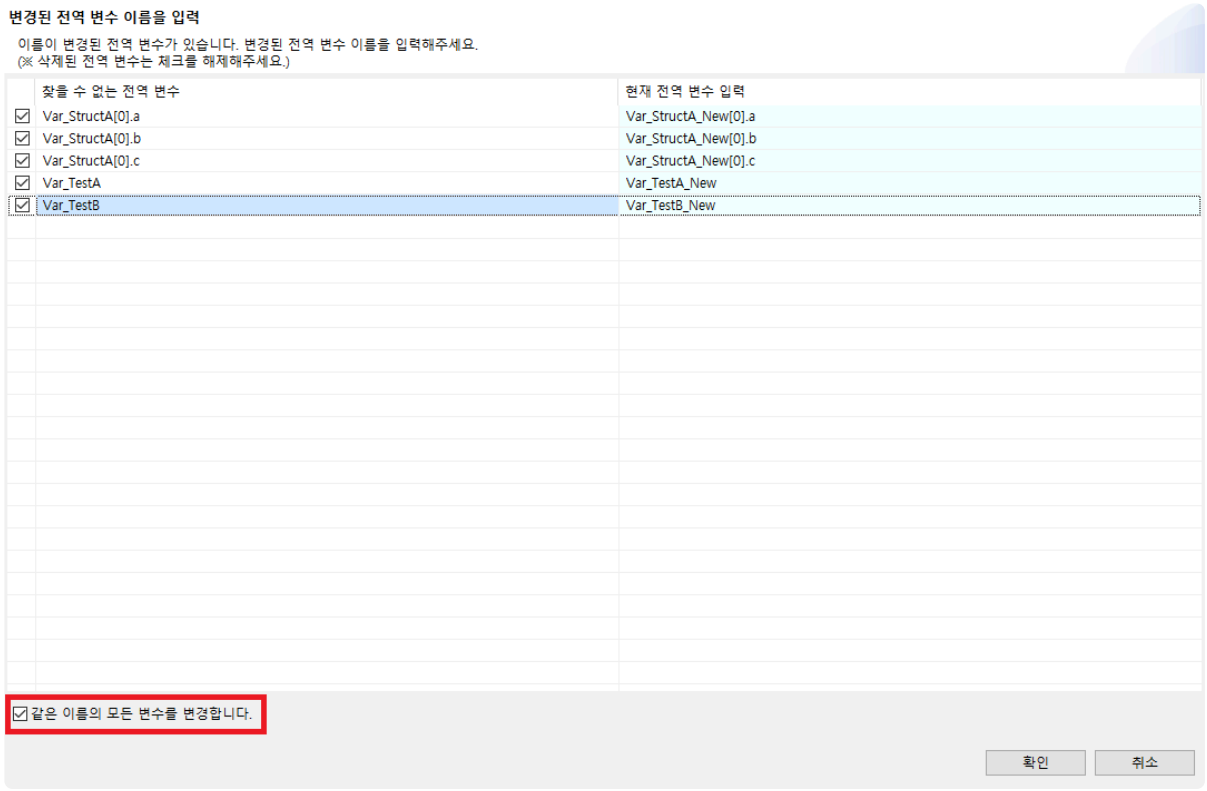


1. 왼쪽 영역은 변경이 감지된 함수의 목록입니다. 설정이 완료된 함수는 ✓ 표시됩니다.
2. 오른쪽 영역은 현재 소스 코드의 함수 목록입니다
    - 함수 이름의 유사도에 따라 정렬됩니다.
    - 유사도가 높은 함수는 자동으로 연결됩니다.
3. 함수의 이름을 검색할 수 있습니다. (*: 임의의 문자열, ?: 임의의 문자)
4. 테스트가 있는 함수를 숨기거나 보이게 합니다.

테스트에 사용하는 전역 변수의 이름 또는 타입이 변경된 경우

테스트에서 사용하는 전역 변수의 이름 또는 타입이 변경된 경우 [전역 변수 재설정] 대화 상자가 뜹니다.

1. 왼쪽 영역에는 찾을 수 없는 전역 변수 목록이 뜹니다.
    - 삭제된 전역 변수의 경우에는 체크박스를 체크 해제하시면 됩니다.
2. 오른쪽 영역에는 현재 전역 변수를 입력하는 텍스트 박스가 있습니다.
    - 사용자가 변수 이름을 수정하면 유사도가 높은 순서대로 전역 변수 목록이 나타납니다.
3. 유효한 전역 변수를 입력하면 텍스트 박스에 빨간색 표시가 사라집니다.



- 구조체의 멤버 변수들과 같이 유사한 이름을 갖는 전역 변수가 다수 변경된 경우, [같은 함수의 모든 변수를 변경합니다.] 기능을 사용하여 전역 변수 이름을 동시에 수정할 수 있습니다. [같은 이름의 모든 변수를 변경합니다.] 체크박스를 체크하고, 전역 변수의 이름을 수정하면 유사한 이름을 갖는 전역 변수의 이름이 동시에 수정됩니다. 체크박스를 체크 해제하면 전역 변수의 이름을 각각 수정할 수 있습니다.

클래스 코드의 대상 함수 이름이 변경된 경우

클래스 코드에 포함된 함수 이름이 변경된 경우, [클래스 코드 재설정] 대화 상자가 뜹니다.

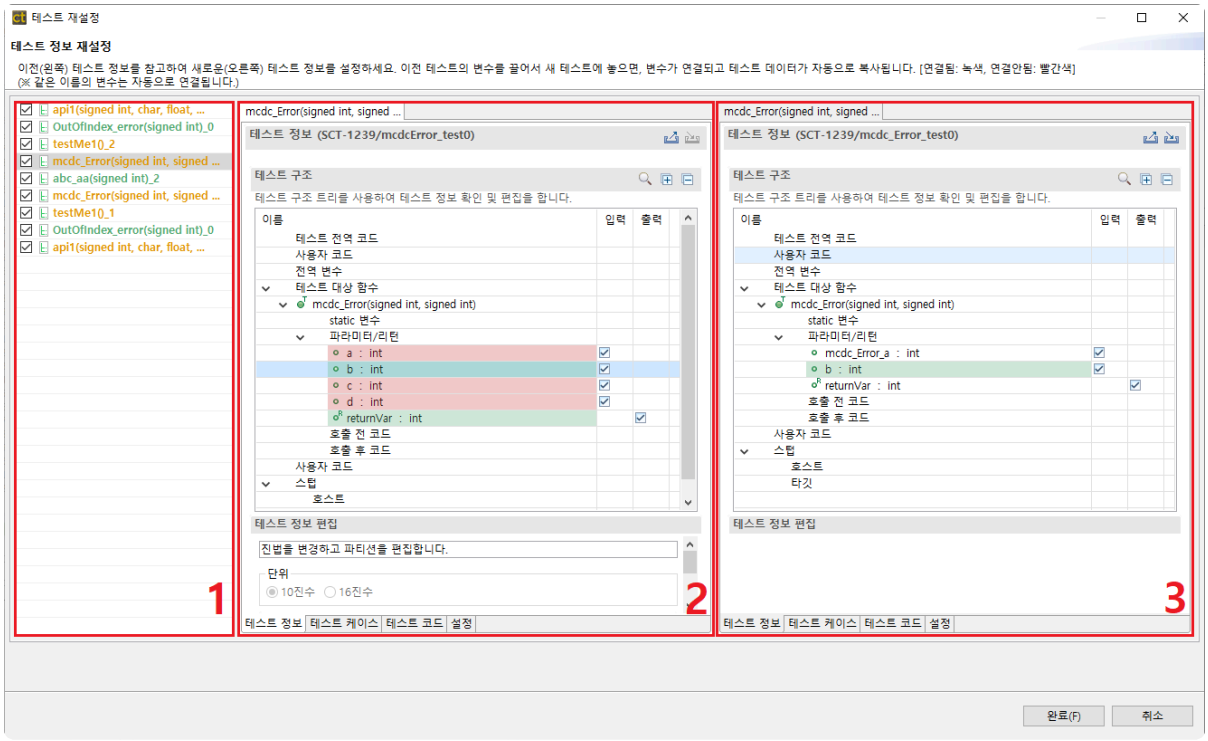
1. 왼쪽 영역에는 재설정 대상이 되는 클래스 코드가 트리 형태로 뜹니다.
    - 클래스와 클래스 코드가 부모 자식 트리 형태로 구성됩니다.

- 클래스 옆 숫자는 (사용자가 확인한 클래스 코드 개수 / 재설정 대상 클래스 코드 개수) 입니다.
- 클래스 코드 옆 숫자는 해당 클래스 코드에서 변경된 부분 및 삭제된 부분의 개수입니다.

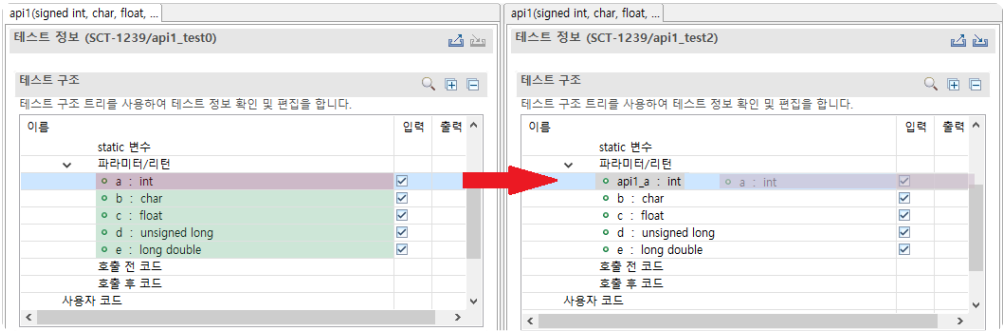
2. 오른쪽 영역에서 재설정 과정을 거친 이전 클래스 코드(회색 배경/좌)와, 이전 클래스 코드에 대비하여 변경된 새 클래스 코드(흰색 배경/우)가 뜹니다.
    - 이전 클래스 코드는 수정이 불가하며 새 클래스 코드는 수정할 수 있습니다.
    - 빨간색 배경은 삭제됐거나 직접 추가한 함수를 나타냅니다.
    - 노란색 배경은 재설정을 통해 변경된 함수를 나타냅니다.
    - 초록색 배경은 새로 추가된 함수를 나타냅니다.

테스트 대상 함수의 리턴 타입 또는 파라미터의 이름이나 개수가 변경된 경우

테스트 대상 함수의 리턴 타입 또는 파라미터의 이름이나 개수가 변경된 경우, [테스트 재설정] 대화 상자가 뜹니다.

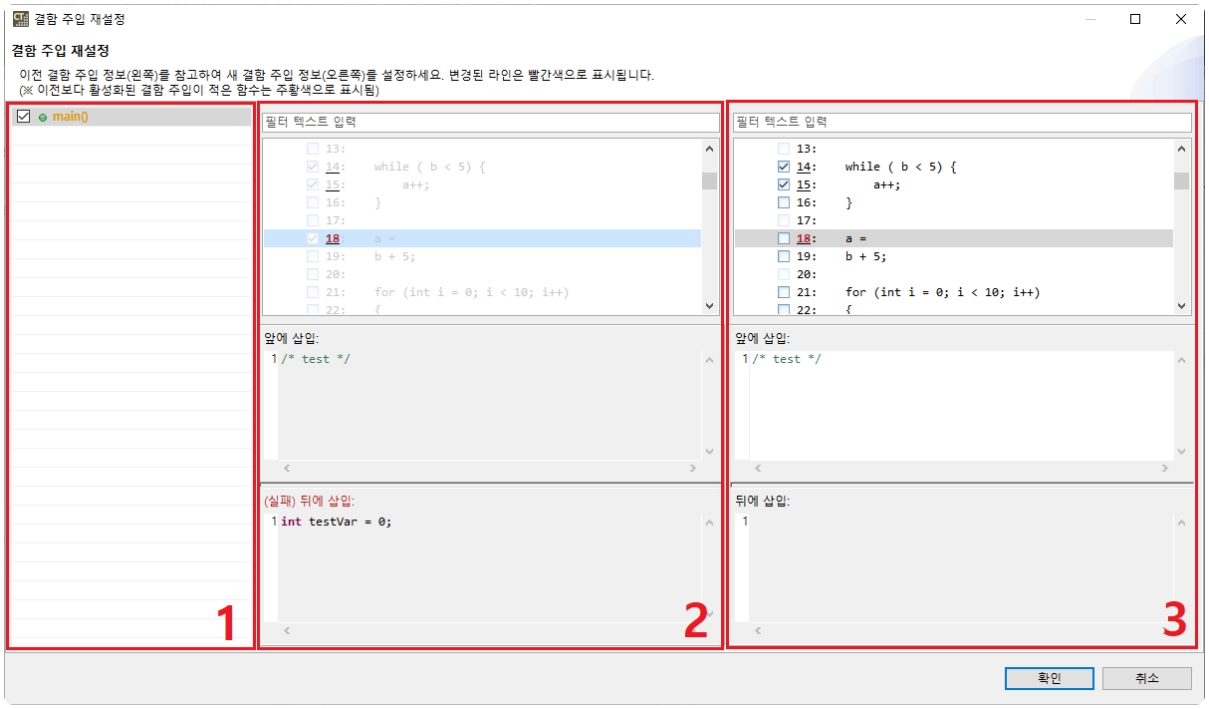


1. 변경된 함수의 목록
    - 체크박스를 체크 해제하면 변경 사항을 저장하지 않고 변경 전 함수의 테스트 정보를 유지합니다.
2. 변경 전 함수의 테스트 정보
    - 변경 후 함수의 테스트 정보와 연결이 된 변수는 녹색으로 표시되고 연결되지 않은 경우 빨간색으로 표시됩니다.
3. 변경 후 함수의 테스트 정보
    - 변경 전 함수의 변수를 선택하면 해당 변수와 연결된 변수를 표시합니다.
    - 변경 전 함수의 변수를 끌어서 변경 후 함수의 변수에 놓으면 테스트 데이터를 복사합니다.



결함 주입 대상 함수의 코드가 변경된 경우

결함 주입 대상 함수의 코드가 변경된 경우, [결함 주입 재설정] 대화 상자가 뜹니다.



결함 주입 함수의 목록은 1번 영역, 변경 전 결함 주입 정보는 2번 영역, 변경 후 결함 주입 정보는 3번 영역에 표시됩니다.

- 결함 주입 함수 목록
  - 체크박스를 체크 해제하면 변경사항을 저장하지 않고 이전 결함 주입 정보를 유지합니다.
- 결함 주입 정보 창
  - 변경 전 결함 주입 정보는 복사만 가능합니다. 단축키(`Ctrl` + `C`) 또는 우클릭으로 복사할 수 있습니다.
  - 변경 후 결함 주입 정보는 수정이 가능합니다. 단축키(`Ctrl` + `C` / `V`) 또는 우클릭으로 복사/붙여넣기할 수 있습니다.
  - 변경된 라인은 라인 번호가 빨간색으로 표시됩니다.
  - 라인을 더블 클릭하면 대응하는 결함 주입 정보 창에서 선택한 라인과 동일한 라인이 선택됩니다.
  - 선택된 라인에 작성된 코드는 하단의 결함 주입 코드 창에서 확인할 수 있습니다.
- 결함 주입 코드 창
  - 선택된 라인의 앞, 뒤에 작성된 코드가 표시됩니다.
  - 결함 주입을 할 수 없는 위치는 코드를 작성할 수 없도록 비활성화됩니다.

프로젝트에 생성된 테스트가 없거나, 결함 주입 라인을 활성화하였으나 작성된 코드가 없는 경우에는 [결함 주입 재설정] 대화 상자가 뜨지 않습니다.

1.3. 자동으로 변경을 감지하지 않는 경우

CT 2023.12에서 무결성 검사로 파악할 수 없는 변경은 다음과 같습니다.

- `typedef`로 정의되지 않은 전역 변수 타입을 변경하는 경우
- 테스트 빌드 오류 (묵시적 캐스팅이 불가능한 경우)
- 테스트 실행 오류 (메모리 범위 초과 등 런타임 오류)
- 전역 변수를 제외한 심볼을 변경하는 경우
    - 파라미터의 하위 타입, 사용자가 매크로로 추가한 심볼, static 변수 등을 변경한 경우
- 위치 변경으로 인한 사이드이펙트
    - 대상 테스트에서 전역 변수 접근 불가로 오류가 발생하는 경우
- 빌드 스텝의 변경

전역 변수 타입을 변경하는 경우, 전역 변수를 제외한 심볼을 변경하는 경우, 위치 변경으로 인한 사이드이펙트는 사용자가 [테스트 재설정] 기능으로 테스트를 수정해야 합니다. 빌드 스텝의 대상 함수가 변경되는 경우는 무결성 검사의 대상이 아니므로 사용자가 빌드 스텝을 삭제해야 합니다.

2. 협업 가이드

CT 2023.12를 사용하여 여러 사람이 협업할 때 작업 내용과 결과를 공유하는 방법은 다음과 같습니다.

- 팀 테스팅 사용 가이드
- 다른 사용자와 프로젝트 공유하기
- 커버리지 가져오기 가이드

2.1. 팀 테스팅 사용 가이드

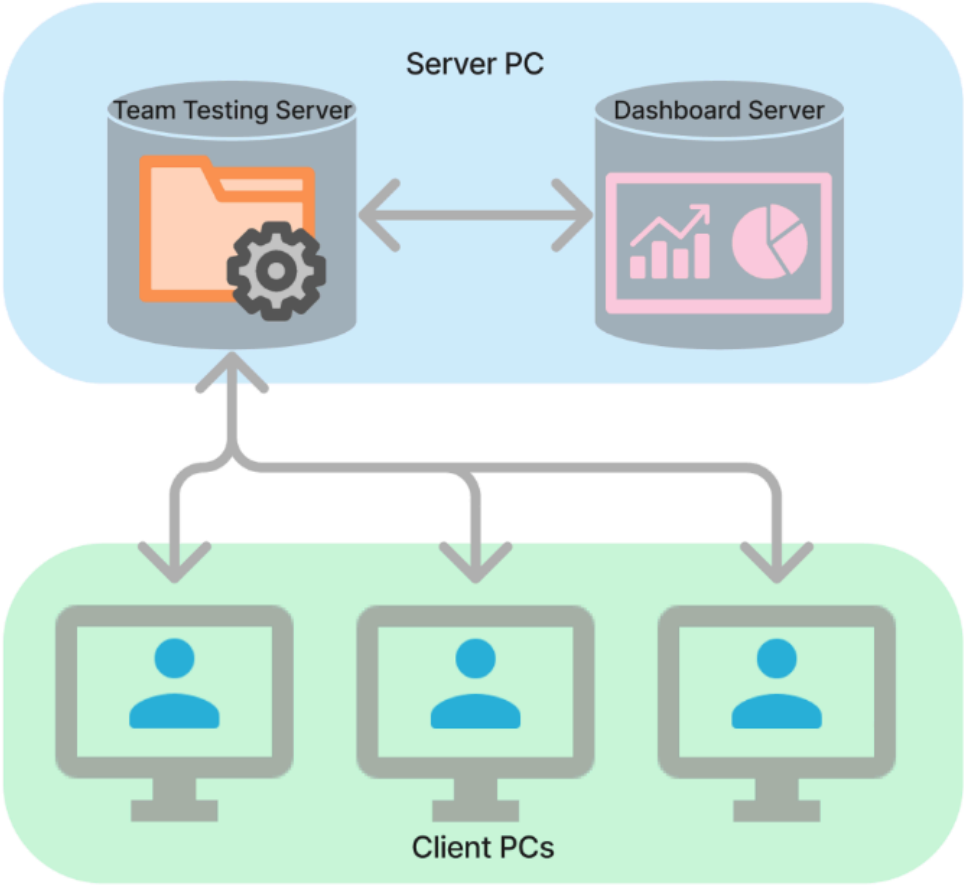
팀 테스팅이란?

시간이 지남에 따라 소프트웨어 규모가 점점 커지고 있습니다. 소프트웨어 규모가 증가하여 테스트 대상의 규모도 역시 증가합니다. 그 반면, 업데이트 주기는 점점 짧아지고 테스트할 수 있는 시간이 줄어 짧은 시간에 더 많은 테스트를 해야 합니다.

팀 테스팅은 하나의 소프트웨어를 여러 사람이 나누어 테스트하는 방식입니다. 팀 테스팅에는 여러 문제점이 있습니다. 첫 번째, 테스트 환경과 테스트 케이스를 다른 사용자와 공유하기 어렵습니다. 두 번째, 사용자가 직접 작성하는 스텁, 클래스 코드 등의 테스트 자원을 중복으로 생성하게 됩니다. 세 번째, 테스트 결과를 병합하기가 어렵습니다. 이 외에도 최종 결과를 확인하기 위해 한 PC에 취합하여 전체 테스트를 재실행해야 하는 등 다양한 어려움이 있습니다.

CT 2023.12는 이런 변화에 발맞추어 더 짧은 시간에 더 많은 테스트를 할 수 있도록 팀 테스팅 기능을 제공합니다. 팀 테스팅 서버는 CT 2023.12가 팀 테스팅 기능을 제공하기 위해 만든 서버입니다. 프로젝트를 진행하는 동안 팀 테스팅 서버는 사용자 간에 프로젝트 구성을 동기화해 주고 스텁과 클래스 코드를 모든 사용자가 공유할 수 있게 합니다. 또한, 테스트를 실행할 때마다 결과를 취합하여 대시보드를 통해 보여줍니다. 사용자는 대시보드를 이용하여 프로젝트 진행 상황을 한눈에 파악할 수 있습니다.

즉, CT 2023.12는 각각의 사용자가 테스트를 진행하는 클라이언트 역할을 합니다. 팀 테스팅 서버는 사용자가 공유할 프로젝트를 저장하고 관리합니다. 대시보드는 팀 테스팅 서버에 저장된 프로젝트 정보와 진행 상황을 볼 수 있는 웹 페이지입니다.



용어

아래는 CT 2023.12의 팀 테스팅에서 사용하는 용어입니다.

- 팀 프로젝트: 팀 테스팅 서버로 내보내어 다른 사용자와 같이 사용하는 프로젝트
- 공유 자원
  - 프로젝트 특성, 환경 설정, 소스 코드, 툴체인 등 프로젝트를 구성하는 설정과 스텝, 클래스 코드 등 테스트와 연결된 자원
  - 사용자 간에 자동으로 공유됨
- 테스트 자원
  - 테스트, 테스트 데이터 등 테스트에 필요한 자원
  - 사용자 간에 자동으로 공유되지 않으며, 공유가 필요한 경우 팀 테스팅 서버에서 가져와서 사용해야 함
- 로컬: 사용자가 작업하는 CT(클라이언트)
- 팀 테스팅 서버: 모든 사용자의 작업 내용이 취합되는 곳
- 커밋: 로컬의 변경 사항을 팀 테스팅 서버에 반영하는 작업
- 업데이트: 다른 사용자의 수정 사항을 로컬로 내려받는 작업
- 충돌: 다른 사용자가 수정하여 커밋한 자원을 업데이트할 때, 내 로컬에서도 수정이 있으면 충돌이라고 함
- 리비전
  - 팀 테스팅 서버의 수정 기록
  - 사용자가 커밋을 하여 팀 프로젝트의 형상이 바뀌면 리비전이 증가함
  - 프로젝트 생성 시 리비전은 1

    - 커밋할 때 1씩 증가

팀 프로젝트 과정

팀 프로젝트는 크게 3단계로 나눌 수 있습니다.

1. 프로젝트 초기 설정
    - 팀 프로젝트를 생성하고, 테스트 환경에 맞게 프로젝트 구성을 변경하는 단계입니다.
    - 사용자가 각각의 PC로 팀 프로젝트를 가져와서 하나의 프로젝트를 공유하여 사용합니다.

2. 테스트 진행
    - 커밋과 업데이트하며 테스트를 진행합니다.
    - 공유 자원의 변경이 생기는 경우 알림을 받아 업데이트할 수 있습니다.
    - 소스 코드의 변경으로 테스트를 사용할 수 없는 경우, 재분석 후 테스트 재설정을 진행합니다.
    - 업데이트 과정에서 충돌이 발생한 경우, 충돌을 해결한 후 테스트를 진행하여야 합니다.

3. 테스트 결과 취합
    - 테스트 결과는 팀 테스팅 서버에서 취합되어 대시보드에서 실시간으로 확인할 수 있습니다.
    - 전체 테스트에 대한 실행이나 보고서 출력이 필요한 경우, 한 PC에 모든 테스트를 가져와서 보고서를 출력할 수 있습니다.

그 외에도 팀 테스팅 서버와의 연결이 불안정한 경우를 대비하여 온/오프라인 모드 기능도 제공합니다.

2.1.1. 프로젝트 초기 설정

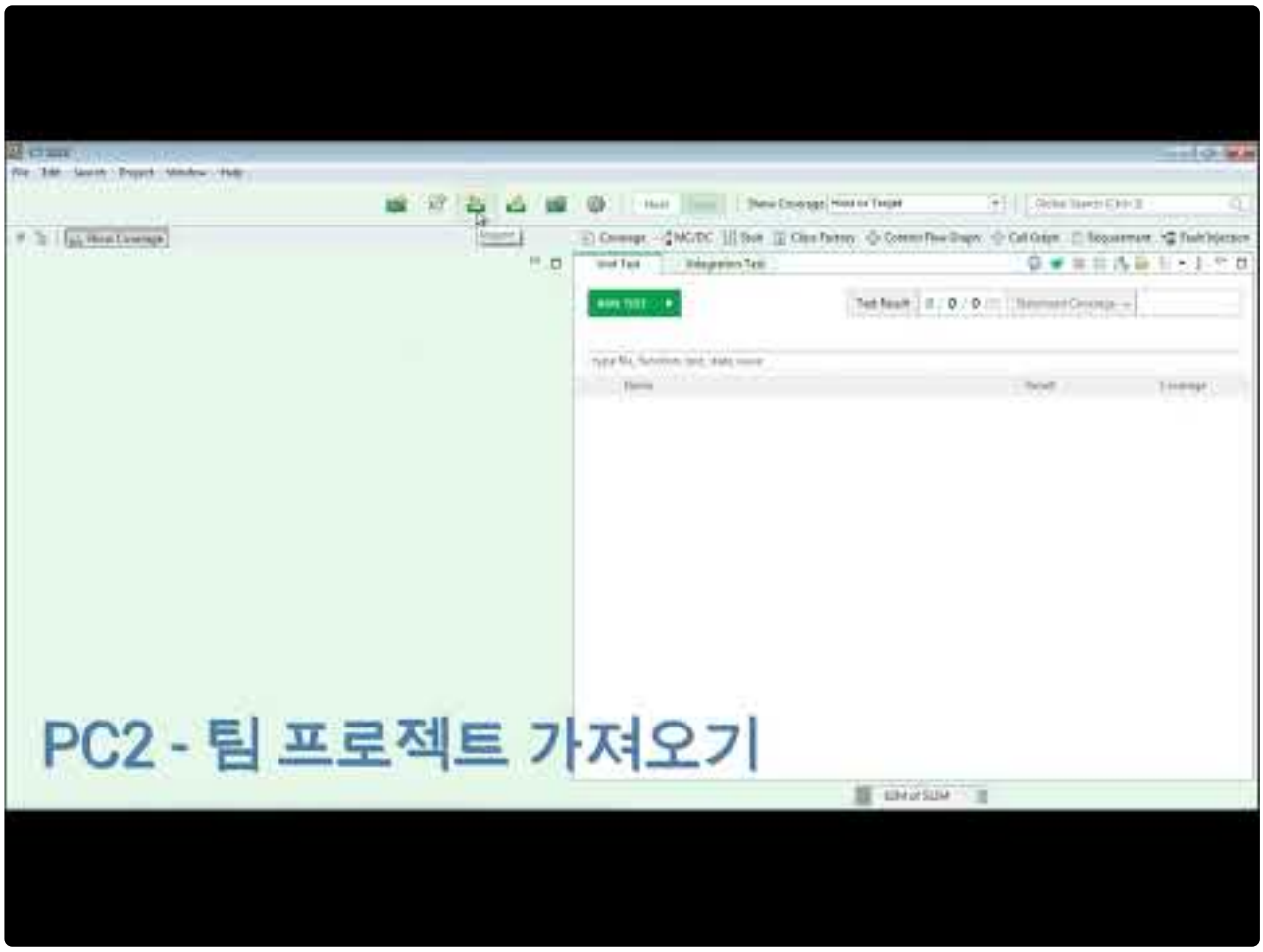
팀 프로젝트를 사용하면 다른 사용자와 프로젝트 환경을 공유할 수 있습니다.
팀 프로젝트를 만드는 방법은 두 가지가 있습니다. 첫 번째는 새로운 팀 프로젝트를 만드는 것입니다. 두 번째는 기존에 사용하던 프로젝트를 팀 프로젝트로 변환하여 사용하는 것입니다. 하위 버전에서 사용하던 프로젝트를 CT 2023.12로 마이그레이션하여 팀 프로젝트로 변환할 수 있습니다.

팀 프로젝트 생성 및 공유

1. 한 PC에서 팀 테스팅 서버와 연결하여 팀 프로젝트를 생성합니다. 이 과정은 매뉴얼의 [팀 프로젝트 생성하기] 페이지를 참고하세요.

2. 테스트 환경에 맞게 팀 프로젝트를 설정합니다. 툴체인 설정을 완료하고, 테스트 진행에 필요한 설정도 완료합니다. 그 후, 변경 사항을 팀 테스팅 서버에 커밋합니다.

3. 다른 사람의 PC에서 해당 팀 프로젝트를 가져와서 사용합니다.



https://www.youtube.com/embed/Enyc5AFeTho?rel=0

팀 프로젝트 변환하기

1. 변환할 프로젝트를 우클릭 하여 팀 프로젝트로 변환합니다.

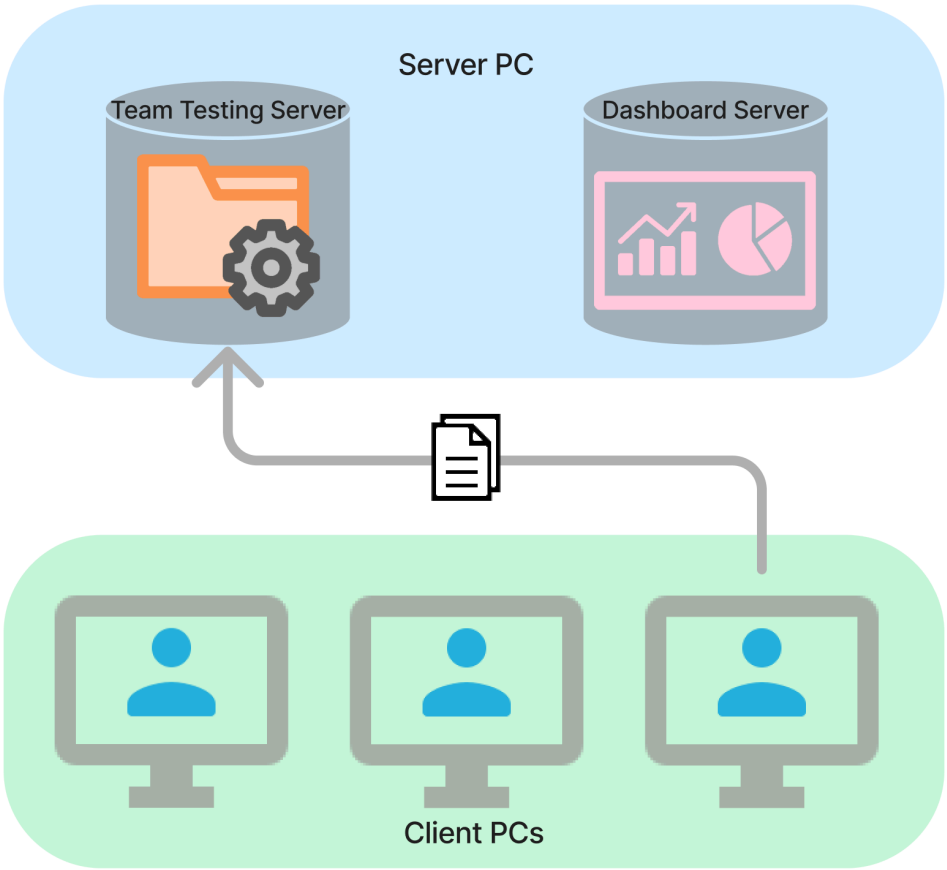
2. 해당 프로젝트의 설정이 완료되지 않은 상태라면 프로젝트 설정을 완료한 후, 커밋을 합니다.

3. 다른 사람의 PC에서 해당 팀 프로젝트를 가져와서 사용합니다.

2.1.2. 커밋 (Commit) 및 업데이트 (Update)

팀 프로젝트를 여러 사용자가 공유한 후에 테스트를 진행하게 됩니다. 테스트 과정에서 로컬 PC의 변경 사항을 팀 테스팅 서버로 내보내거나 팀 테스팅 서버의 변경 사항을 로컬 PC로 가져와야 합니다. 로컬 PC의 변경 사항을 팀 테스팅 서버로 내보내는 것을 커밋(Commit), 팀 테스팅 서버의 변경 사항을 로컬로 가져오는 것을 업데이트(Update)라고 합니다.

커밋

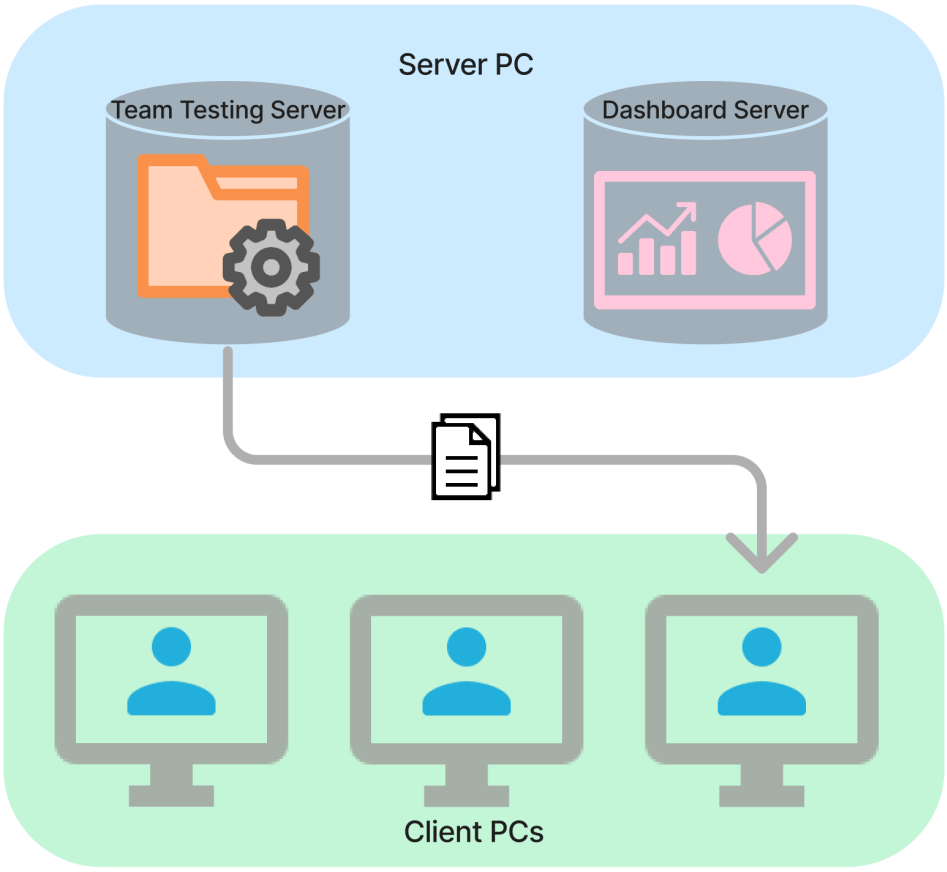


로컬 PC의 작업 내용을 팀 테스팅 서버에 반영하는 것을 커밋이라 합니다. 팀 테스팅 서버에 반영된 작업 내용은 다른 사용자들과 공유됩니다. 팀 테스팅 서버와 로컬의 리비전이 다른 상태로 커밋하면 다른 사용자의 변경 사항을 되돌리거나 덮어쓸 수 있습니다. 커밋할 때는 팀 테스팅 서버와 로컬의 리비전이 같아야 합니다. 만약 로컬의 리비전이 팀 테스팅 서버보다 낮다면 업데이트한 후에 커밋하여야 합니다.

사용자는 두 가지 방법으로 커밋할 수 있습니다.

1. 테스트 실행 후, 변경 사항과 테스트 결과를 함께 커밋(자동 커밋)
2. 사용자가 직접 커밋(수동 커밋)
    - 사용자가 직접 커밋하는 경우, 커밋할 내용을 커밋 대화상자에서 확인할 수 있습니다.

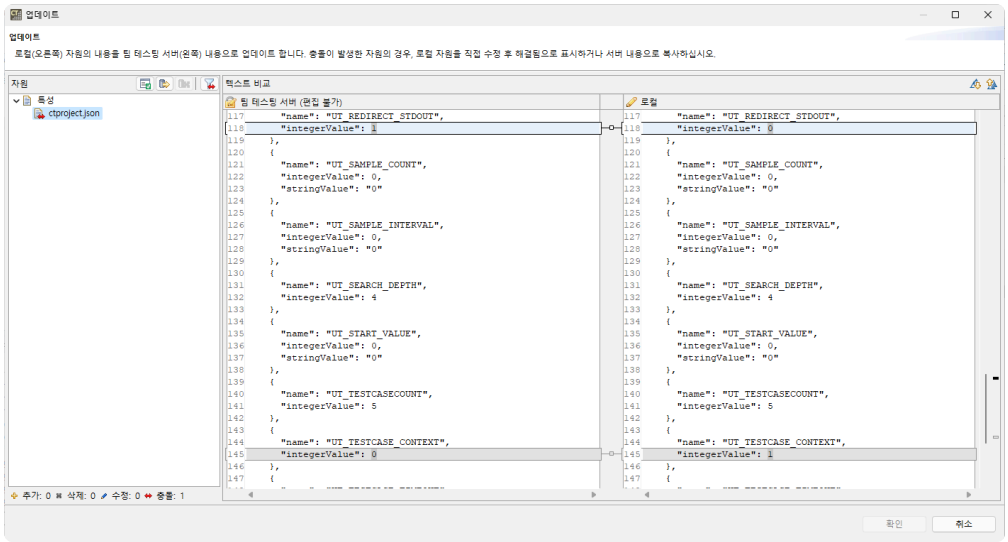
업데이트



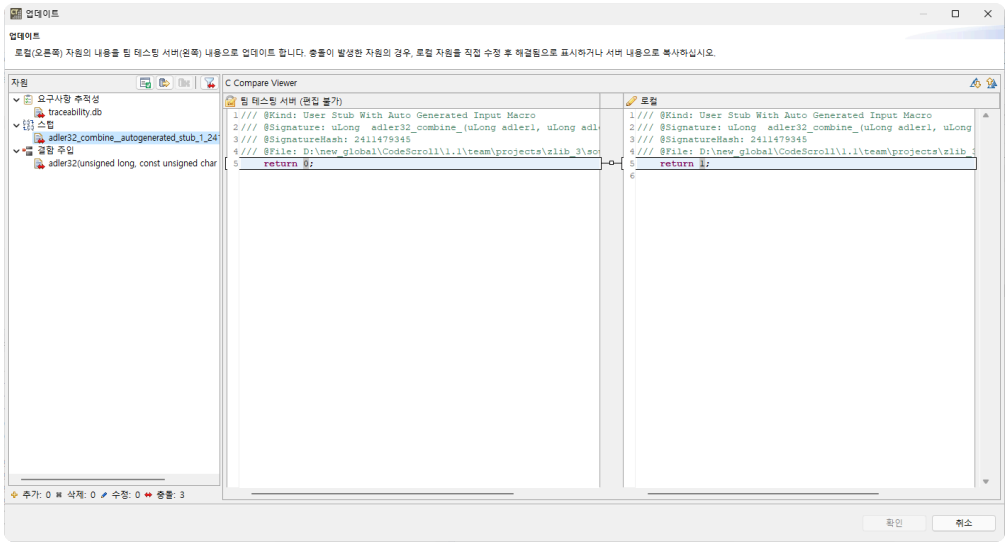
팀 테스팅 서버의 작업 내용을 로컬 PC로 가져오는 것을 업데이트라 합니다. 업데이트할 때, 팀 테스팅 서버에서 내려받을 자원이 로컬에서 이미 수정한 자원인 경우를 충돌이라고 합니다. 충돌이 발생하면 업데이트 대화상자에서 충돌을 해결하고 업데이트를 진행합니다.

업데이트는 다음과 같이 진행됩니다.

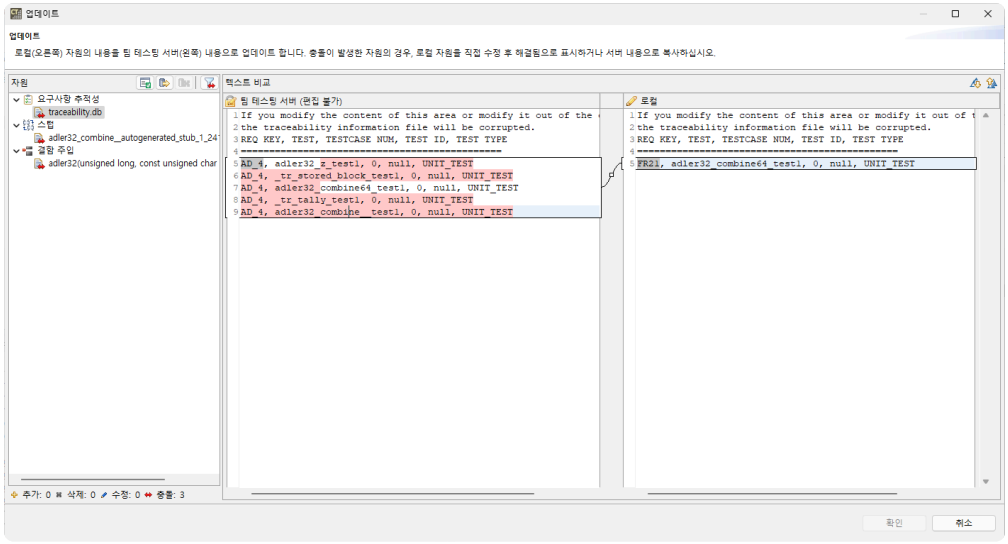
1. 팀 테스팅 서버와 로컬의 리비전을 확인하고 업데이트합니다.
2. 업데이트 대화상자에서 업데이트할 자원을 확인합니다. 자원의 종류에 따라 다른 형식의 비교 뷰어를 사용하여 업데이트 대화상자가 여러 번 뜰 수 있습니다.
    - 프로젝트 특성, 툴체인 등 JSON이나 XML 형식으로 관리하는 자원은 텍스트 비교로 업데이트 내용을 확인할 수 있습니다.



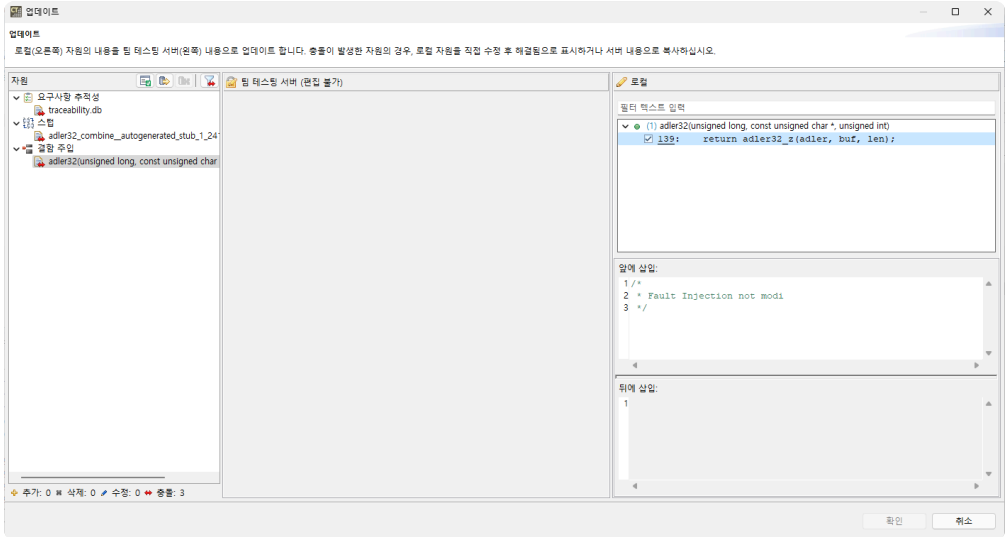
- 스텁, 클래스 코드 등 코드 형태의 자원은 소스 코드 비교로 업데이트 내용을 확인합니다.



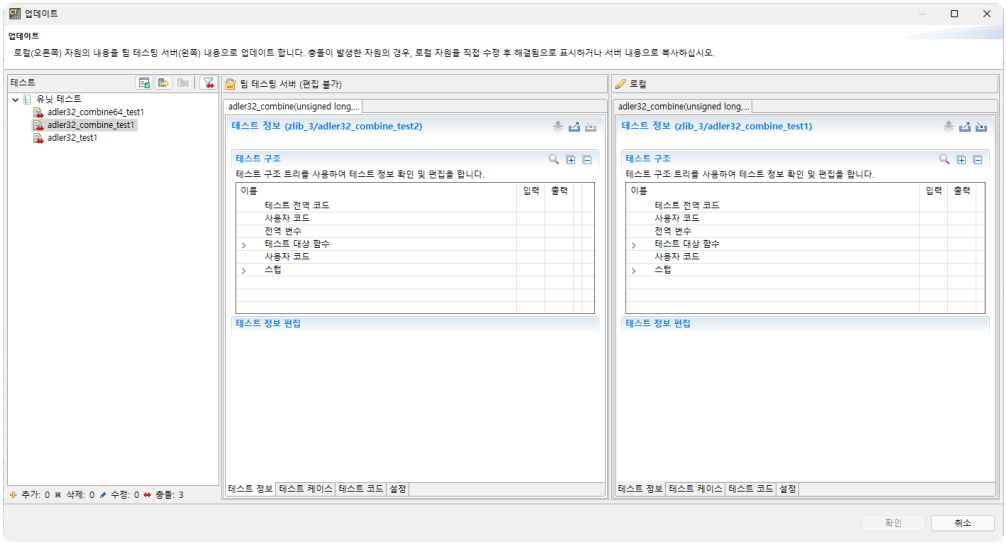
- 요구사항 추적성 등 DB로 관리하는 자원은 DB의 데이터를 텍스트 형식으로 비교하여 업데이트 내용을 확인합니다.



- 결함 주입의 경우, 결함 주입 뷰와 유사한 형태로 업데이트 내용을 확인합니다.



- 테스트의 경우, 테스트 편집기의 형태로 업데이트 내용을 확인합니다.



3. 충돌이 발생하면 두 가지 방법으로 충돌을 해결할 수 있습니다.
    - 서버의 형상으로 저장하기
        - (서버에서 로컬로 복사) 아이콘을 선택하여 팀 테스팅 서버의 형상을 로컬에 복사합니다.. 팀 테스팅 서버에서 로컬로 복사하는 경우, 로컬에서의 변경 사항을 팀 테스팅 서버의 형상으로 덮어쓰며 로컬 형상을 수정할 수 없습니다. (복사 취소) 아이콘을 통해 복사한 형상을 원래대로 되돌릴 수 있습니다.
    - 사용자가 직접 수정하여 저장하기
        - 팀 테스팅 서버의 형상을 복사하지 않고 사용자가 직접 로컬의 형상을 수정합니다. 현재 로컬의 형상과 팀 테스팅 서버의 형상을 참고하여 사용자가 작성한 형상으로 저장합니다. 사용자가 직접 수정하면 (해결됨으로 표시) 아이콘을 선택하여 해당 자원의 충돌이 해결됨을 표시하여야 합니다.
4. 모든 충돌이 해결되면 [확인] 버튼이 활성화되며 [확인]을 클릭하여 업데이트를 진행합니다.
5. 업데이트한 자원의 종류에 따라 업데이트 후 재분석이 필요합니다.

팀 테스팅에서는 로컬의 형상을 항상 최신으로 유지할 수 있도록 업데이트 알림 기능을 제공합니다. 업데이트 알림 외에도 커밋할 때 팀 테스팅 서버와 로컬의 리비전이 다르면 업데이트가 필요하다는 안내를 합니다.

공유 자원의 변경

스텁, 클래스 코드 등 테스트 결과에 영향을 줄 수 있는 공유 자원을 변경하면 해당 자원과 연결된 테스트의 결과가 바뀔 수 있습니다. 변경된 자원을 커밋하면 팀 테스팅 서버에 있는 테스트의 결과를 신뢰할 수 없게 되므로 해당 자원과 연결된 테스트의 결과가 팀 테스팅 서버에서 삭제됩니다. 이런 경우, 프로젝트의 커버리지가 감소할 수 있습니다.

```
void func() {
 if( returnNum() == 1 ) {
    /* ... */
 } else {
    /* ... */
    }
}
```

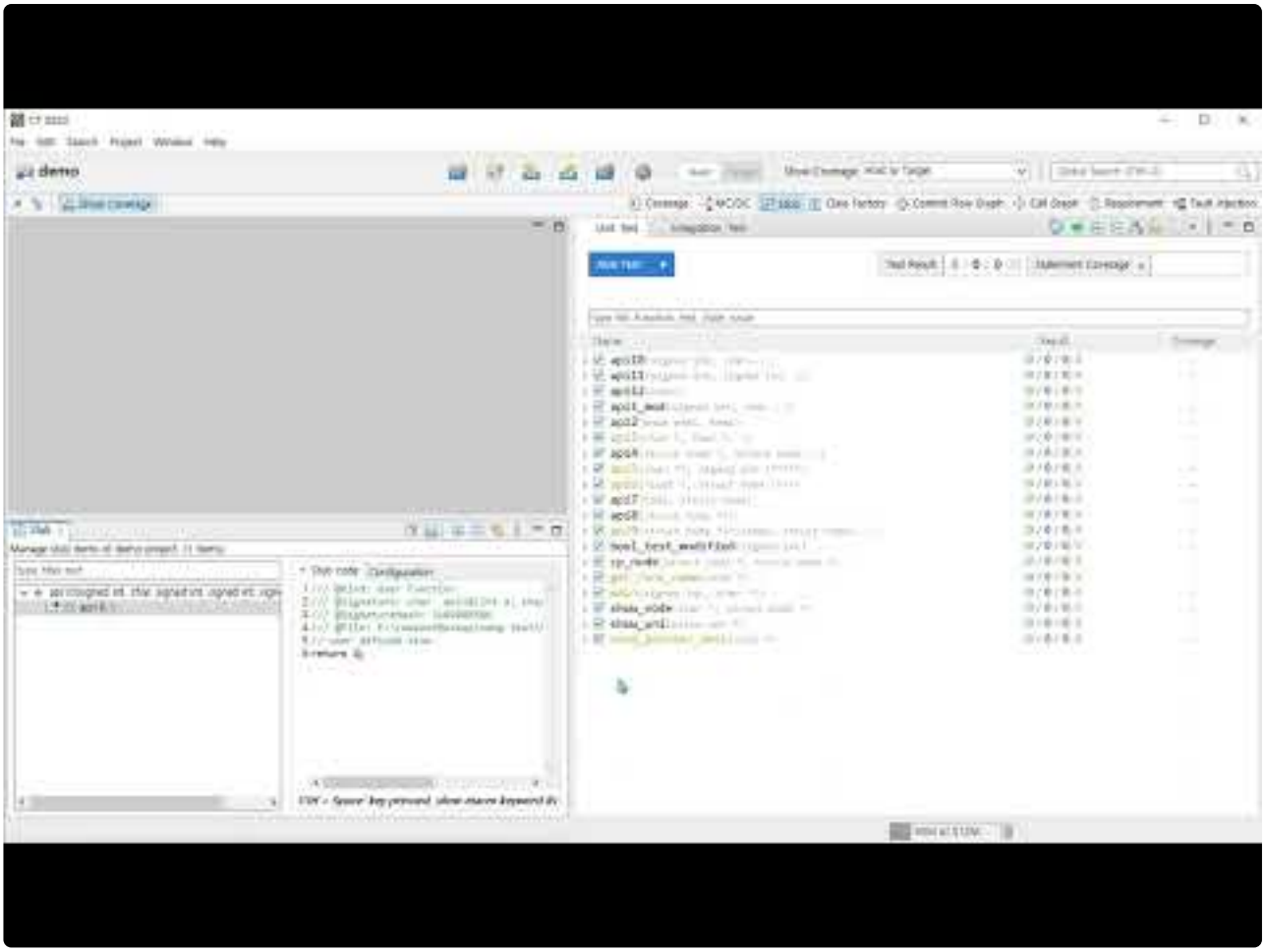
위 코드에서 `returnNum()`에 대한 스텁을 생성하여 `void func()`에 대한 테스트에 연결하면 `returnNu`

m() 스텁의 반환 값에 따라 커버리지가 달라질 수 있습니다. 테스트와 연결된 공유 자원을 수정할 때는 연결된 테스트를 재실행하여야 합니다. 수정한 자원을 커밋하면 해당 자원과 연결된 테스트의 실행 결과가 팀 테스팅 서버에서 삭제됩니다. 실행 결과가 삭제된 테스트는 '재실행 필요 테스트'로 관리됩니다. 업데이트할 때, 재실행 필요 테스트가 있으면 [유닛/통합 테스트] 뷰에서 해당 테스트에 ! 표시를 합니다. 재실행 필요 테스트를 다시 실행하면 새로운 결과가 커밋되고 !가 사라집니다.
공유 자원을 변경하고 해당 자원과 연결된 테스트를 실행하여 자동 커밋하면 실행된 테스트는 새로운 테스트 결과가 커밋되었기 때문에 재실행 필요 테스트로 관리하지 않습니다.

참고 영상

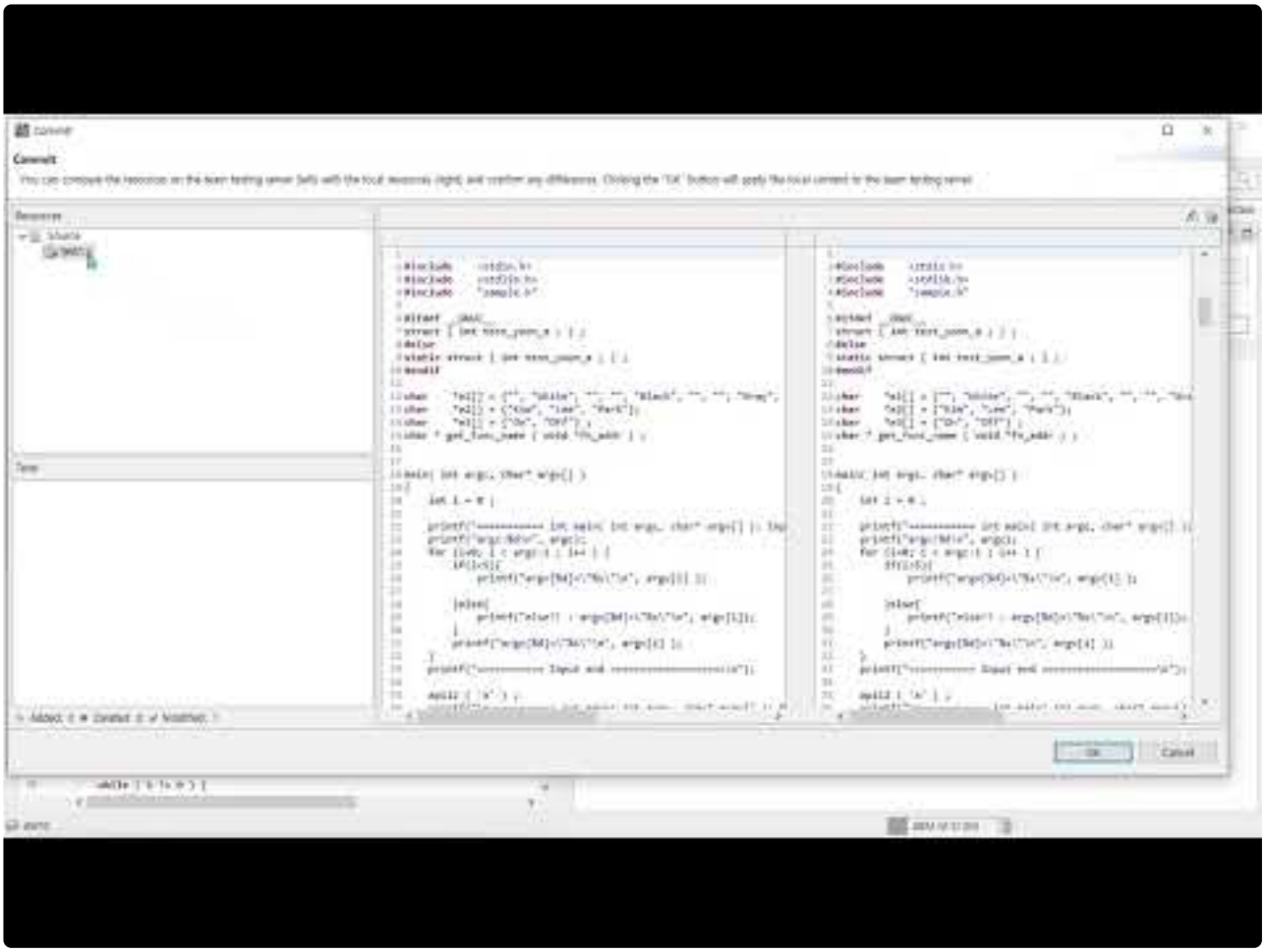
커밋/업데이트/충돌 관련하여 참고할 수 있는 영상입니다.

- 스텁을 생성한 후 테스트를 실행하여 자동으로 커밋하고 다른 PC에서 커밋한 스텁을 업데이트하는 영상입니다.



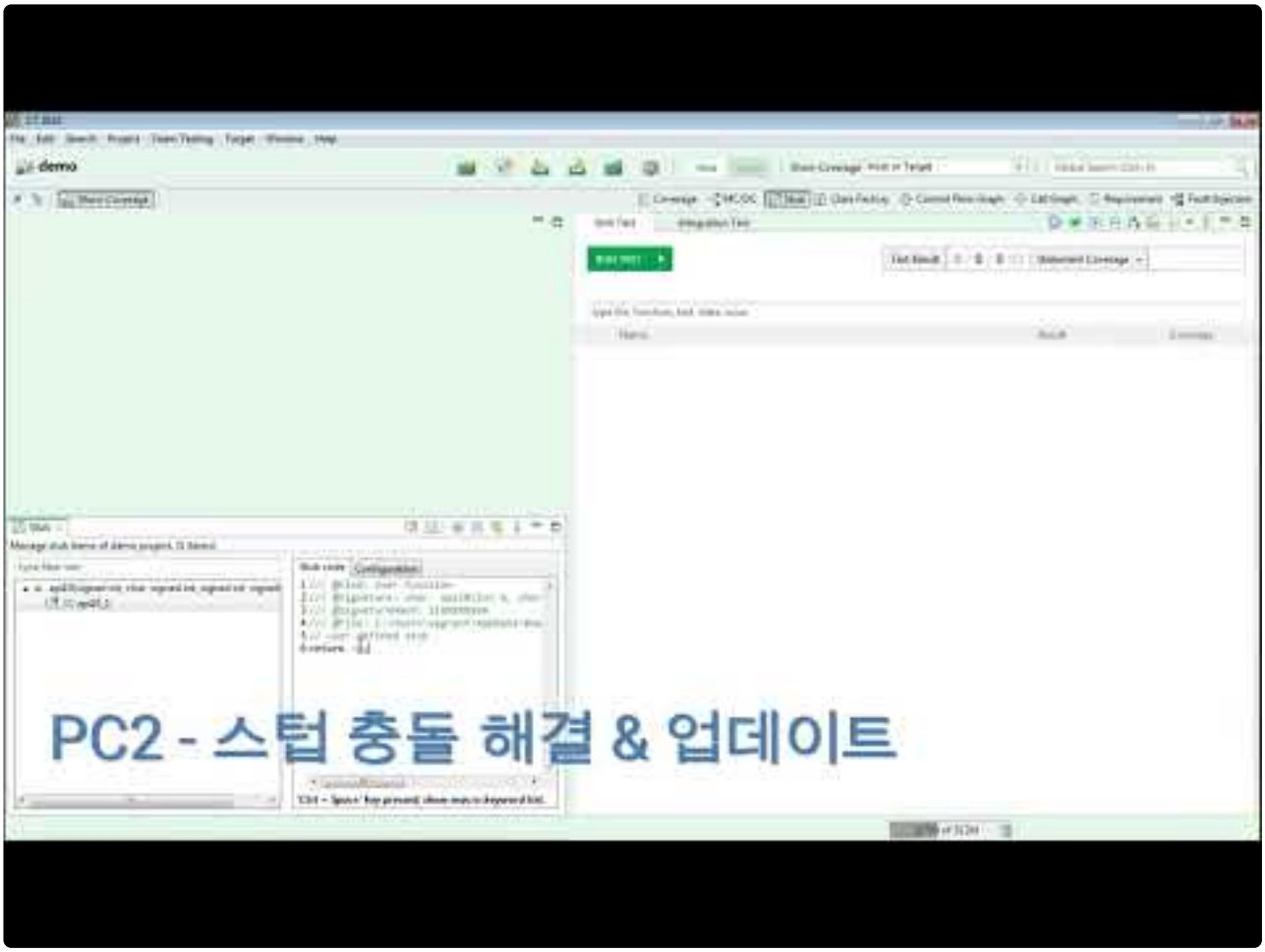
https://www.youtube.com/embed/77Dpg8CYoeo?rel=0

- 소스 코드를 변경한 후 수동으로 커밋하고, 다른 PC에서 업데이트 알림을 통해 업데이트하는 영상입니다.



https://www.youtube.com/embed/k6801k-71BQ?rel=0

- 사용자 스텁을 수정하여 커밋하고 업데이트하는 과정에서 발생한 충돌을 해결하는 영상입니다.



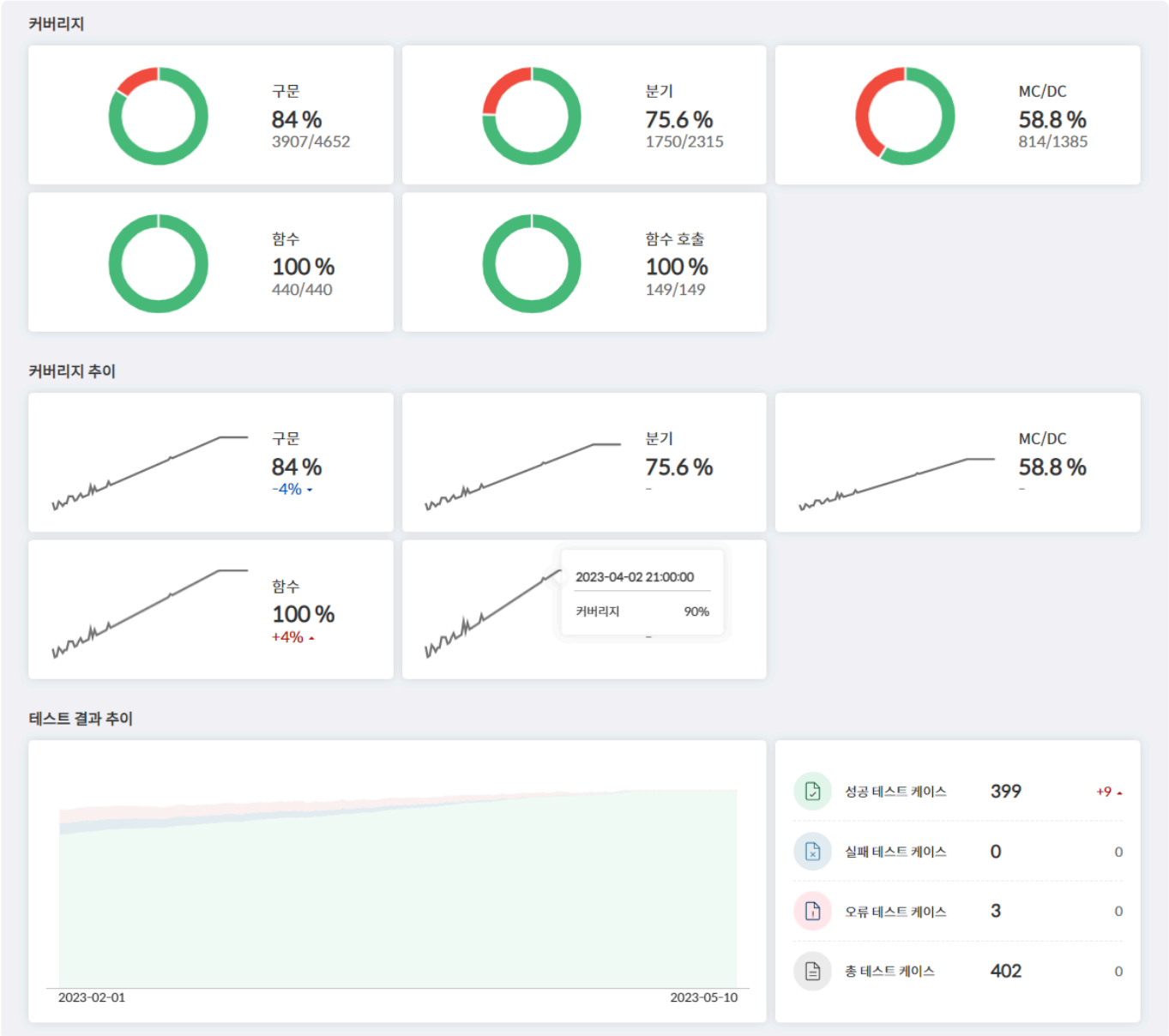
https://www.youtube.com/embed/67QPhrysG5k?rel=0

2.1.3. 테스트 결과 취합

팀 프로젝트에 대한 테스트가 모두 종료된 후에 결과를 취합하여야 합니다. 취합한 결과를 확인하는 방법은 두 가지입니다. 첫 번째는 대시보드에서 확인하는 방법입니다. 팀 프로젝트로 테스트를 진행하면 팀 테스팅 서버에서 모든 테스트의 결과를 취합하여 대시보드에서 보여줍니다. 두 번째는 보고서를 생성하여 확인하는 방법입니다. 현재 CT 2023.12는 팀 테스팅 서버에서 보고서 생성 기능을 제공하지 않아서 하나의 PC에 테스트를 모두 모아 보고서를 생성하여야 합니다.

대시보드

팀 테스팅 서버에 취합된 결과를 요약하여 대시보드에서 확인할 수 있습니다.



보고서 생성

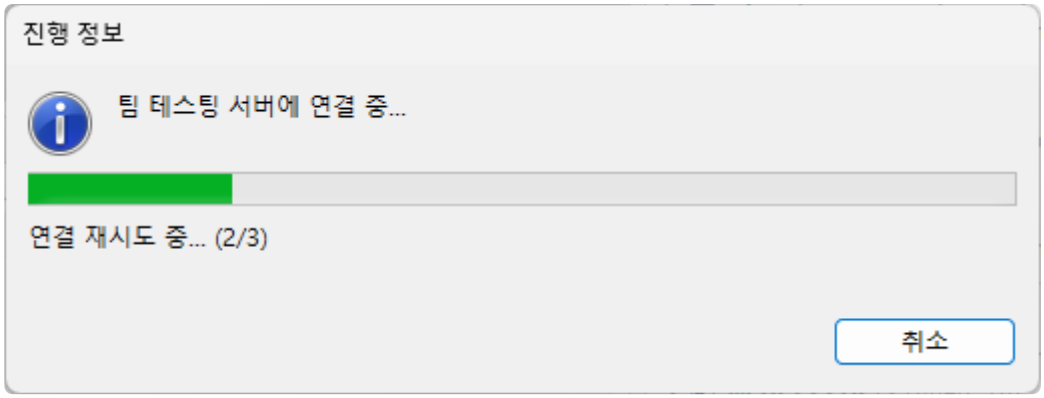
하나의 PC에 테스트를 가져와 해당 테스트에 대한 보고서를 생성할 수 있습니다.

1. [팀 테스트 가져오기] 기능을 사용하여 팀 테스팅 서버에 있는 테스트를 가져옵니다.

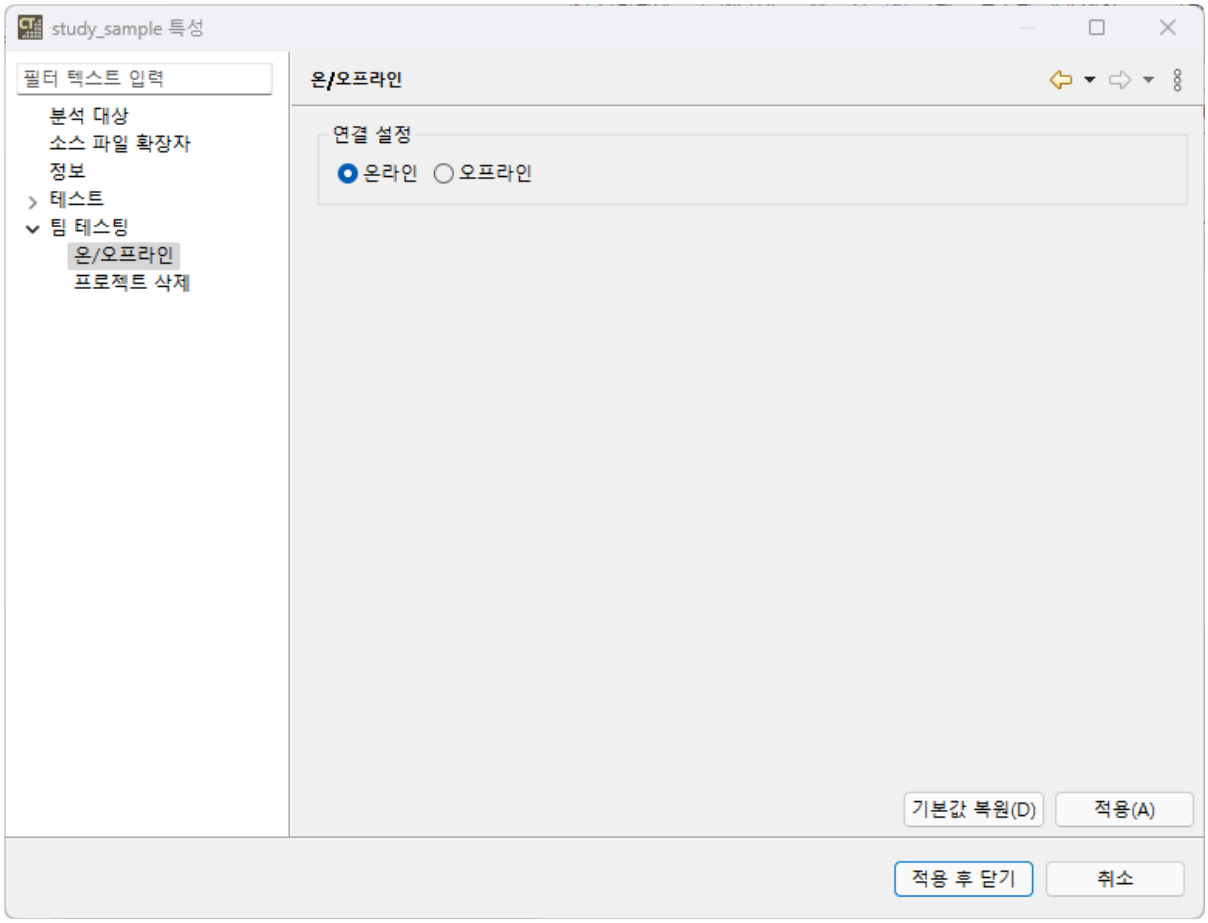
2. 팀 테스트를 가져올 때, 테스트 결과는 가져오지 않기 때문에 모든 테스트를 다시 한번 실행합니다.
3. [테스트 보고서 생성] 기능으로 해당 프로젝트의 보고서를 생성합니다.

2.1.4. 온/오프라인 모드

팀 테스팅 서버가 꺼져있거나, 네트워크 연결이 원활하지 않아 간헐적으로 끊기는 환경에서 테스트하면 팀 테스팅 서버 연결 재시도 대화상자로 인해 테스트하기 불편할 수 있습니다. 이 경우 해당 프로젝트를 오프라인 모드로 변경하여 사용할 수 있습니다.



네트워크 연결이 원활하지 않은 환경에서 연결 재시도 창이 반복해서 나오지 않도록 팀 테스팅 서버와 연결을 끊어 오프라인 모드로 작업할 수 있습니다. [특성] > [팀 테스팅] > [온/오프라인]에서 온/오프라인 모드를 변경할 수 있습니다.



오프라인 모드에서는 커밋, 업데이트 등 팀 테스팅 서버와 연결이 필요한 작업을 할 수 없습니다. 즉, 테스트를 실행해도 결과를 커밋하지 않습니다. 팀 테스팅 서버와 연결이 원활해지면 다시 온라인 모드로 전환할 수 있습니다. 온라인 모드로 전환하면 서버의 자원과 이름이 겹치지 않도록 자원의 이름을 동기화합니다. 또한, 팀 테스팅 서버의 형상과 비교하여 최신 리비전으로 업데이트를 진행합니다. 로컬의 형상을 항상 최신으로 유지하기 위해 반드시 업데이트한 후 작업을 진행하시기를 바랍니다.

팀 테스팅 서버와 동기화하지 않으므로 오프라인 모드를 가능한 한 짧게 사용하시기를 바랍니다.

2.2. 다른 사용자와 프로젝트 공유하기

CT에서 사용하는 프로젝트를 다른 사람과 공유할 수 있습니다.
Controller Tester 3.3 이상에서는 [프로젝트 내보내기], [프로젝트 가져오기] 기능을 사용합니다.

- 프로젝트 공유 시 가이드
- RTV 프로젝트 공유 시 가이드

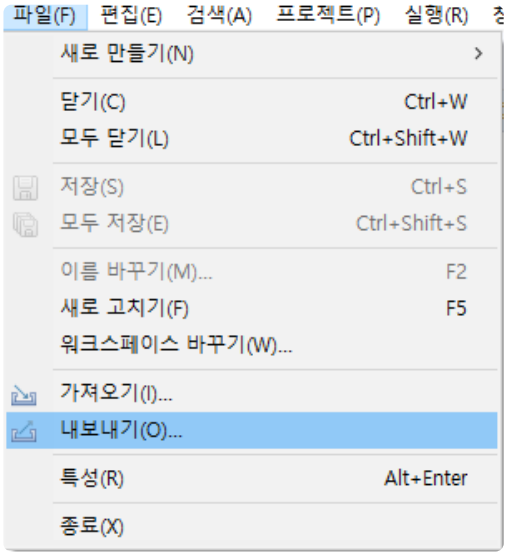
2.2.1. (Ver.3.3 이후) 프로젝트 공유 시 가이드

Controller Tester 3.3 이후부터는 [프로젝트 내보내기], [프로젝트 가져오기] 기능으로 손쉽게 프로젝트를 공유할 수 있습니다.

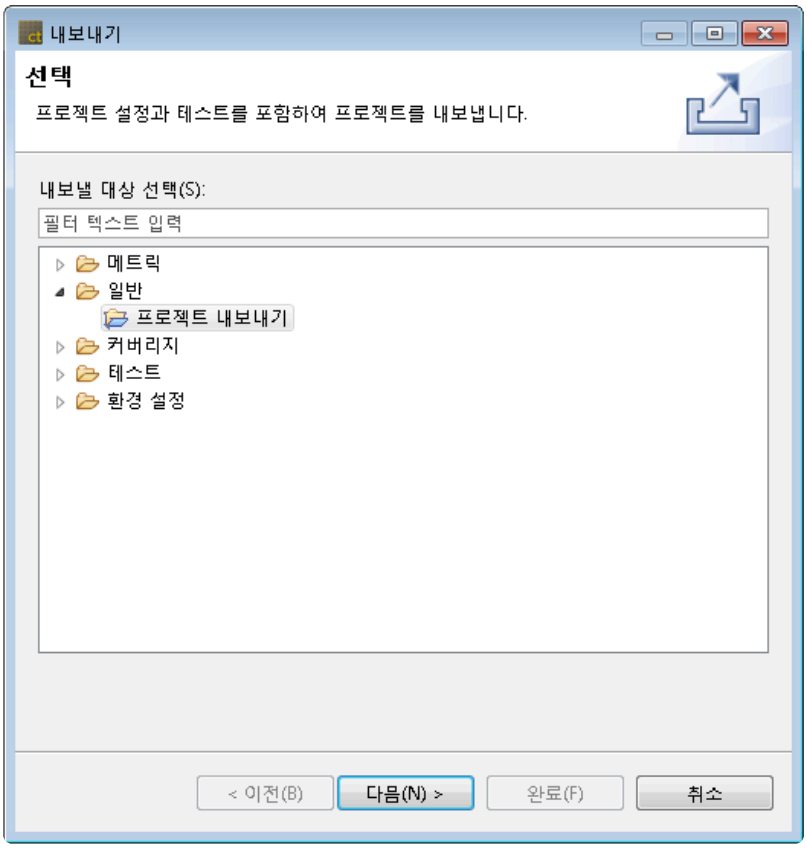
2.2.1.1. 프로젝트 내보내기

프로젝트 설정과 테스트를 포함하여 프로젝트를 내보낼 수 있습니다.

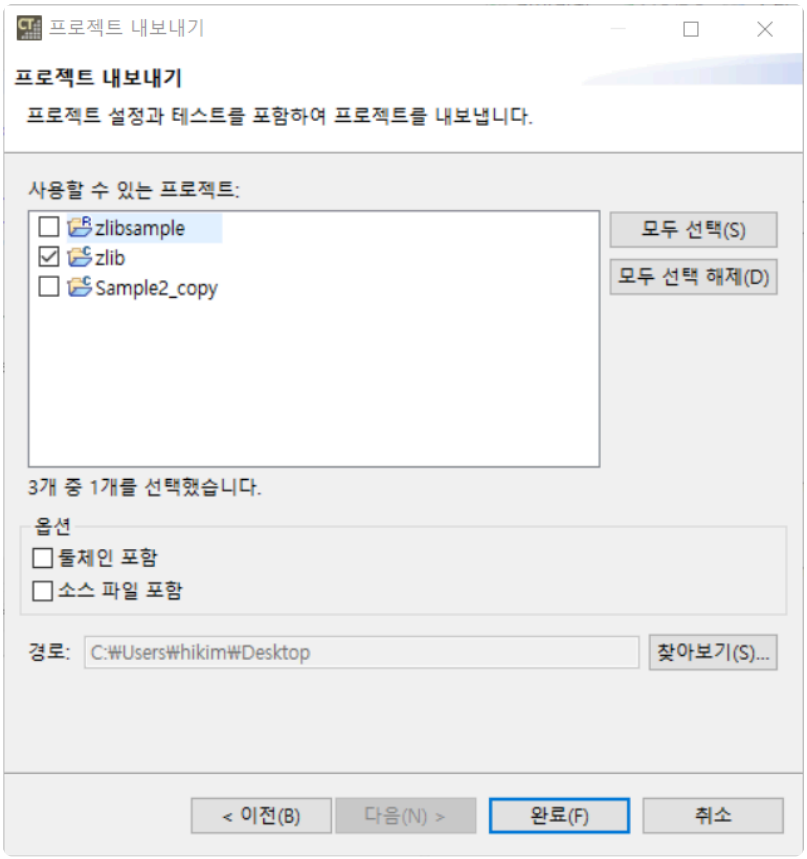
1. 메인 메뉴에서 [파일] > [내보내기]를 클릭합니다. 내보내기 마법사가 열립니다.



2. [일반] > [프로젝트 내보내기]를 클릭합니다.



3. 내보내기 대상 프로젝트와 내보낼 경로를 선택한 후, [완료] 버튼을 클릭합니다.



- (Ver.3.7 이후) 프로젝트를 내보낼 때 툴체인과 소스 파일을 포함해서 내보낼 수 있습니다.

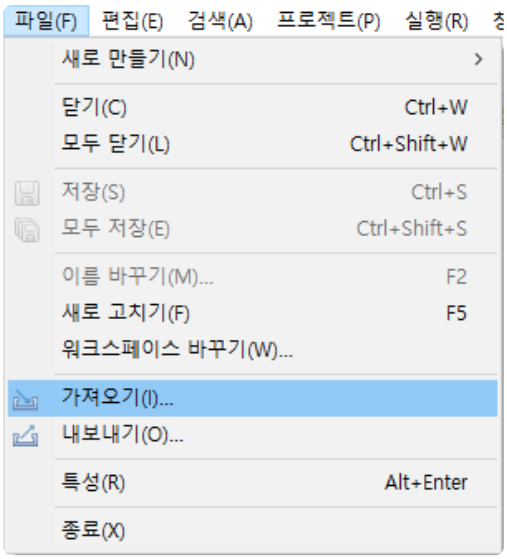
4. 내보낸 경로에 내보내기 한 프로젝트 이름이 포함된 폴더가 생긴 것을 확인할 수 있습니다. 폴더를 압축하여 공유하고자 하는 사용자의 컴퓨터로 옮깁니다.

2.2.1.2. 프로젝트 가져오기

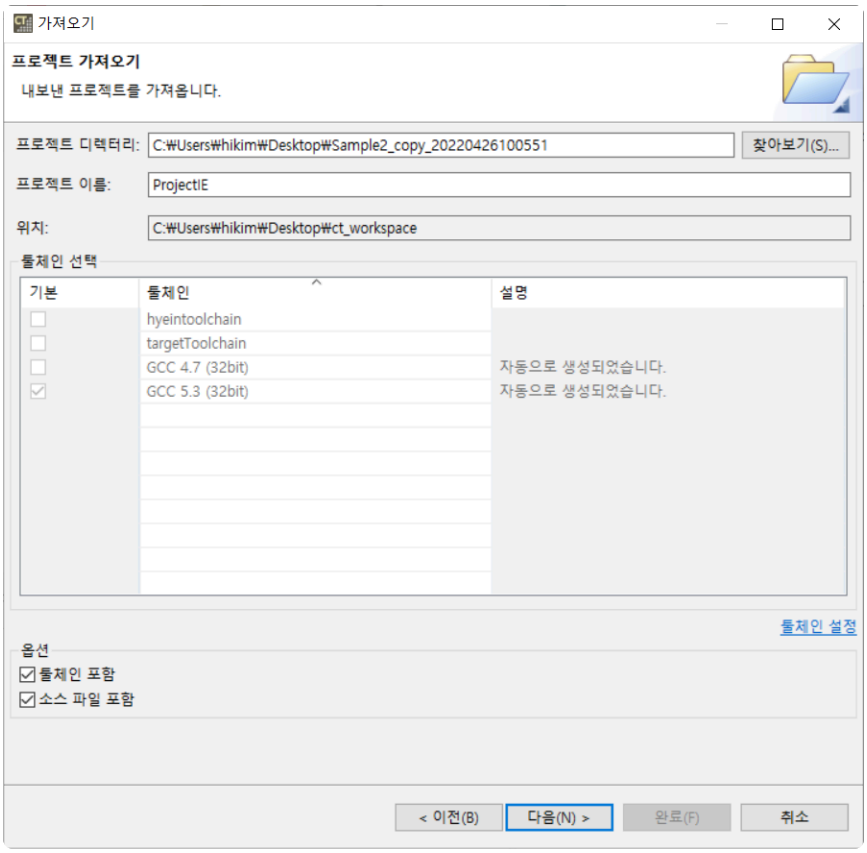
프로젝트 가져오기 기능을 이용하여, 다른 PC에서 내보낸 프로젝트를 워크스페이스로 가져올 수 있습니다.

일반 프로젝트 가져오기

1. 메인 메뉴에서 [파일] > [가져오기]를 클릭합니다. 가져오기 마법사가 열립니다.



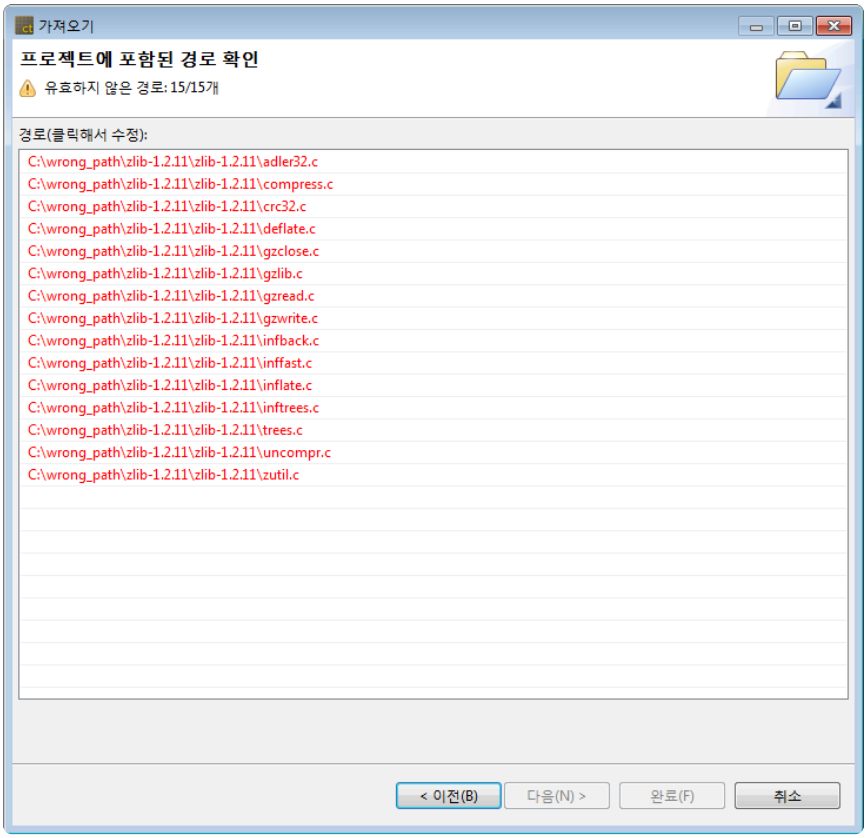
2. [일반] > [프로젝트 가져오기]를 클릭한 후 [다음] 버튼을 클릭합니다.
3. [찾아보기] 버튼을 클릭하여 내보낸 프로젝트에 해당하는 디렉터리를 찾습니다.
4. 동일한 이름의 프로젝트가 이미 워크스페이스에 존재하는 경우, 프로젝트 이름을 수정해야 합니다.



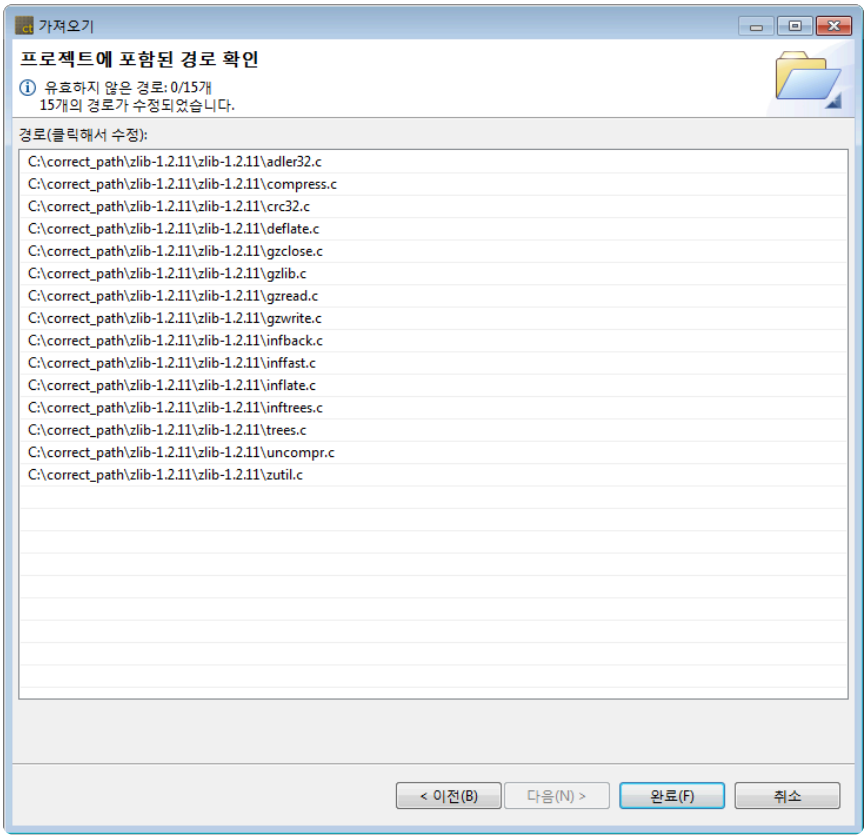
   - (Ver.3.7 이후) 디렉터리에 툴체인이나 소스 코드가 포함된 경우, 해당 옵션이 자동으로 체크됩니다.

가져올 프로젝트의 툴체인과 동일한 이름의 툴체인이 없는 경우, 먼저 가져올 프로젝트의 툴체인을 내보낸 후 가져와야 합니다. 자세한 내용은 User Manual 문서의 [툴체인 가져오기] 및 [툴체인 내보내기] 항목을 참조하십시오.

5. [다음]버튼을 클릭합니다.
6. 가져오기할 프로젝트에 포함된 소스 경로를 확인할 수 있습니다. 유효하지 않은 경로는 빨간색으로 표시되며, 경로 창을 클릭하여 수정할 수 있습니다.



7. 유효하지 않은 경로가 있을 경우, 하나의 파일 경로를 수정하면 연관된 파일 경로가 자동으로 수정됩니다. 이 때 상단에서 수정된 경로의 개수를 확인할 수 있습니다.



✻ 윈도우 절대 경로 형식이 아닌 경우에는 경로가 유효한지 여부를 검사하지 않습니다.

8. [완료]버튼을 클릭합니다.

RTV 프로젝트 가져오기

RTV C/C++ 프로젝트는 일반 C/C++ 프로젝트 가져오기와 동일한 방법으로 가져올 수 있습니다.

1. 메인 메뉴에서 [파일] -> [가져오기]를 클릭합니다. 가져오기 마법사에서 [일반] -> [프로젝트 가져오기]를 선택하고 [다음]을 누릅니다.
2. [찾아보기] 버튼을 클릭하여 가져올 프로젝트의 디렉터리를 선택합니다. 디렉터리를 선택하면 가져오기할 프로젝트 정보로부터 툴체인이 자동으로 선택됩니다. [다음] 버튼을 클릭합니다.

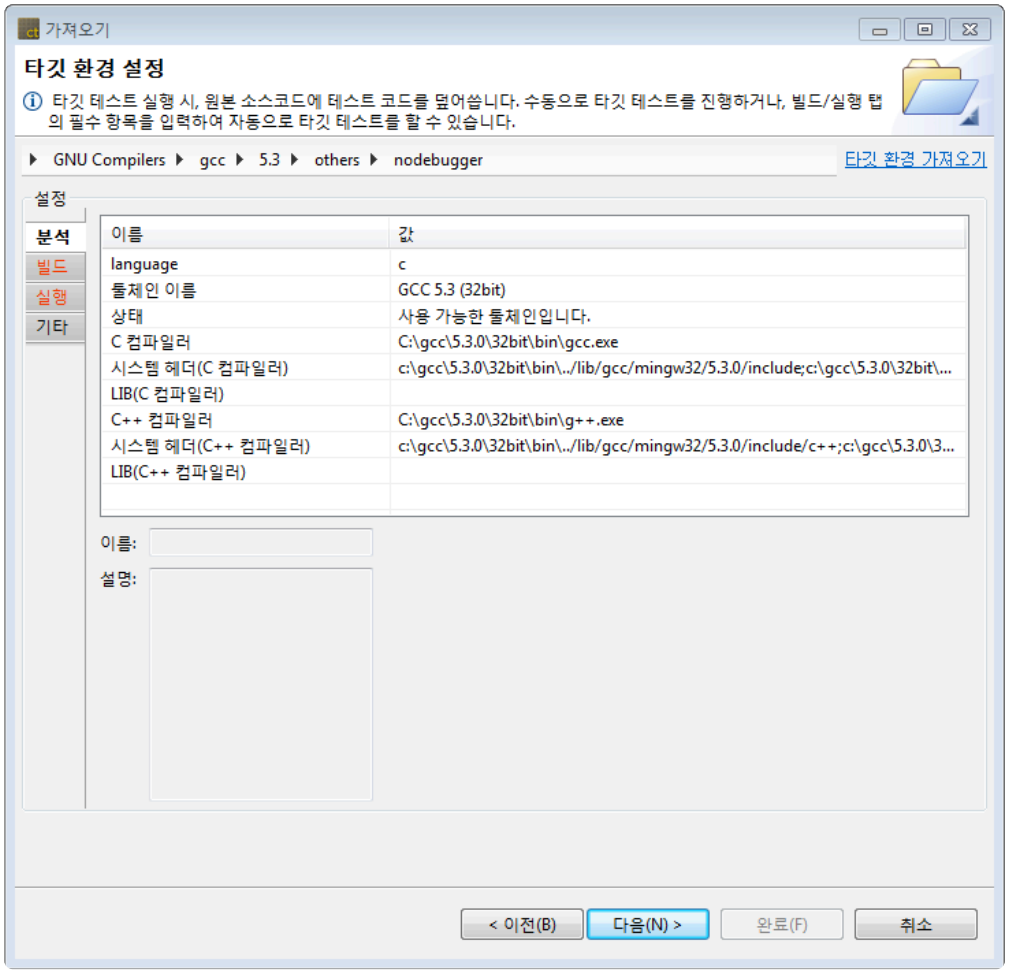
✻ 가져올 프로젝트와 동일한 RTV 서버 및 툴체인 정보가 없으면, 가져올 프로젝트로부터 자동으로 RTV 서버 및 툴체인 정보를 생성합니다.

3. 가져오기할 프로젝트에 포함된 소스 경로를 확인할 수 있습니다. 유효하지 않은 경로는 빨간색으로 표시되며, 경로 창을 클릭하여 수정할 수 있습니다.
4. [완료]버튼을 클릭합니다.

타깃 프로젝트 가져오기

타깃 C/C++ 프로젝트를 가져올 때는 타깃 환경 설정을 추가로 작성해야 합니다.

1. 메인 메뉴에서 [파일] -> [가져오기]를 클릭합니다. 가져오기 마법사에서 [일반] -> [프로젝트 가져오기]를 선택하고 [다음]을 누릅니다.
2. [찾아보기] 버튼을 클릭하여 가져올 프로젝트의 디렉터리를 선택합니다. 디렉터리를 선택하면 가져오기할 프로젝트 정보로부터 툴체인이 자동으로 선택됩니다. [다음] 버튼을 클릭합니다.
3. 타깃 프로젝트의 경우 [타깃 환경 설정] 창이 뜹니다. 가져올 프로젝트 정보로부터 타깃 환경 설정을 불러오며, 유효하지 않은 경로를 가진 항목은 빨간색으로 표시됩니다.



✻ 타깃 C/C++ 프로젝트가 아니더라도 타깃 환경 설정을 포함하고 있는 프로젝트라면, [프로젝트 가져오기]를 수행할 때 타깃 환경 설정 창이 뜹니다.

! 유효하지 않은 경로를 포함하고 있어도 타깃 환경 설정을 완료하고 다음으로 넘어갈 수 있으나, 원클릭 타깃 테스트 실행이 안될 수 있습니다.

4. 타깃 환경 설정을 완료하고 [다음] 버튼을 클릭합니다.
5. 가져오기할 프로젝트에 포함된 소스 경로를 확인할 수 있습니다. 유효하지 않은 경로는 빨간색으로 표시되며, 경로 창을 클릭하여 수정할 수 있습니다.
6. [완료]버튼을 클릭합니다.

2.2.2. (Ver.3.2 이전) RTV 프로젝트 공유 시 가이드

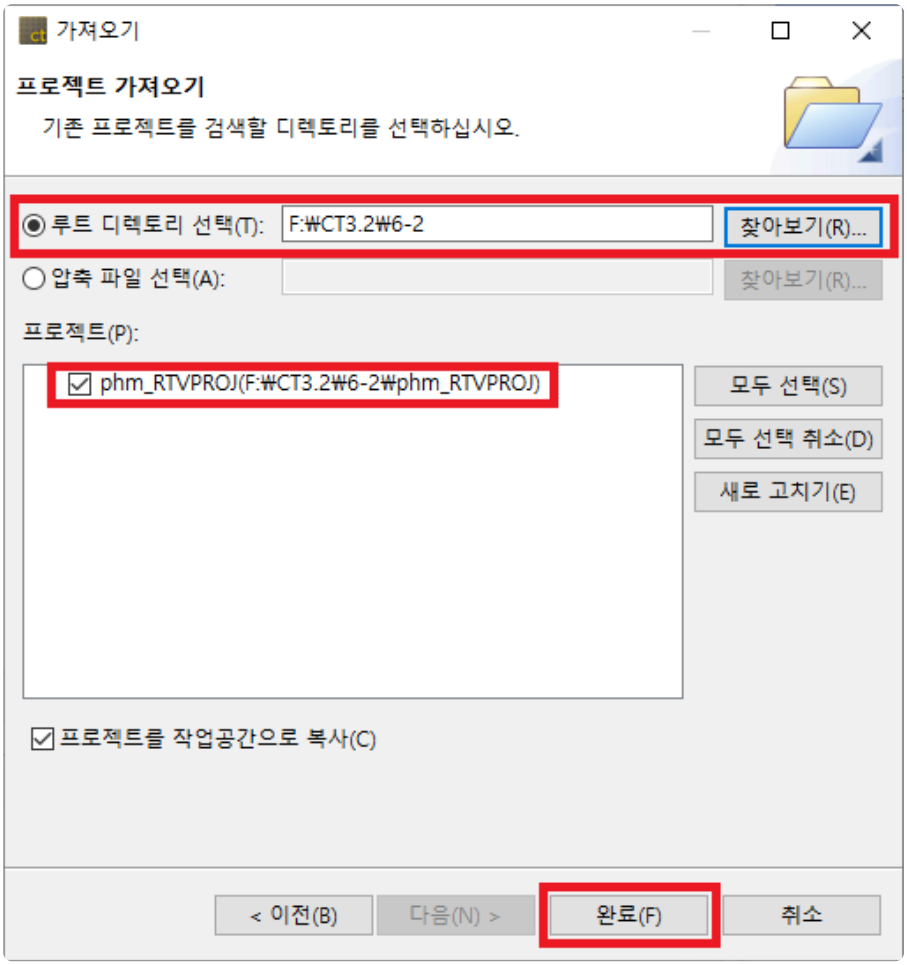
RTV 프로젝트는 툴체인과 소스 파일 정보를 RTV 서버에서 가져올 수 있기 때문에 쉽게 프로젝트를 공유할 수 있습니다.
사용 환경에 따른 단계별 시나리오는 다음과 같으며, 해당 시나리오대로 수행했을 때 RTV 프로젝트를 공유할 수 있습니다.

- 프로젝트 공유 시나리오
- RTV 서버 사용 가이드

2.2.2.1. 프로젝트 공유 시나리오

[기존 프로젝트를 워크스페이스로 가져오기] 기능을 사용할 경우

1. RTV 프로젝트를 생성하면 CT 워크스페이스 아래에 RTV 프로젝트 디렉토리(이하 RTV_A 프로젝트)가 생성됩니다.
2. 프로젝트를 공유받고자 하는 사용자가 위 단계에서 생성된 RTV_A 프로젝트 디렉토리를 전달받아, 자신이 사용하는 CT 워크스페이스 경로에 RTV_A 프로젝트 디렉토리를 복사 후 붙여넣습니다.
3. [가져오기] > [일반] > [기존 프로젝트를 워크스페이스로] 기능을 사용하여 해당 프로젝트 디렉토리 최상위 경로를 선택해 가져옵니다.



4. 프로젝트에 필요한 정보들을 RTV 서버에서 받아오게 되며, [프로젝트의 툴체인 또는 리소스 설정이 올바르지 않습니다. 자동으로 재설정하시겠습니까?] 라는 메세지 창이 뜨는 경우 '예'를 클릭하면 자동으로 RTV 설정(RTV 서버와 프로젝트 생성 시 사용한 툴체인 등록)이 완료됩니다.
5. Controller Tester 테스트 네비게이터 뷰에 RTV_A와 동일한 이름의 RTV 프로젝트(이하 RTV_A' 프로젝트)가 생성된걸 확인할 수 있습니다.
6. [테스트 네비게이터 뷰]의 RTV_A' 프로젝트를 우클릭하고 [재분석]을 수행합니다.
7. 동일한 RTV 서버와 연결되어 있을 때 수행해야 합니다.

[RTV 빌드 C/C++ 프로젝트 생성] 기능을 사용할 경우

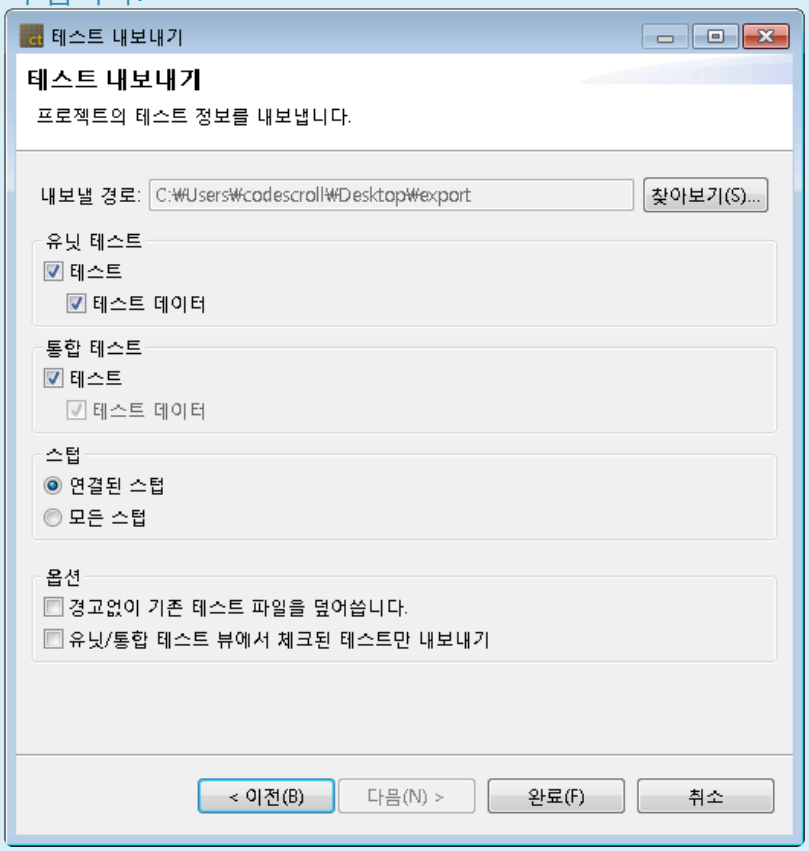
1. RTV 프로젝트를 생성하면 CT 워크스페이스 아래에 RTV 프로젝트 디렉토리(이하 RTV_A 프로젝트)가 생성됩니다.

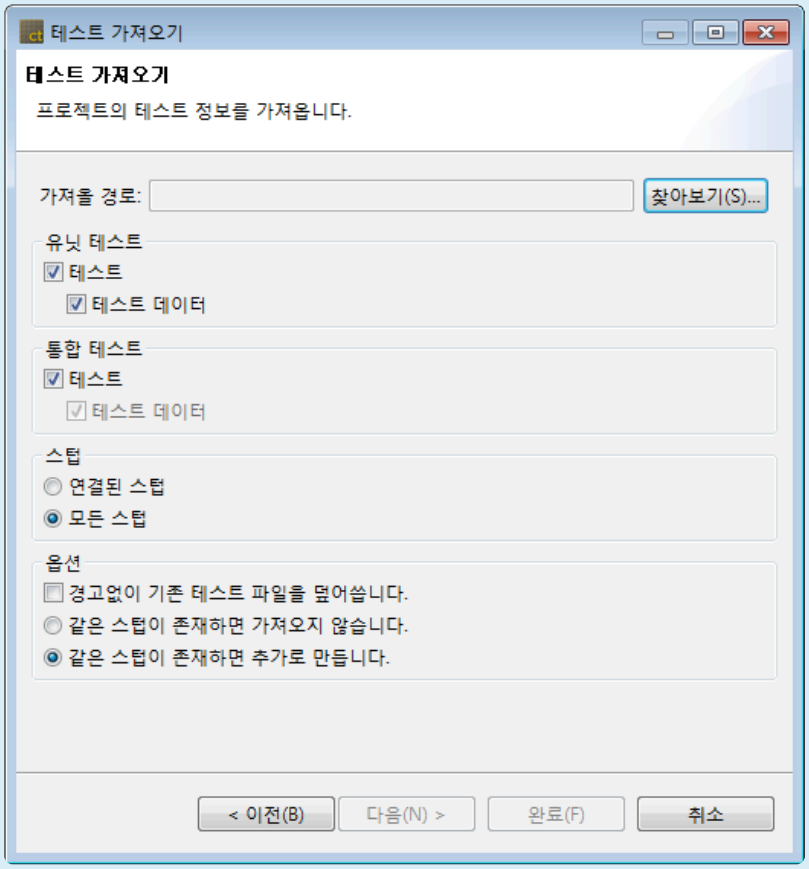
2. 프로젝트를 공유받고자 하는 사용자가 자신이 사용하는 CT에서 위 프로젝트를 생성한 곳과 동일한 RTV 서버를 연결하고, 동일한 RTV 툴체인을 등록합니다.
3. 프로젝트 생성 마법사에서 [RTV 빌드 C/C++ 프로젝트] 를 선택하여 RTV 프로젝트(이하 RTV_A' 프로젝트)를 생성합니다.
4. CT 워크스페이스의 RTV_A 프로젝트 폴더로부터 *$(프로젝트 폴더)/.csdata/link.mk* 파일을 가져와 RTV_A' 프로젝트 폴더의 link.mk 파일을 덮어씁니다.
5. 동일한 테스트 데이터를 공유하고자 할 때는 하단의 [참고2]를 확인하십시오.

✻ 소스 파일이 있는 경로가 길면 전체 소스 파일을 제대로 못 가져오는 경우가 있을 수 있습니다. 소스 파일이 있는 경로가 너무 긴 경우, CT의 글로벌 경로를 드라이브 바로 아래에 지정하도록 합니다. (ex. *C:\temp*)
CT 글로벌 경로를 수정하는 방법은 CT 패키지가 설치된 위치에 있는 CodeScroll.ini 파일을 열고 -g 옵션 아래의 default를 새롭게 설정할 글로벌 경로로 바꾸면 됩니다.

```
-startup
plugins/org.eclipse.equinox.launcher_1.4.0.v20161219-1356.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.500.v20170531-1133
-data
@noDefault
-g
C:\temp
-vmargs
```

✻ 동일한 프로젝트를 공유한 경우에도, 유닛 테스트를 각각 생성하면 커버리지 결과가 다를 수 있습니다. 테스트를 공유할 때에는 [내보내기] > [테스트 내보내기] 기능을 사용하여 테스트를 내보낸 후, [가져오기] > [테스트 가져오기] 기능을 사용하여 내보냈던 테스트를 가져와야 합니다.




테스트 가져오기
테스트 가져오기
프로젝트의 테스트 정보를 가져옵니다.
가져올 경로:
찾아보기(S)...
유닛 테스트
테스트
테스트 데이터
통합 테스트
테스트
테스트 데이터
스텁
연결된 스텁
모든 스텁
옵션
경고없이 기존 테스트 파일을 덮어씁니다.
같은 스텁이 존재하면 가져오지 않습니다.
같은 스텁이 존재하면 추가로 만듭니다.
< 이전(B)
다음(N) >
완료(F)
취소

2.2.2.2. RTV 서버 사용 가이드

하나의 RTV 서버를 사용하는 경우

1. RTV 서버에 csbuild capture 기능을 사용하여 빌드한 프로젝트가 있는 경우
    a. 별도의 설정이 필요 없이, 위의 프로젝트 공유 시나리오에 따라 프로젝트를 가져올 수 있습니다.
2. RTV 서버 연결은 되어 있으나, 서버(아이피/포트) 정보가 다른 경우
    a. 프로젝트를 생성할 당시의 서버(아이피/포트) 정보를 가져오기 때문에, 기존 서버 정보를 가져오며 툴체인 정보를 가져오지 않습니다.
    b. 서버 정보를 동일 서버에 접속할 수 있도록 수정한 후, 툴체인 가져오기를 통해 동일한 이름의 툴체인을 가져옵니다. 이 때, 프로젝트에서 사용한 툴체인의 경로는 같아야 합니다.

> ! 둘 이상의 RTV 서버를 사용하는 경우(동일한 소스 파일, 툴체인을 구성하여 사용하려는 경우나 RTV 서버가 설치된 가상머신 파일을 전달받아 사용하려는 경우)에는 RTV 프로젝트 공유가 어려울 수 있습니다.

2.3. 커버리지 가져오기 가이드

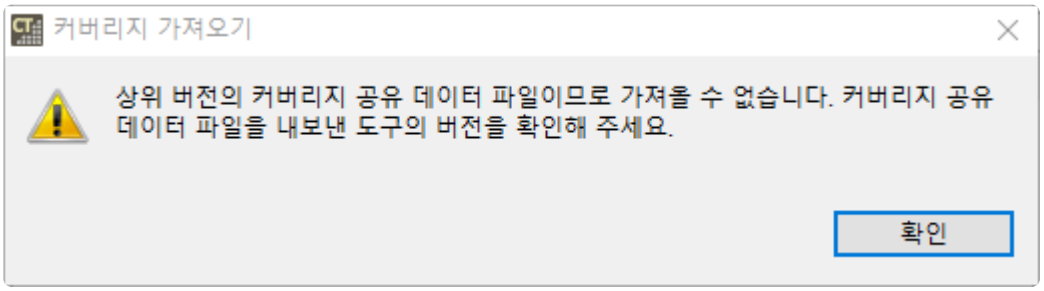
다른 환경의 CT 2023.12나 COVER에서 커버리지를 가져올 때, 다음 세 가지 기준에 따라 가져옵니다. 기준에 맞지 않는 경우, 커버리지 가져오기가 실패할 수 있습니다.

- 커버리지 공유 파일의 버전
- 삼항 연산자 설정
- 커버리지 종류

2.3.1. 버전에 따른 커버리지 가져오기

CT 2023.12로 커버리지를 가져올 때, 커버리지 공유 파일의 버전을 확인합니다.

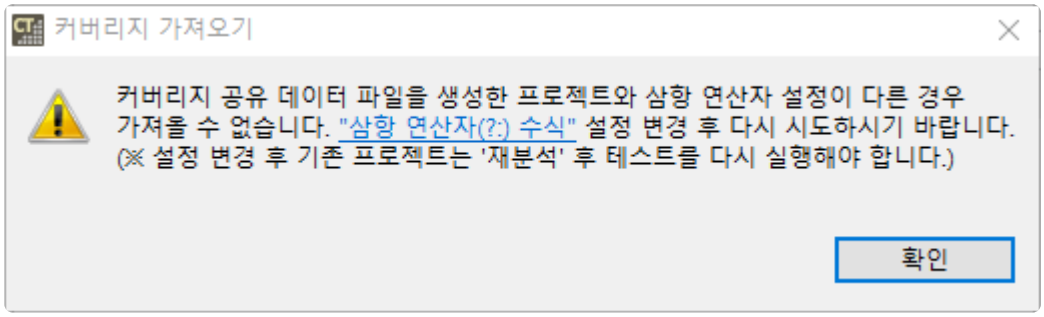
- 버전이 높은 커버리지 공유 파일을 가져올 경우, 커버리지를 가져올 수 없습니다.



- 버전이 낮은 커버리지 공유 파일을 가져올 경우, 커버리지를 내보낸 도구의 설정에 따라 일부 함수의 커버리지를 가져오지 못할 수 있습니다.

2.3.2. 삼항 연산자 설정에 따른 커버리지 가져오기

CT 2023.12의 삼항 연산자 설정이 커버리지 공유 파일을 내보낸 도구와 다른 경우, 커버리지를 가져올 수 없습니다.



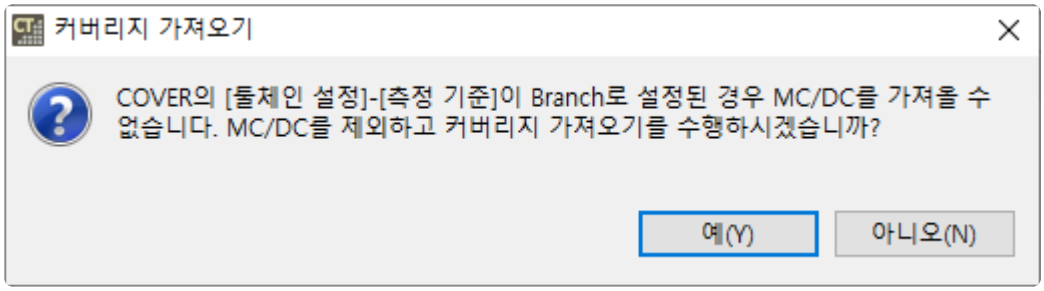
해당 경고창의 링크를 누르거나 [환경설정] > [테스트] > [커버리지] > [분기 및 MC/DC 커버리지 측정 대상 연산자]에서 [삼항 연산자(?:) 수식] 설정을 가져올 파일에 맞게 변경합니다.

- COVER의 [툴체인 설정] > [측정 기준]이 [COVER] > [Branch]인 경우, [삼항 연산자(?:) 수식] 설정을 꺼야합니다.
- COVER의 [툴체인 설정] > [측정 기준]이 [COVER] > [MC/DC]인 경우, [삼항 연산자(?:) 수식] 설정을 켜야합니다.

> ! 해당 설정을 변경하면 재분석 후 테스트를 다시 실행해야 합니다.

2.3.3. 커버리지 종류에 따른 커버리지 가져오기

Controller Tester 3.6부터, 커버리지 종류가 다른 경우에도 사용자가 커버리지를 가져올 수 있습니다.



[예]를 선택하면, MC/DC 커버리지를 제외하고 구문 커버리지와 분기 커버리지를 가져옵니다. [아니오]를 선택할 경우 커버리지를 가져오지 않습니다.

3. 시나리오(시간 기반) 테스트 사용 가이드

요구사항 테스트를 하다보면 다음과 같은 요구사항을 만나게 됩니다.

```
자동차의 열려있던 문이 닫히면, 실내등이 5초간 켜진 후 꺼진다.
```

이를 위한 타이머 함수가 있을 때, 해당 타이머 함수를 정해진 시간마다 반복 호출하여 5초 후 실내등이 꺼졌는지 확인하여야 합니다.

이처럼 주기적으로 호출되는 함수를 테스트해야 하는 경우가 있습니다. CT 2023.12부터 이러한 테스트를 돕기 위해 시나리오 테스트 기능을 제공합니다.

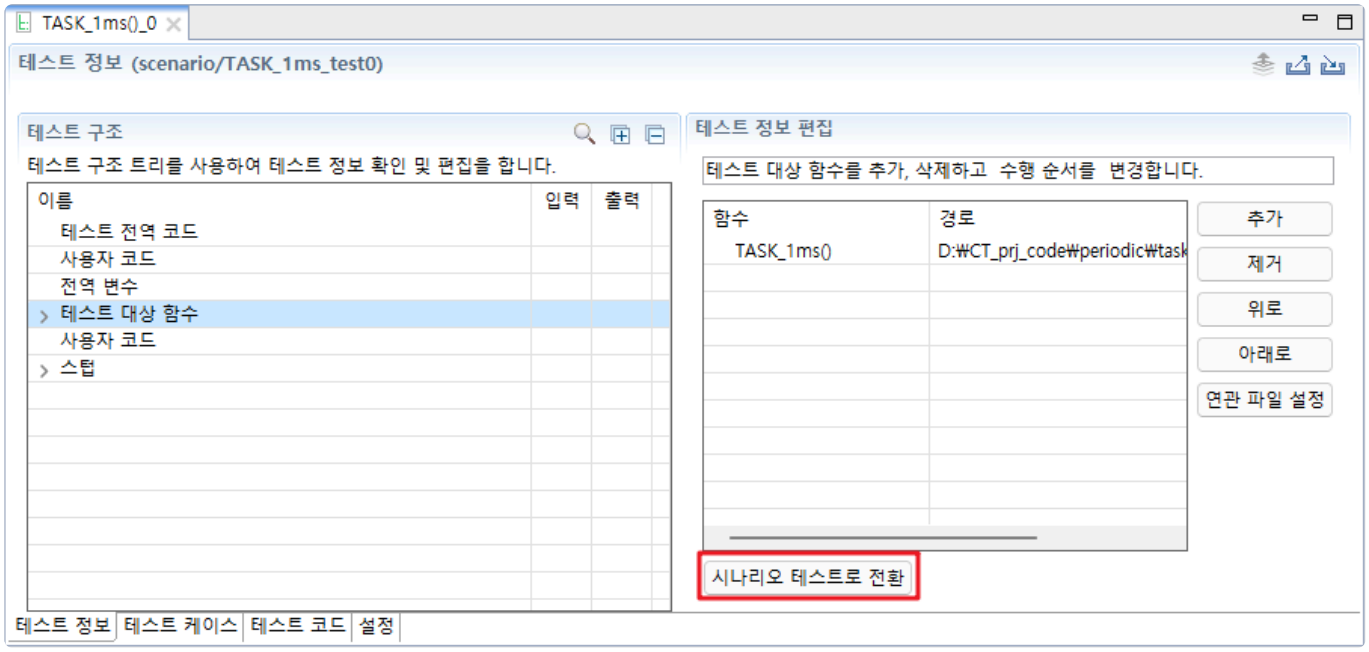
시나리오 테스트의 조건

일반 테스트를 시나리오 테스트로 전환하기 위해서는 아래 조건을 모두 만족하여야 합니다.

- C 프로젝트
- 테스트 대상 함수에 반환형이 void이고 매개변수가 없는 함수만 있어야 함

시나리오 테스트로 전환

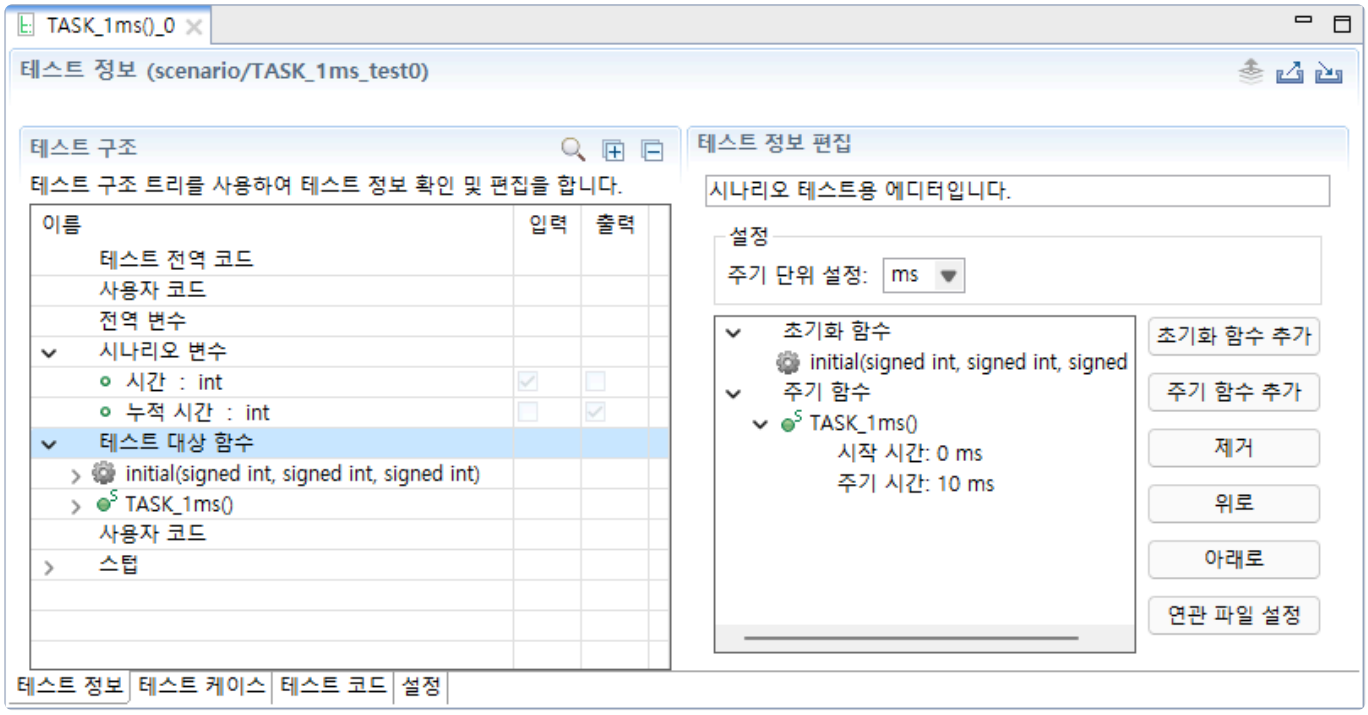
[테스트 편집기] > [테스트 정보] > [테스트 대상 함수]에서 [시나리오 테스트로 전환]을 선택하여 시나리오 테스트로 전환합니다.



> 시나리오 테스트로 전환한 후에는 일반 테스트로 되돌릴 수 없습니다.

시나리오 테스트는 테스트 케이스 컨텍스트가 유지됩니다. 즉, 이전 테스트 케이스의 상태를 유지한 채 다음 테스트 케이스를 실행합니다.

시나리오 테스트



시나리오 테스트를 돕기 위한 변수 두 개가 추가됩니다.

- 시간
  - [테스트 케이스 탭] 에서 값을 입력할 수 있습니다.
  - 테스트 케이스별로 for 문의 반복 횟수를 결정합니다.
- 누적 시간
  - 테스트 케이스의 for 문 반복 횟수를 누적하여 보여줍니다.

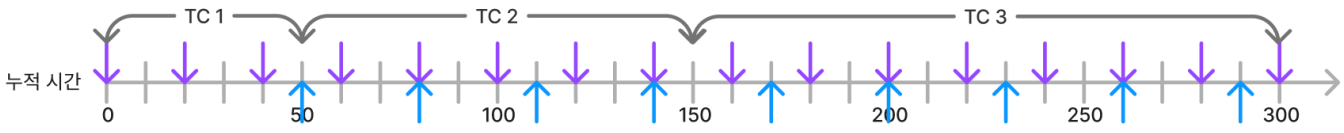
테스트 대상 함수 중, 주기 함수에 두 가지 설정이 추가됩니다.

- 시작 시간
  - 함수 호출을 시작하는 시간입니다.
  - 시작 시간이 40인 경우 누적 시간이 0~39일 때는 해당 함수가 호출되지 않습니다.
- 주기 시간
  - 함수 호출을 반복하는 시간 간격입니다.
  - 주기 시간이 10인 경우, 누적 시간이 10, 20, 30, … 일 때 호출됩니다.

아래는 시나리오 테스트에 대한 이해를 돕기 위한 그림입니다.

	시작 시간	주기 시간
20ms_Task	0	20
30ms_Task	50	30

TC	시간	누적 시간
1	50	50
2	100	150
3	150	300



주기 단위는 ms, μs, ns로 설정할 수 있습니다. 이 값은 테스트 보고서와 테스트 편집기에서만 표시되며 실제 실행 시간에 영향을 미치지 않습니다. 즉, 주기 단위를 ms로 설정하고 누적 시간이 5000이라고 하여도 실제로 5초 동안 수행되지 않습니다. 시나리오 테스트는 해당 시간을 모사하여 테스트를 진행합니다.

시나리오 테스트의 테스트 코드

[테스트 정보]에 설정한 값을 토대로 아래와 같이 테스트 코드가 생성됩니다.
초기화 함수의 경우, for 문을 반복하기 전에 한 번만 호출되는 함수이며 1번 테스트 케이스에서 최초로 한번 호출됩니다. 이후, 다른 테스트 케이스에서는 호출되지 않습니다.

```
        /*Declaration (parameter/return/target object) variables*/
        unsigned int CS_TC_SPENT_TIME = 0;  // 시간의 입력값이 저장됩니다.
        static unsigned int CS_TOTAL_SPENT_TIME = 0; // 테스트 케이스의 반복 횟수
를 저장합니다. 이 변수의 값이 누적 시간의 출력값으로 저장됩니다.

        /*Input*/
        CS_TC_SPENT_TIME = CS_INT_INPUT(unsigned int,"CS_TC_SPENT_TIME");

        /* call initial function */
        if(CS_TOTAL_SPENT_TIME==0) {
                // 초기화 함수가 추가되는 위치
        }

        // 주기 함수들이 반복되는 부분. 시간의 입력값만큼 for문을 반복합니다.
        for(int CS_CYCLE_INDEX = 0; CS_CYCLE_INDEX < CS_TC_SPENT_TIME ; CS_CYC
LE_INDEX++ ) {
                if( (CS_TOTAL_SPENT_TIME>=0) && ((CS_TOTAL_SPENT_TIME-0)%1
0==0) ) {

                        /* TASK_1ms() */
                        TASK_1ms();
```

```
            }
            CS_TOTAL_SPENT_TIME++;
        }

        /*Output*/
        CS_INT_OUTPUT(CS_TOTAL_SPENT_TIME, "CS_TOTAL_SPENT_TIME");
```

예제

시나리오 테스트를 활용하여 다양한 요구사항을 만족할 수 있습니다.

- 시나리오 테스트 실행 중에 특정 변수의 변화를 확인하기
- 전역 변수의 값에 따라 함수 호출 여부를 결정하기

3.1. 시나리오 테스트 실행 중에 특정 변수의 변화를 확인하기

시나리오 테스트 실행 중에, 특정 변수의 값을 확인하고자 할 때는 테스트 케이스를 나누어 확인합니다. 예를 들어, 주기 단위가 ms인 테스트에서 2초와 4초에 특정 변수의 변화를 알고 싶다면 다음과 같이 테스트 케이스를 설계합니다.

테스트 케이스(TC)	시간	누적 시간
1	1999	1999
2	1	2000
3	1999	3999
4	1	4000

TC1과 TC2를 통해 2초 전/후의 값 변화를 확인하고 TC3과 TC4를 통해 4초 전/후의 값 변화를 확인합니다.

이러한 방식으로 테스트 케이스를 나누어 원하는 시점에 원하는 값을 확인할 수 있습니다. 간단한 예제를 통해 자세히 설명하겠습니다.

소스 코드 및 요구사항

예제로 사용할 소스 코드와 요구사항은 아래와 같습니다.

```
#include <stdio.h>
#include <stdbool.h>

typedef enum {
    CLOSED, OPEN
} OpenCloseState;

typedef enum {
    OFF, ON
} OnOffState;

OpenCloseState doorState;
OpenCloseState doorSensor;
OnOffState ignitionState;
OnOffState lightState;

void initial() {
        doorState = OPEN;
        doorSensor = OPEN;
        ignitionState = OFF;
```

```
        lightState = OFF;
}

void lightOn() { if (lightState != ON)     lightState = ON; }

void lightOff() { if (lightState != OFF)     lightState = OFF; }

void setDoorSensor(OpenCloseState sensor) {
        doorSensor = sensor;
}

void tick() {
    static int timer = 0;

    if (doorState == OPEN && doorSensor == CLOSED) {
        timer = 500;
        lightOn();
    } else if (ignitionState == ON){
        timer = 0;
        lightOff();
    }

    if (timer > 0)     timer--;

    if (timer == 0)     lightOff();

    doorState = doorSensor;
}
```

`요구사항: 자동차의 열려있던 문이 닫히면, 실내등이 5초간 켜진 후 꺼진다.`

테스트 설계

이러한 요구사항을 만족하는 테스트를 설계하려고 합니다.

주기 함수와 주기 단위 설계

`실내등이 5초간 켜진 후 꺼진다`와 `timer = 500;`을 통해, timer의 시간 단위는 ms이며, `tick()` 함수가 10ms마다 호출되는 함수임을 알 수 있습니다. 따라서, `tick()`을 주기 함수로 설정하고, 주기 단위를 ms, `tick()`의 시작 시간을 0, 주기 시간을 10으로 설정합니다.

초기화 함수 설계

`initial()` 함수를 초기화 함수로 호출하여 각 센서와 상태에 초깃값을 할당합니다. 이때, `doorState`와 `doorSensor`의 초깃값 모두 `OPEN`입니다. 자동차의 `열려있던 문이 닫히면`이라는 요구사항에 따라 `setD`

oorSensor() 함수의 매개변수로 CLOSED 값을 주어 문 닫힘을 설정합니다.

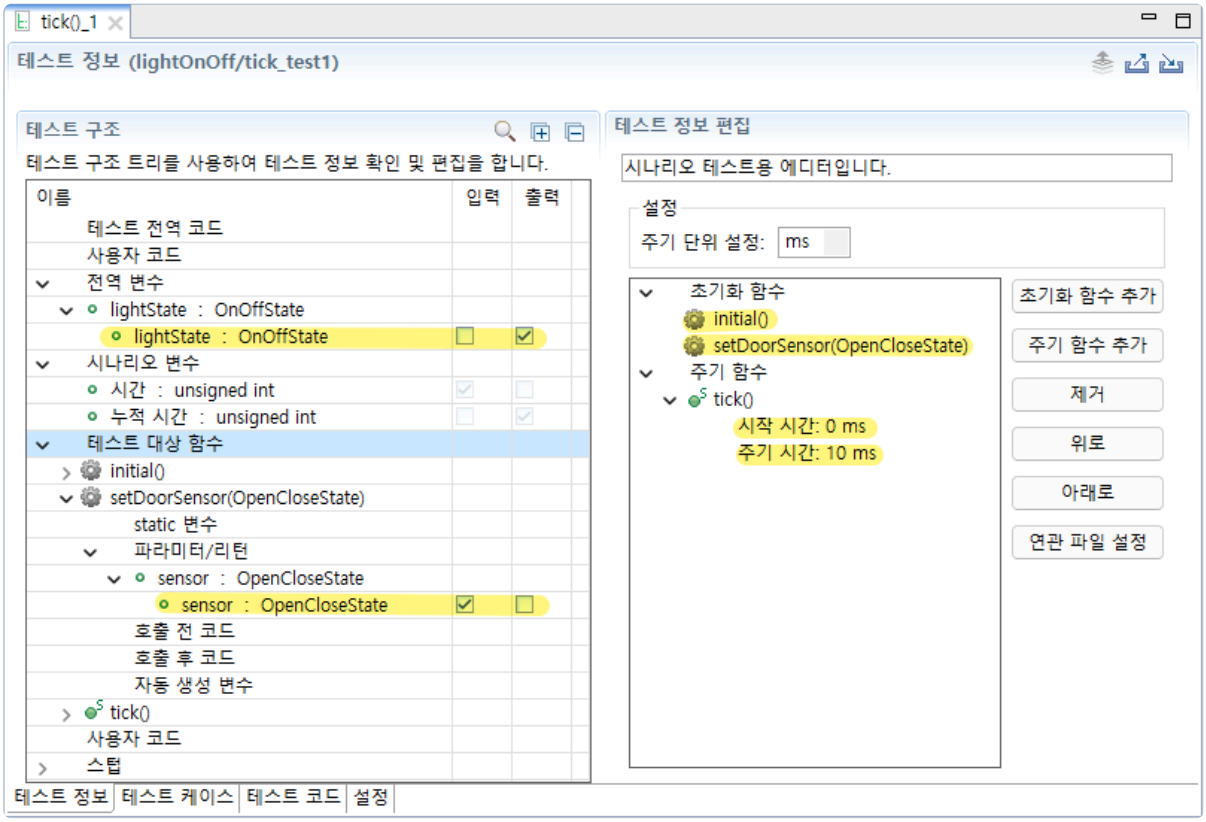
테스트 케이스 및 시나리오 변수(시간) 설계

요구사항을 확인하기 위해 lightState의 출력값을 확인합니다. 5초간 실내등이 켜져 있고 5초 이후에 실내등이 꺼져야 합니다. 즉, tick()이 499회 호출될 때까지 lightState가 ON이고, 500번째 호출에서 lightState가 OFF가 됩니다. 이를 확인하기 위해 첫 번째 테스트 케이스에서 시간을 4990, 두 번째 테스트 케이스에서 시간을 10으로 입력합니다.
전역 변수에서 lightState를 선택하고 [출력]에 체크하여 테스트 후 lightState의 값을 확인합니다.

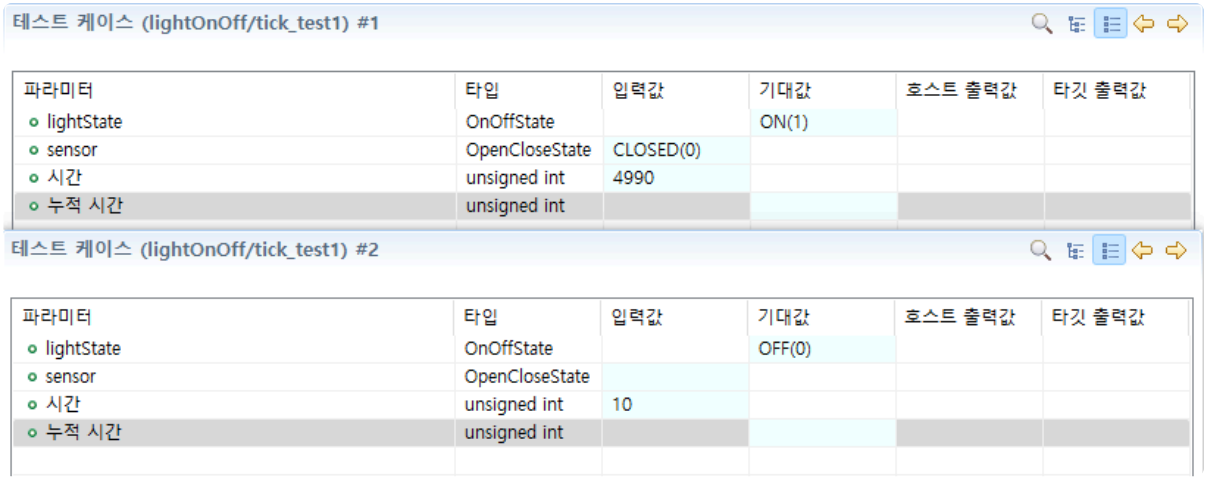
테스트 작성

위 설계를 토대로 테스트를 작성하면 다음과 같습니다.

- [테스트 정보] 탭

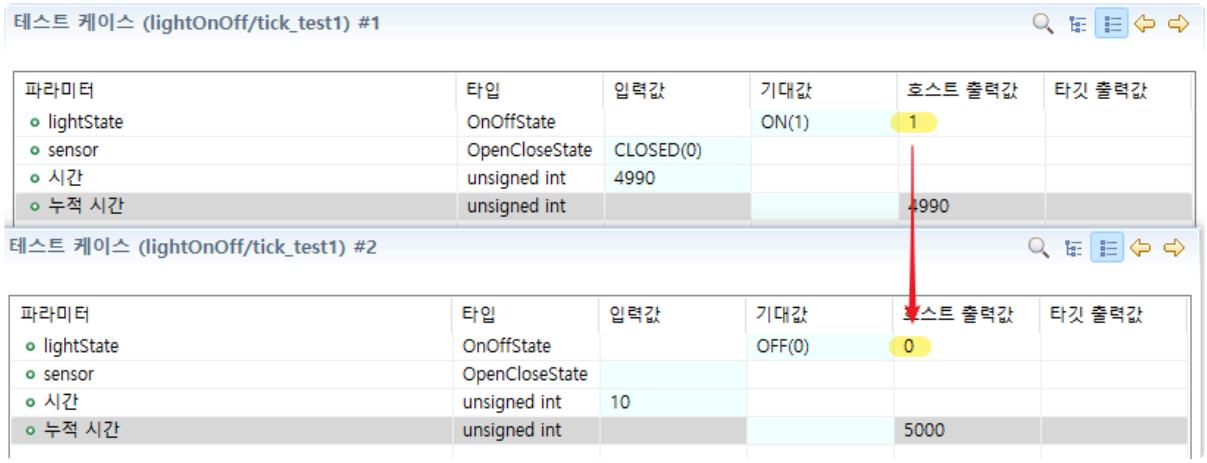


- [테스트 케이스] 탭



테스트 결과 확인

테스트를 실행하고 [테스트 케이스] 탭에서 결과를 확인합니다.



0ms~4990ms 동안 `lightState`가 `ON(1)`이고 5000ms가 될 때 `lightState`가 `OFF(0)`이 됩니다. 시나리오 테스트를 이용하여 소스 코드가 요구사항을 만족하는 것을 확인했습니다.

3.2. 전역 변수의 값에 따라 함수 호출 여부를 결정하기

전역 변수의 값에 따라 함수의 호출 여부가 결정될 때는 [함수 호출 전/후 코드]를 이용하여 테스트합니다.

```
// Before call code
if( globalVar == 1 ) {
// Function call code.
func()
// After call code
}
```

이러한 방식으로 함수 호출 전/후 코드에 if 문을 추가하여 특정 조건을 만족할 때 함수를 호출하도록 합니다. 신호등과 음향신호기 예제를 통해 자세히 설명하겠습니다.

소스 코드 및 요구사항

예제로 사용할 소스 코드와 요구사항은 아래와 같습니다.

```
#include <stdio.h>

typedef enum {
    RED,
    GREEN
} TrafficLightState;

typedef enum {
        ON,
        OFF
} SoundSystemState;

TrafficLightState trafficLight;
SoundSystemState soundSystem;

void init() {
        trafficLight = GREEN;
        soundSystem = OFF;
}

void setSoundSystem (SoundSystemState state) {
        soundSystem = state;
}

void alarmForBlind() {
```

```
        if( trafficLight == GREEN ) {
                printf("beep for blind\n");
        } else if( trafficLight == RED ) {
                printf("warning for blind\n");
        }
}

void tick() {
    static int timer = 50;

        if(timer > 0) {
                timer--;
        }

        if( timer == 0 ) {
                timer = 50;
                if( trafficLight == GREEN ) {
                        trafficLight = RED;
                } else if( trafficLight == RED ) {
                        trafficLight = GREEN;
                }
        }
}
```

```
요구사항: 빨간불과 파란불은 각각 5분 동안 켜진다.
음향신호기가 켜져 있고 신호등이 파란불이면, 시각장애인을 위한 신호가 1초에 한 번씩 출력된다.
음향신호기가 켜져 있고 신호등이 빨간불이면, 시각장애인을 위한 경고가 1초에 한 번씩 출력된다.
```

테스트 설계

이러한 요구사항을 만족하는 테스트를 설계하려고 합니다.

주기 함수와 주기 단위 설계

`tick()` 함수에서 신호등의 `timer`가 `50`이고, 요구사항에서 각각의 등이 5초씩 켜진다고 하였으므로, 주기 단위는 ms이고 `tick()`의 주기 시간은 100ms입니다.
음향신호기가 1초에 한 번씩 신호를 출력하므로 `alarmForBlind()`의 주기 시간은 1000ms(1초)입니다.

초기화 함수 설계

`initial()` 함수를 초기화 함수로 호출하여 각 센서와 상태에 초깃값을 할당합니다. 이때, `trafficLight`의 초깃값은 `GREEN`이고 `soundSystem`의 초깃값은 `OFF`입니다.

테스트 케이스 및 변수 설계

요구사항을 확인하기 위하여 음향신호기를 켜고 파란불, 빨간불일 때 신호를 확인합니다. 이에 맞게 테스트 케이스를 다음과 같이 설계하겠습니다.

TC	시간	음향신호기
1	5000	OFF
2	5000	OFF
3	5000	ON
4	5000	ON
5	5000	OFF
6	5000	OFF

음향신호기의 값을 바꿔주기 위해 사용자 코드에 다음과 같이 입력합니다. 테스트 케이스가 3일 때, 음향신호기를 켜고 테스트 케이스가 5일 때, 음향신호기를 끕니다.

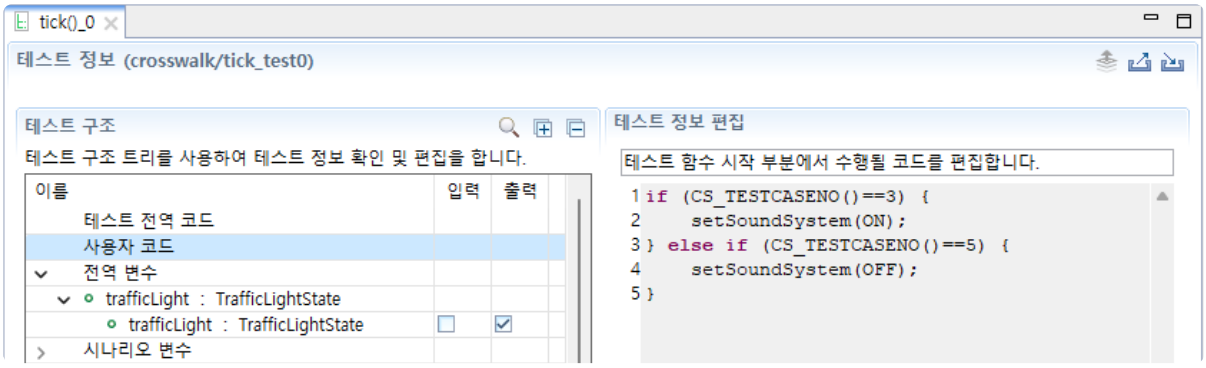
```
if (CS_TESTCASENO()==3) {
        setSoundSystem(ON);
} else if (CS_TESTCASENO()==5) {
        setSoundSystem(OFF);
}
```

전역 변수에서 `trafficLight`를 선택하고 [출력]에 체크하여 신호등의 상태를 확인합니다.

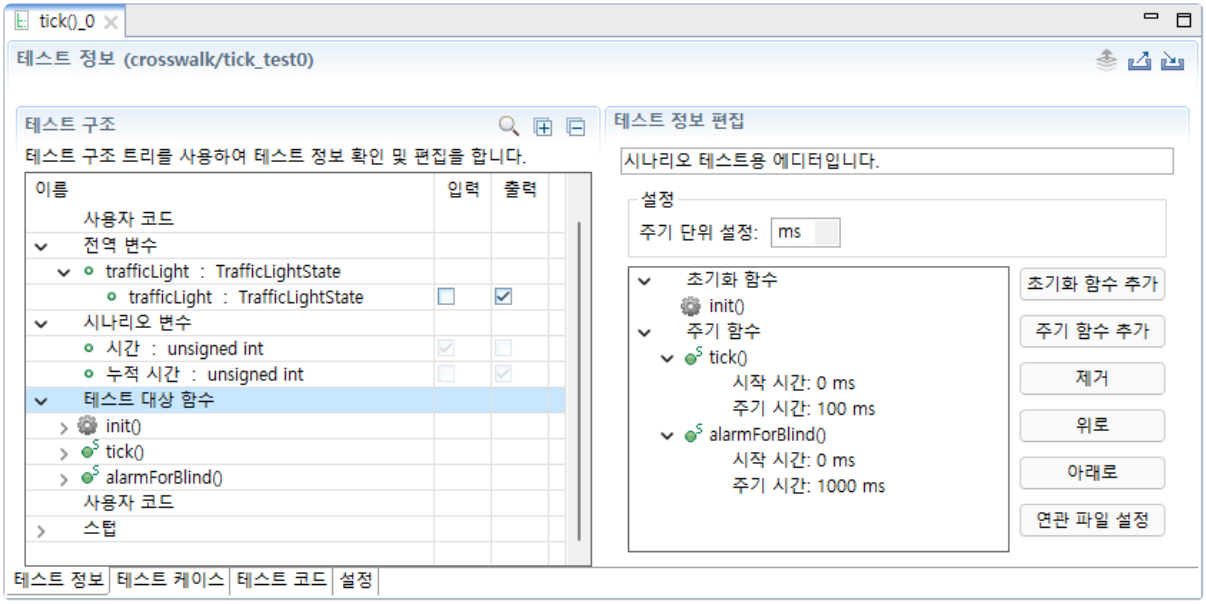
테스트 작성

위 설계를 토대로 테스트를 작성하면 다음과 같습니다.

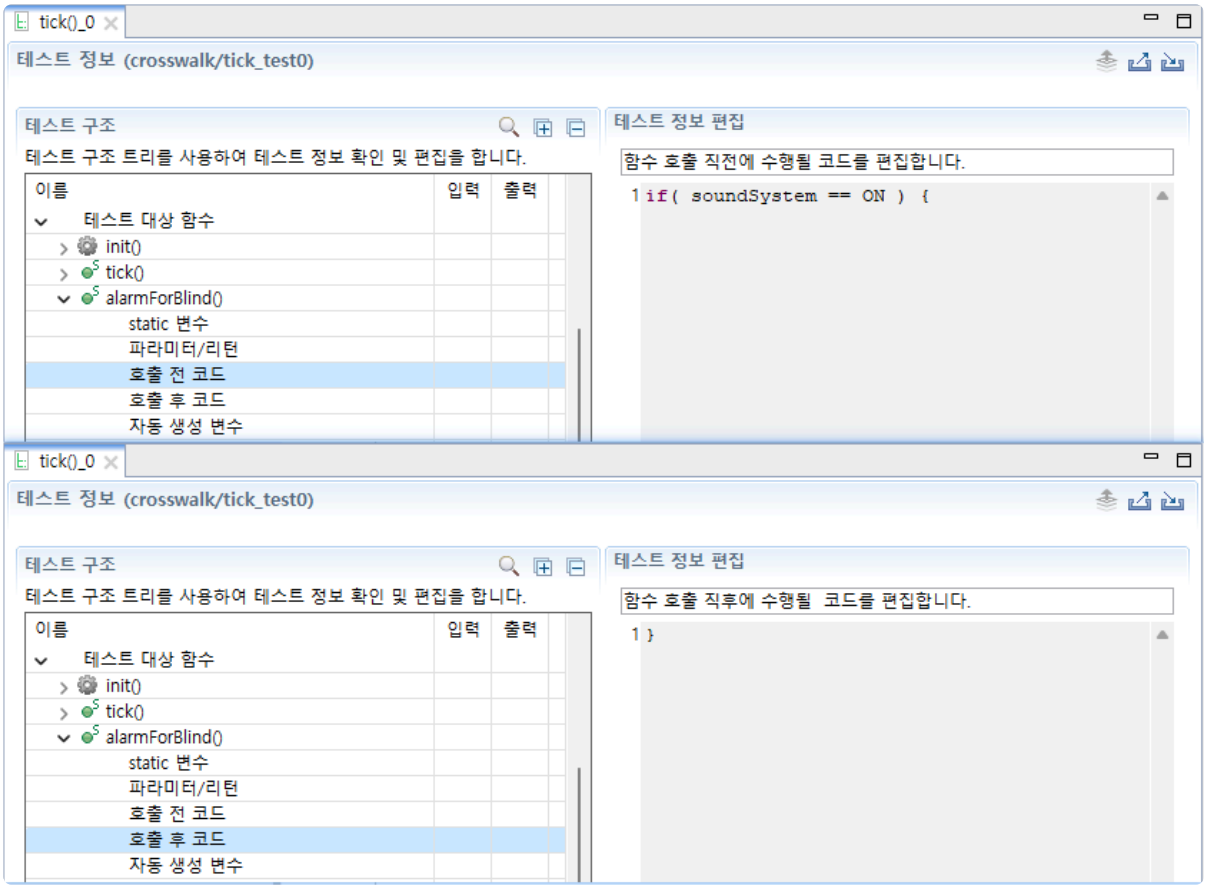
- [테스트 정보] > [사용자 코드]



- [테스트 정보] > [테스트 대상 함수]



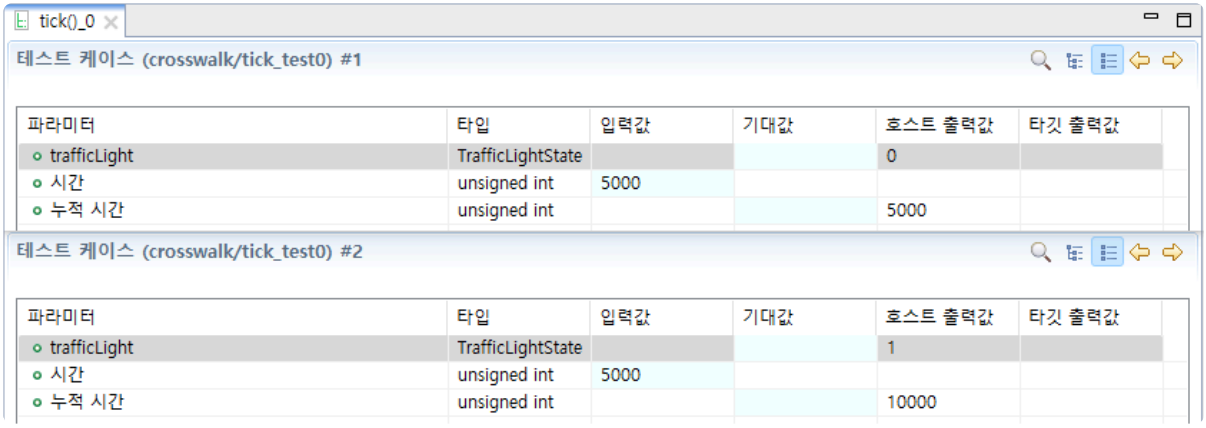
- [테스트 정보] > [호출 전/후 코드]



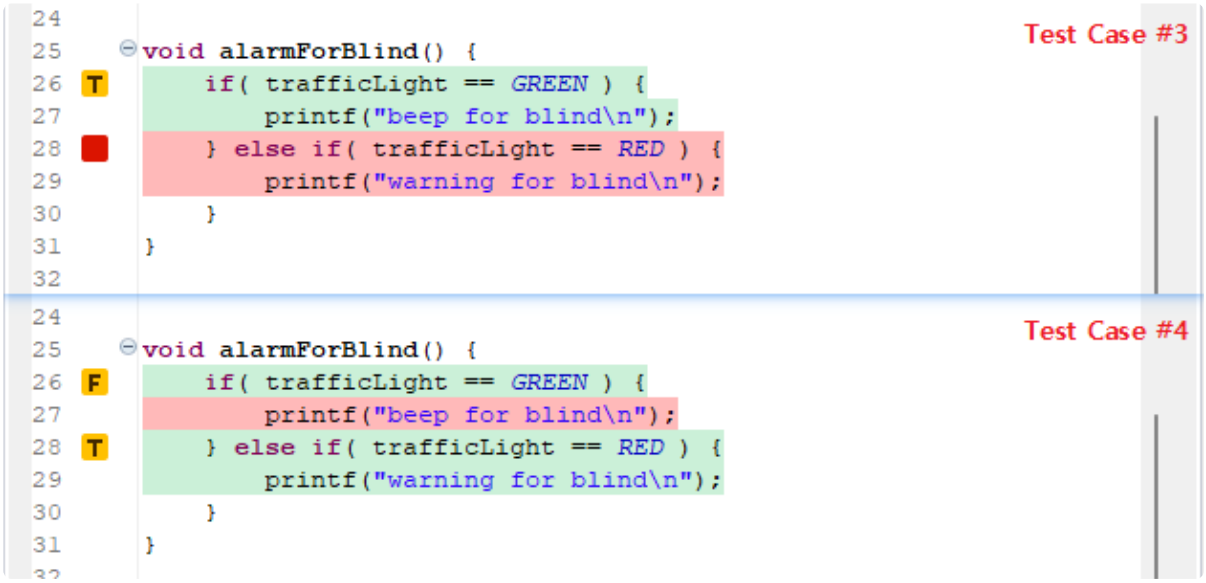
테스트 결과 확인

테스트를 실행하고 [테스트 케이스] 탭과 [소스 코드 편집기]에서 결과를 확인합니다.

- [테스트 케이스] 탭에서 5초 단위로 `trafficLight`가 바뀌는 것을 확인합니다.



- [소스 코드 편집기]에 표시되는 커버리지를 통해 테스트 케이스 #3에서 신호가 울리고 테스트 케이스 #4에서 경고가 울린 것을 확인합니다.



4. C++ 테스트 가이드

CT 2023.12를 사용하여 C++를 테스트하는 방법은 아래와 같습니다.

- 클래스 팩토리 뷰를 이용한 C++ 테스트 가이드

4.1. 클래스 팩토리 뷰를 이용한 C++ 테스트 가이드

클래스 팩토리의 사용 목적

C++ 소스 코드를 테스트할 때, 추상 클래스는 객체 생성을 할 수 없어 테스트가 어렵습니다. 클래스 팩토리를 이용하면 추상 클래스에 대한 테스트를 용이하게 할 수 있고 클래스 객체를 설계할 때 발생하는 반복 작업도 줄일 수 있습니다.

클래스 팩토리의 주요 기능

- 추상 클래스를 상속받는 콘크리트 클래스를 자동으로 생성
- 테스트에 일괄적으로 적용하여 반복 작업을 최소화

클래스 팩토리 활용

이 문서에서는 클래스 팩토리를 이용하기 전에 C++ 테스트를 하기 위한 기본 개념을 설명합니다. 그 후, 클래스 팩토리를 활용하는 방법을 설명합니다.

4.1.1. C++ 테스트를 하기 위한 기본 개념

[클래스 팩토리 뷰]를 이용하여 C++ 테스트를 하기 전에 필요한 기본 개념을 간략하게 설명합니다.

순수 가상 함수와 추상 클래스

순수 가상 함수

- 선언은 있고 정의가 없는 가상 함수
- `= 0`으로 표시
- derived class에 구현하는 가상 함수

추상 클래스

- 순수 가상 함수를 멤버로 갖고 있는 클래스
- 추상 클래스는 객체를 생성할 수 없음
  - 포인터형이나 참조형으로 변수 선언
    - ex. `AbstractClass * class1;`
  - 상속을 받아서 객체를 만들어야 함
- 객체 지향 프로그래밍의 다형성을 지원
- 추상 클래스를 상속받는 함수에서 순수 가상 함수를 재정의해야 함
  - 추상 클래스를 상속받는 derived class에서 순수 가상 함수를 재정의하지 않으면 derived class 도 추상 클래스가 됨

```
class Abstract {
        virtual void f() = 0; // pure virtual
}; // "Abstract" is abstract

class Concrete : Abstract {
        void f() override {} // non-pure virtual
        virtual void g();     // non-pure virtual
}; // "Concrete" is non-abstract

class Abstract2 : Concrete {
        void g() override = 0; // pure virtual overrider
}; // "Abstract2" is abstract

int main()
{
        // Abstract a; // Error: abstract class
        Concrete b; // OK
        Abstract& a = b; // OK to reference abstract base
        a.f(); // virtual dispatch to Concrete::f()
        // Abstract2 a2; // Error: abstract class (final overrider of g() is p
ure)
```

```
}
```

4.1.2. 추상 클래스의 객체 생성 코드를 테스트에 활용

소스 코드를 분석할 때, 추상 클래스의 객체를 만들 수 있도록 추상 클래스를 상속받는 콘크리트 클래스의 객체 생성 코드가 클래스 팩토리에서 자동으로 생성됩니다. 추상 클래스의 객체 생성 코드에서는 사용자가 쉽게 콘크리트 클래스를 만들 수 있도록 콘크리트 클래스에 대한 틀이 제공됩니다.

테스트를 생성할 때, 소스 코드에 해당 추상 클래스를 상속받는 콘크리트 클래스가 존재하면 해당 클래스가 테스트와 연결되고 콘크리트 클래스가 존재하지 않으면 클래스 팩토리의 객체 생성 코드가 연결됩니다.

객체 생성 코드를 추가하여 다양한 형식의 추상 클래스를 테스트에 적용할 수 있습니다.

4.1.3. 클래스 팩토리를 사용한 C++ 테스트 설계

Controller Tester 3.5부터 추상 클래스 뿐만 아니라 대부분의 클래스에 대해 클래스 팩토리를 사용할 수 있습니다.

Controller Tester 3.5 클래스 팩토리의 장점

클래스 팩토리를 이용하여 단순 반복 작업을 줄일 수 있습니다.

- 외부 데이터를 가져오는 클래스 객체
  - ex. 데이터 베이스, 외부 입출력, 서버 등
- [테스트 편집기]에서 복잡하게 설계해야 하는 클래스 객체이지만 여러 테스트에 동일하게 사용해야 하는 경우

클래스 팩토리를 이용하여 객체를 생성하고 적용하는 방법

1. [클래스 팩토리 뷰]의 클래스를 우클릭하여 [생성]을 이용해 클래스 객체 생성 코드를 만든다
2. 클래스 객체 생성 코드를 테스트 설계에 맞게 수정한다.
3. 클래스 객체 생성 코드를 테스트에 적용한다.
   - 일괄 적용
   - 개별 적용

4.1.4. C++ 테스트에서 모의 객체 사용하기

모의 객체의 사용 목적

C++ 소스 코드를 테스트할 때, 실제 객체를 만드는 데에 비용이 많이 들거나 테스트에서 객체에 대한 의존성이 높아 테스트하기 어려운 경우가 있습니다. 모의 객체를 사용하면 효율적으로 객체에 대한 의존성을 대체할 수 있습니다. 추가로 모의 객체의 예상 호출 횟수 등을 정의하여 객체를 의도한대로 사용하는지 확인할 수 있습니다.

사용 가능한 툴체인

- GCC 6.0 이상
- Visual Studio 2015 이상

모의 객체 명세

- 모의 객체의 매개변수와 반환값 지정
- 모의 객체의 예상 호출 횟수 설정
- 호출 순서 확인
- 매개 변수에 제약 사항 추가
- 그 외

모의 객체 활용

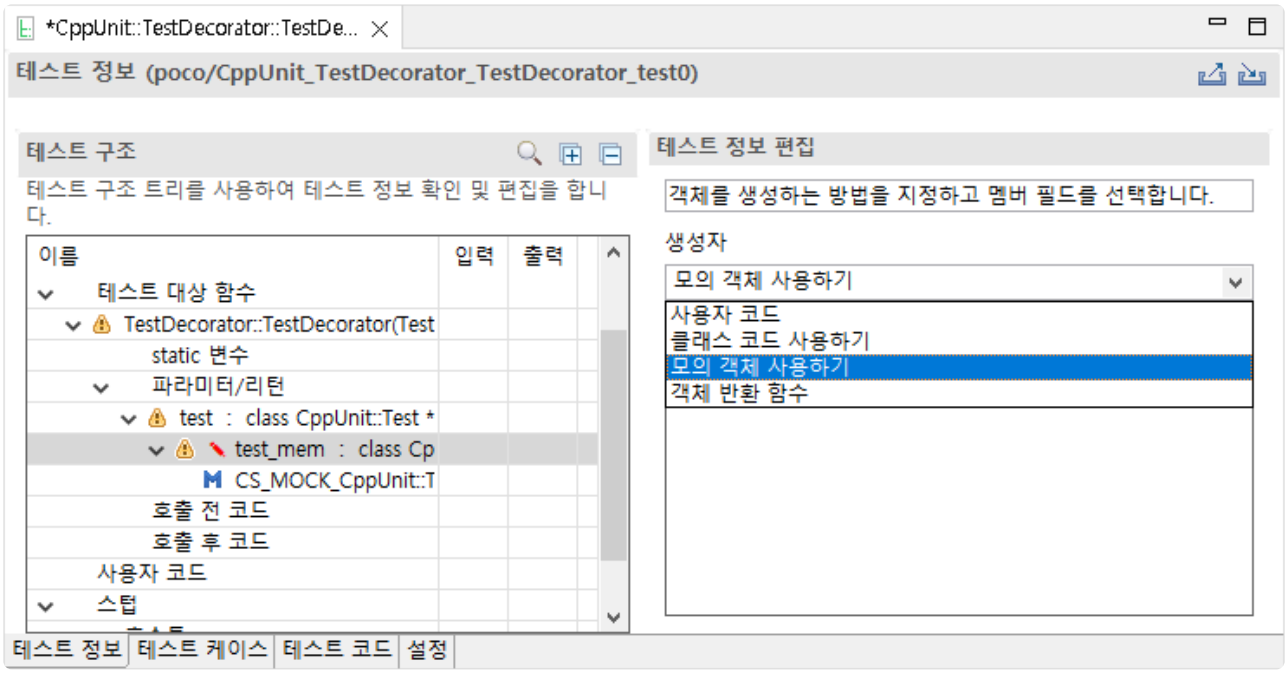
이 문서에서는 C++ 테스트에서 모의 객체를 활용하는 방법을 설명합니다.

- 모의 객체 생성하기
- 모의 객체 명세 추가하기

4.1.4.1. 모의 객체 생성하기

모의 객체 생성

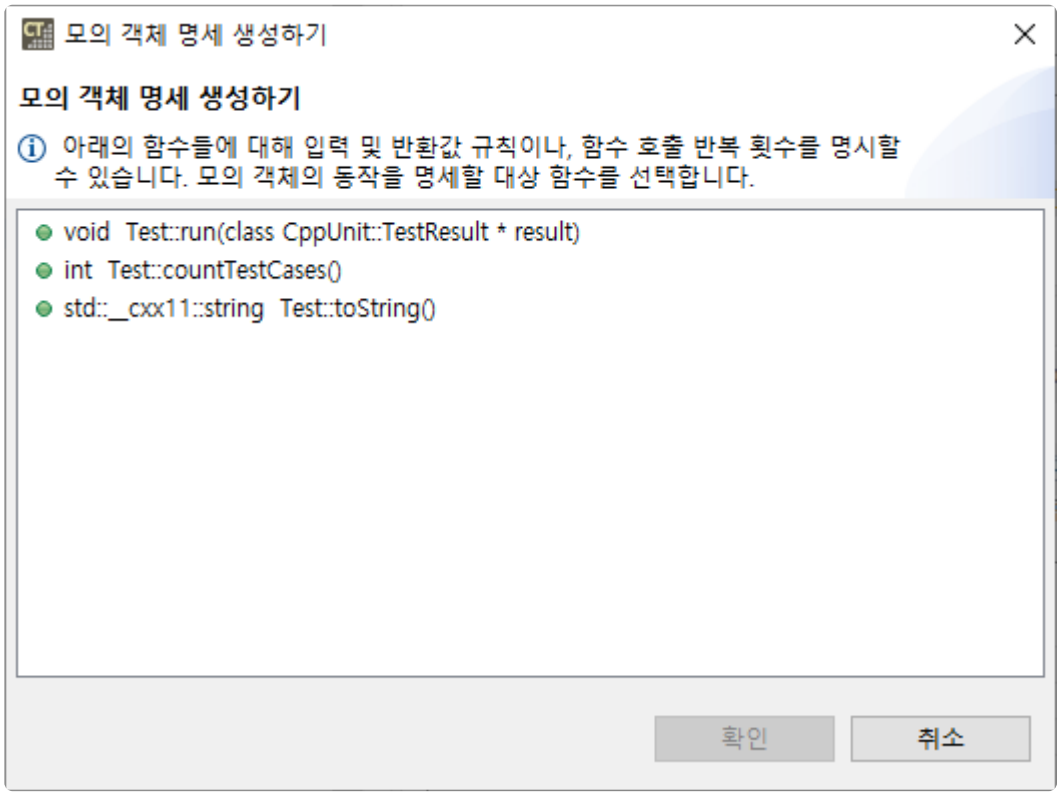
1. 모의 객체를 생성할 테스트를 더블 클릭하여 [테스트 편집기]를 엽니다.
2. [테스트 정보 탭]에서 테스트 구조 트리를 펼쳐 모의 객체를 생성할 객체를 선택합니다.
3. 오른쪽 테스트 정보 편집 영역의 생성자에서 [모의 객체 사용하기]를 선택합니다.



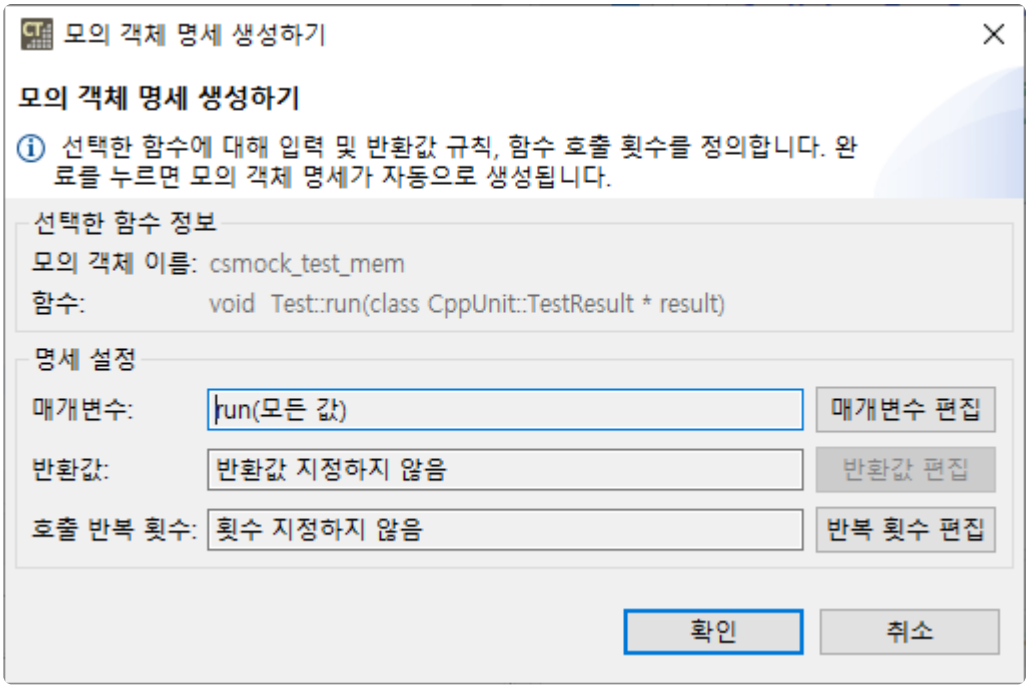
4.1.4.2. 모의 객체 명세 생성하기

모의 객체 명세를 자동으로 생성하기

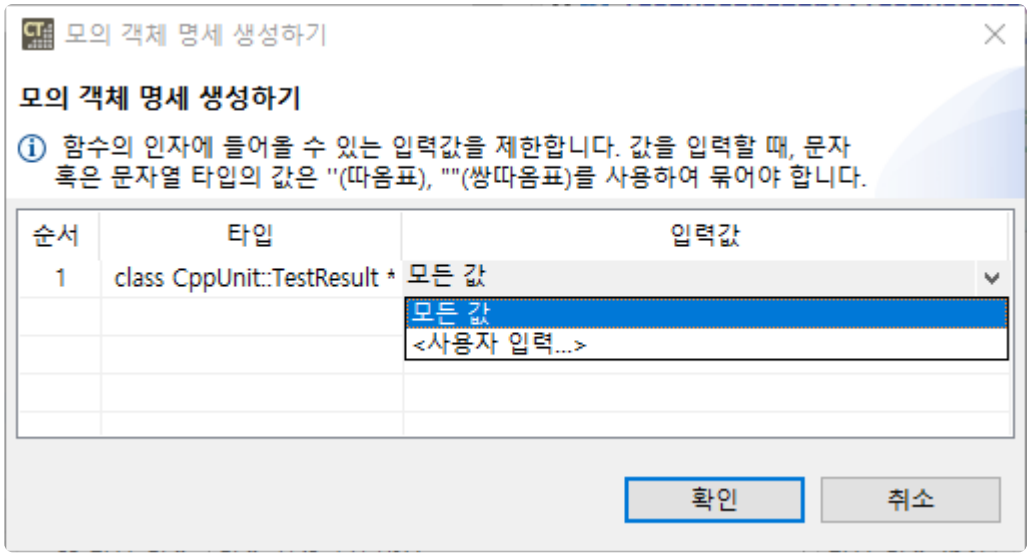
1. [테스트 정보 탭]에서 생성된 모의 객체를 클릭합니다.
2. 오른쪽 테스트 정보 편집 영역에서 [모의 객체 명세 생성 마법사...] 버튼을 클릭합니다.
    - 모의 객체의 명세가 비어있으면 모의 객체를 클릭할 때 [모의 객체 명세 생성하기] 마법사를 자동으로 띄워줍니다.



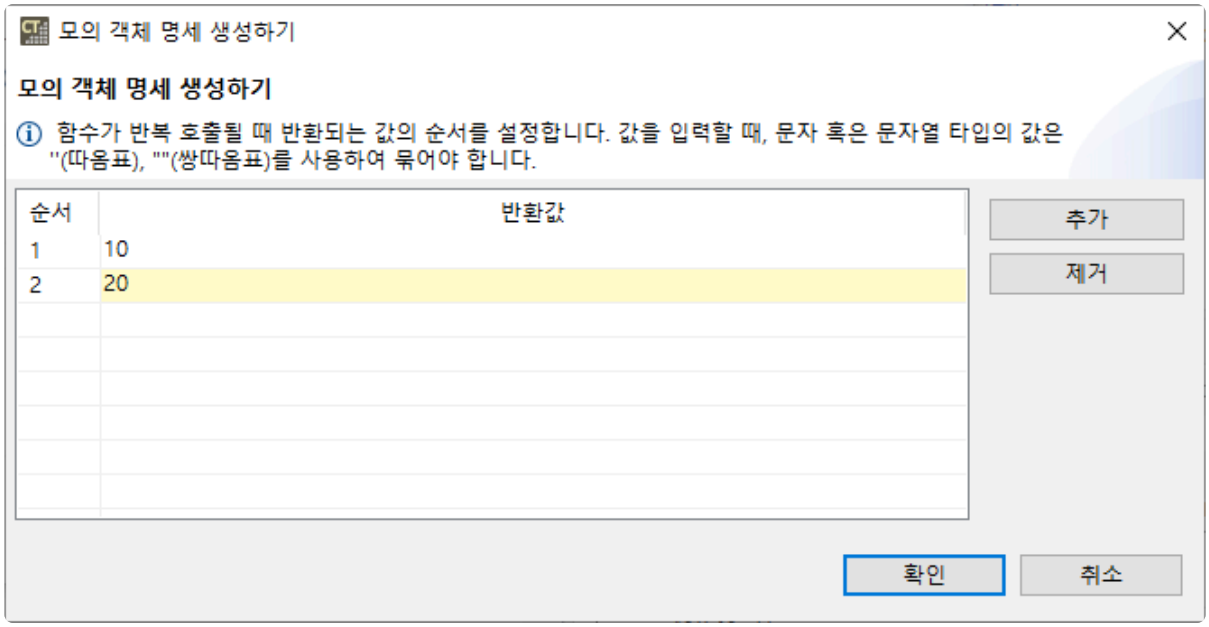
3. [모의 객체 명세 생성하기] 마법사에서 대상 함수를 선택하고 [확인] 버튼을 클릭합니다.
4. 매개변수, 반환값, 호출 반복 횟수를 편집합니다.



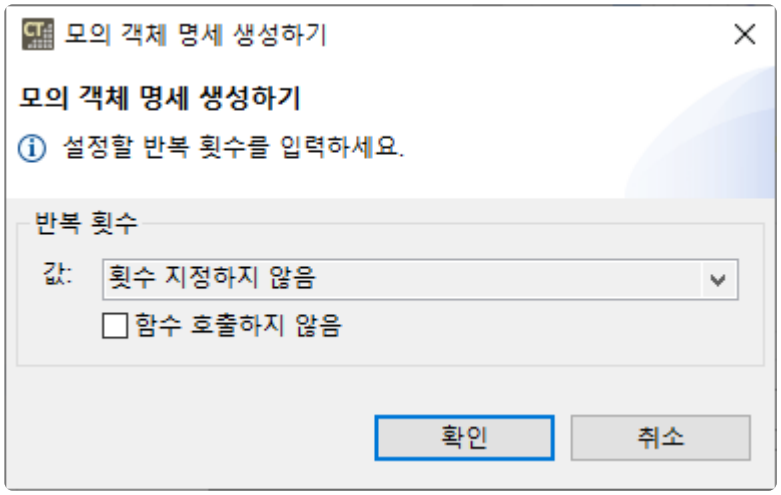
- [매개변수 편집] 버튼을 클릭하여 해당 함수에서 사용하는 매개변수에 대한 명세를 작성할 수 있습니다.



  - [모든 값]을 선택하면 해당 매개변수의 값을 제한하지 않습니다.
  - [<사용자 입력...>]을 통해 매개변수 값을 제한할 수 있습니다. 예를 들어, 입력값에 `1`을 입력한 후 테스트를 실행할 경우 해당 매개변수가 `1`이 아니면 테스트가 실패합니다.
- [반환값 편집] 버튼을 클릭하여 해당 함수의 반환값을 결정할 수 있습니다.



  - [추가]를 선택하여 함수가 호출될 때 반환할 값을 추가합니다.
  - [제거]를 선택하여 마지막에 추가한 반환값을 제거합니다.
  - 반환값을 하나만 설정하면 해당 반환값을 반복하여 반환합니다.
  - 반환값을 여러 개 설정하면 함수가 호출될 때, 순서대로 해당 값을 반환합니다. 이 경우, 해당 반환값의 개수만큼 함수가 호출되지 않으면 테스트가 실패합니다.
- [반복 횟수 편집] 버튼을 클릭하여 해당 함수 호출 횟수에 대한 명세를 작성할 수 있습니다.



- [횟수 지정하지 않음]을 선택하면 호출 횟수를 제한하지 않습니다.
- [<사용자 입력...>]을 통해 함수 호출 횟수를 제한할 수 있습니다. 예를 들어, 함수 호출 횟수를 3으로 설정하고 테스트를 실행할 때, 해당 함수가 세 번 호출되지 않으면 테스트가 실패합니다.
- [함수 호출하지 않음]은 호출 횟수를 0으로 지정하는 것과 동일합니다. 이 경우, 해당 함수가 호출되면 테스트가 실패합니다.

> ! 반환값과 호출 반복 횟수를 동시에 설정하려면 [모의 객체 명세 생성하기] 마법사로 생성한 명세를 참고하여 [테스트 편집기]에서 직접 작성해야 합니다.

5. [확인] 버튼을 클릭하여 명세를 만듭니다.

모의 객체 명세를 직접 작성하기

Controller Tester에서 [모의 객체 명세 생성 마법사...]로 생성한 명세를 수정하거나 다양한 명세를 직접 작성할 수 있습니다. 자세한 내용은 이 문서를 참고하시기 바랍니다.

5. CI/CD 환경 및 CLI 가이드

CI/CD 환경이나 CLI를 사용하여 테스트하는 방법은 다음과 같습니다.

- CT Jenkins plugin 사용 가이드
- CLI 사용 가이드

5.1. CT Jenkins plugin 사용 가이드

CT Jenkins plugin은 CT 2023.12 프로젝트를 지속적 통합 및 지속적 배포(CI/CD)하기 위한 확장 기능입니다. CT 2023.12의 테스트를 자동화하여 팀 또는 조직의 개발 프로세스를 더욱 효율적으로 관리할 수 있습니다.

요구 사항

1. CT

2023.12 버전 이상의 CT를 설치해야 합니다.

2. Jenkins

젠킨스 설치 방법은 젠킨스 문서를 참조하세요. (Installing Jenkins)

3. CT Jenkins plugin

- hpi 파일로 설치
    1. Jenkins 관리 > Plugins > Advanced settings의 Deploy plugin 항목에서 ct-jenkins-plugin.hpi 파일을 선택하고 deploy합니다.
    2. 설치가 완료되면 빌드 환경에서 CT environment 항목이, Build Steps에서 CT test execution과 CT custom command 항목이 추가된 것을 확인할 수 있습니다.

빌드 환경 설정

[Jenkin 관리] > [System] > [CT (Controller Tester)]에서 CT 실행 환경을 설정합니다.

CT (Controller Tester)

CT 설치 경로

(예: C:₩Program Files₩Suresoft₩CT 2023)

C:₩Program Files₩Suresoft₩CT 2023

팀 테스팅 서버 IP

팀 테스팅 서버 Port

CT 라이선스

- ○ Node-locked
- ● Floating

  서버 운영체제

  - ● Windows
  - ○ Linux

  IP

  Port

5.1.1. 프리스타일 프로젝트 (Single Job) 생성하기

프리스타일 프로젝트에서는 빌드 환경 및 빌드 스텝을 추가해서 프로젝트를 구성할 수 있습니다.
빌드 스텝에는 CT test execution, CT custom command 두 가지 옵션이 있으며, 둘 중 하나의 빌드 스텝만 사용하여 프로젝트를 구성하는 것을 권장합니다.

빌드 스텝 – CT test execution

프로젝트 설정

CT Jenkins plugin으로 테스트할 프로젝트를 선택합니다.

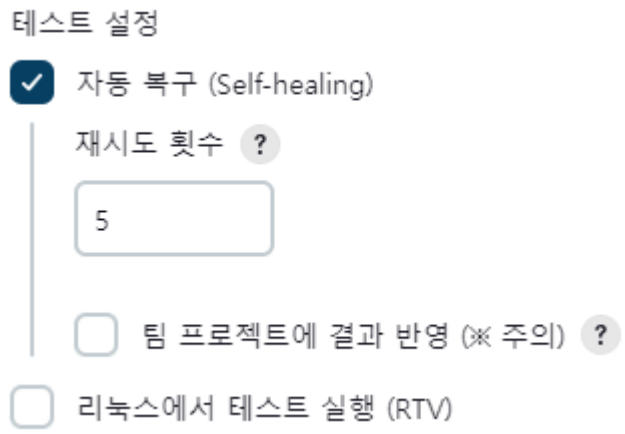
- 일반 프로젝트
  - 일반 프로젝트에 CT에서 내보낸 프로젝트의 경로를 입력합니다.
  - 프로젝트를 내보낼 때는 소스 코드와 툴체인을 포함해서 내보내야 합니다.
- CT 팀 프로젝트
  - CT 팀 프로젝트에 팀 테스팅 서버에 있는 팀 프로젝트 이름을 입력합니다.
  - 대상이 되는 팀 프로젝트는 분석된 상태여야 합니다.

소스 코드 설정

Git 저장소의 특정 branch로 리그레션 테스트할 때 소스 코드 설정을 사용합니다.
소스 코드 설정을 사용하면 Git 저장소의 소스 코드 최상위 경로를 기준으로, 위에서 선택한 프로젝트의 소스 코드와 동기화합니다.
이 설정으로 변경되는 소스 코드에 대한 CT 테스트 결과를 CI/CD 환경에서 확인할 수 있습니다.
옵션을 선택하지 않으면 사용자가 설정한 프로젝트의 소스 코드 형상으로 테스트합니다.

테스트 설정

테스트 실행 옵션을 설정합니다.



- 자동 복구 (Self-Healing)
  CT Jenkins plugin이 자동으로 무결성 검사 후, 재설정 후보를 선택해 테스트합니다.
  자동 복구는 모든 테스트가 성공할 때까지 또는 재시도 횟수까지 실행됩니다.

- 리눅스 환경에서 테스트
  RTV 테스트가 필요한 프로젝트인 경우 선택합니다.

보고서 설정

결과 보고서를 생성할 포맷을 선택합니다.

빌드 스텝 – CT custom command

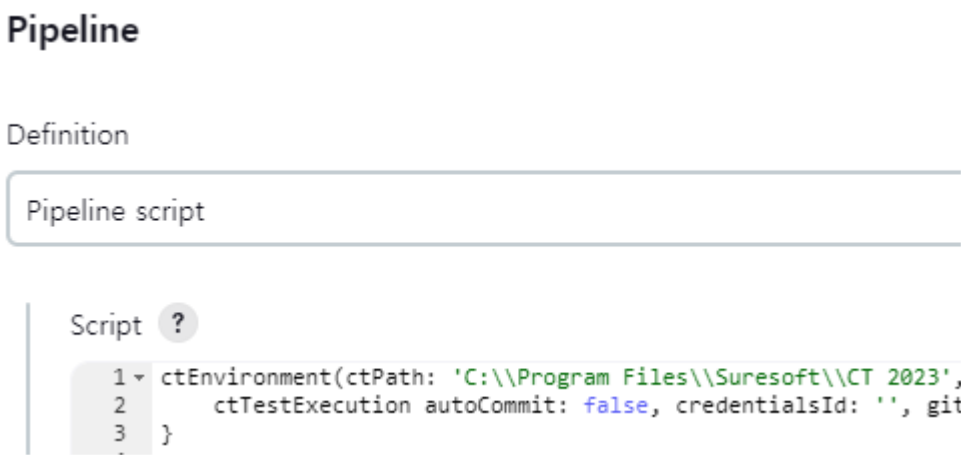
CT test execution 빌드 스텝을 사용하지 않고, 워크스페이스 및 CT CLI 설정을 사용자가 설정할 수 있는 빌드 스텝입니다.
상세 사용 방법 및 이슈는 매뉴얼의 문제 해결 페이지 하단에 있는 기술지원 연락처를 통해 문의해 주시기 바랍니다.

5.1.2. 파이프라인 프로젝트 생성하기

파이프라인 프로젝트에서는 Pipeline script를 작성하여 프로젝트를 구성할 수 있습니다.

빌드 스크립트 설정

Pipeline Syntax에서 Snippet Generator를 이용해서 파이프라인 스크립트를 작성하는 것을 권장합니다. Snippet Generator에서 에서 빌드 환경 설정 스텝(ctEnvironment)과 CT 테스트 수행 스텝(ctTestExecution)을 선택해 스크립트를 생성할 수 있으며, 반드시 ctTestExecution 스텝이 ctEnvironment 스텝 내부에 포함되어야 합니다.

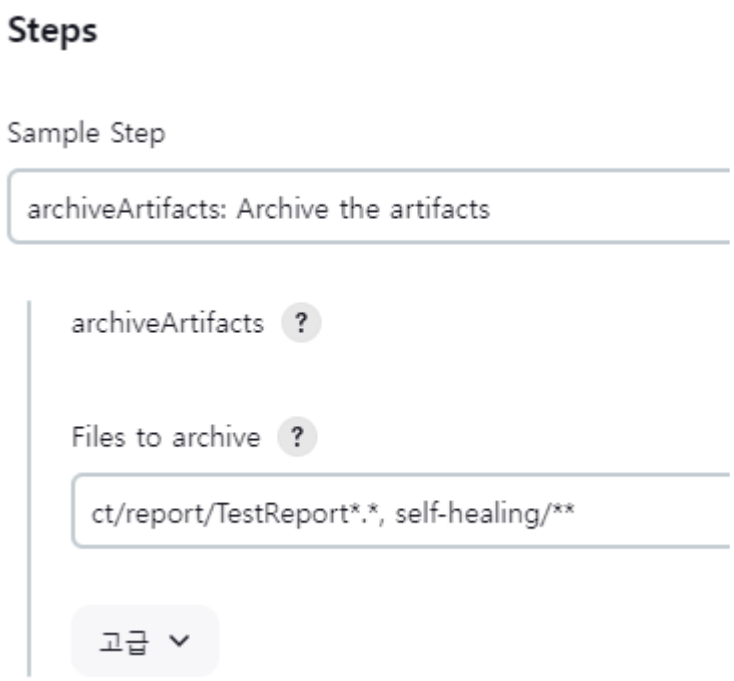


빌드 후 조치 설정

파이프라인 프로젝트의 경우 빌드 후 조치를 자동으로 해주지 않기 때문에, 별도로 추가해주어야 합니다. 빌드 후 조치 역시 Snippet Generator를 이용해 생성할 수 있으며, 추가할 수 있는 항목은 아래와 같습니다.

결과물 저장

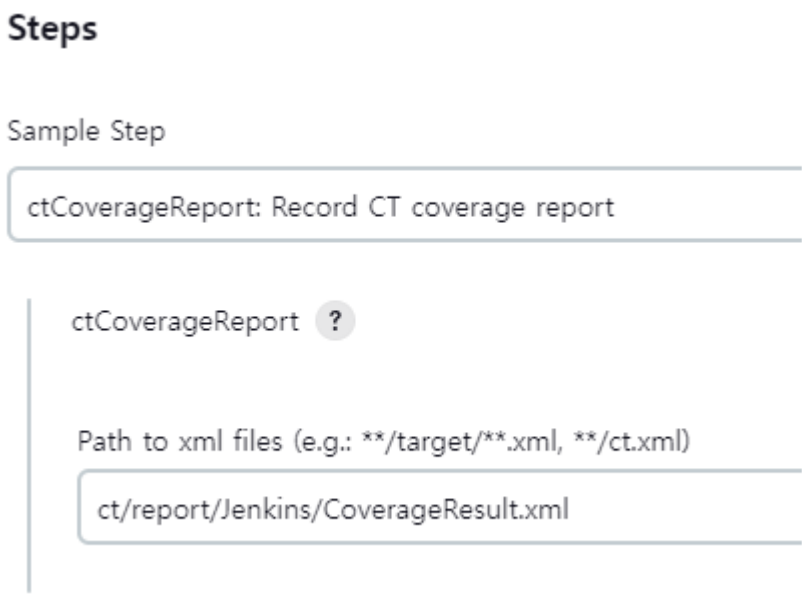
Sample Step에서 archiveArtifacts를 선택하고 다음과 같이 입력합니다.



- Files to archive: ct/report/TestReport*.*, self-healing/**

커버리지 결과 확인

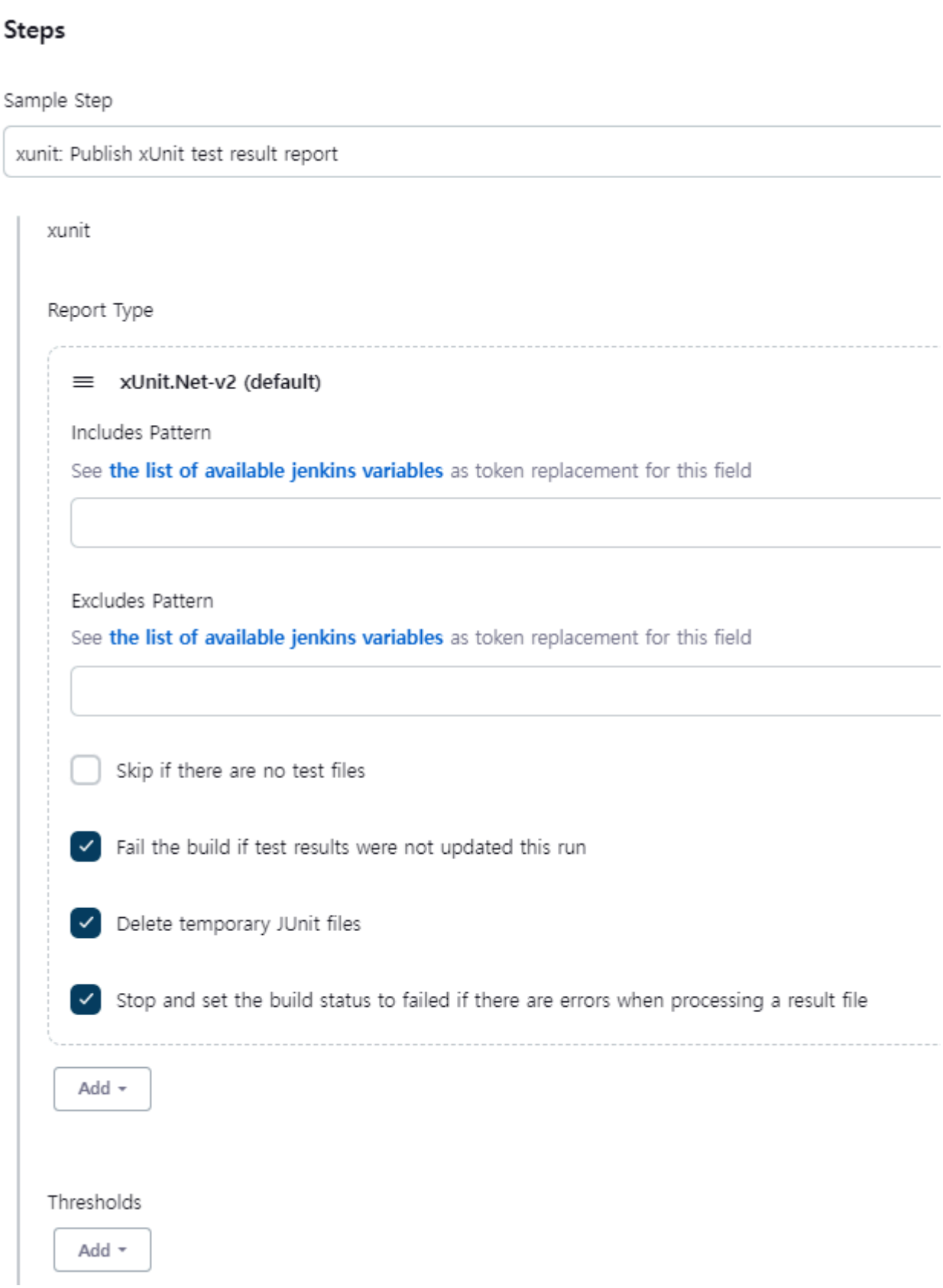
Sample Step에서 ctCoverageReport를 선택하고 다음과 같이 입력합니다.



- Path to xml files: ct/report/Jenkins/CoverageResult.xml

테스트 결과 확인

Sample Step에서 xUnit.Net-v2를 선택하고 다음과 같이 입력합니다.



- Report Type: xUnit.Net-v2 (default)
- Includes Pattern: ct/report/Jenkins/TestResult.xml

스크립트 예시

위 내용을 기반으로 구성된 전체 스크립트 예시는 다음과 같습니다.

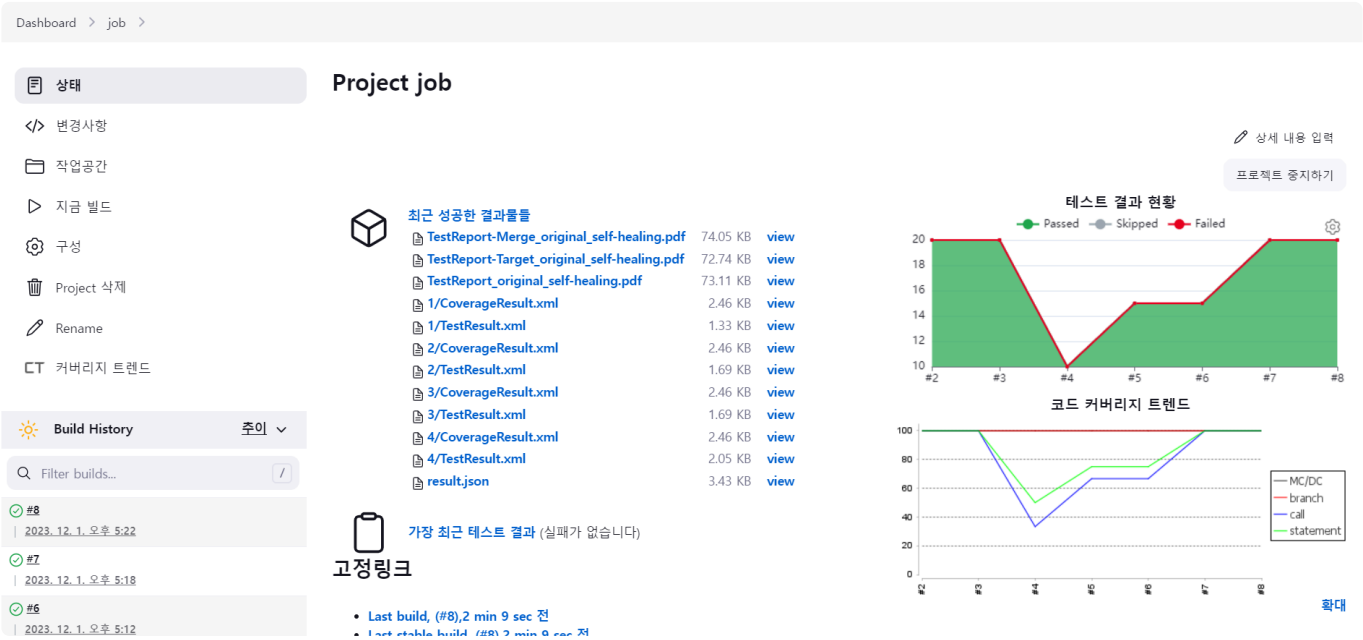
```
pipeline {
    agent any

    stages {
        stage('Test') {
            steps {
                ctEnvironment(ctPath: 'C:\\Users\\CodeScroll\\CodeScroll', licenseOption: 'FLOATING', port: '8080', serverIp: '10.10.10.10', serverOs: '
                    ctTestExecution autoCommit: false, credentialsId: 'id', gitSourceBranch: 'second', gitSourcePath: 'git@github.com', gitSourceRootPath
                }
            }
        }

        stage('Publish') {
            steps {
                archiveArtifacts artifacts: 'ct/report/TestReport_*.*, self-healing/**', followSymlinks: false
                ctCoverageReport execPattern: 'ct/report/Jenkins/CoverageResult.xml'
                xunit checksName: '', tools: [xUnitDotNet(excludesPattern: '', pattern: 'ct/report/Jenkins/TestResult.xml', stopProcessingIfError: true)]
```

5.1.3. 결과 확인

CT Jenkins plugin 프로젝트 수행 후 결과를 확인하는 방법에 대해 설명합니다.

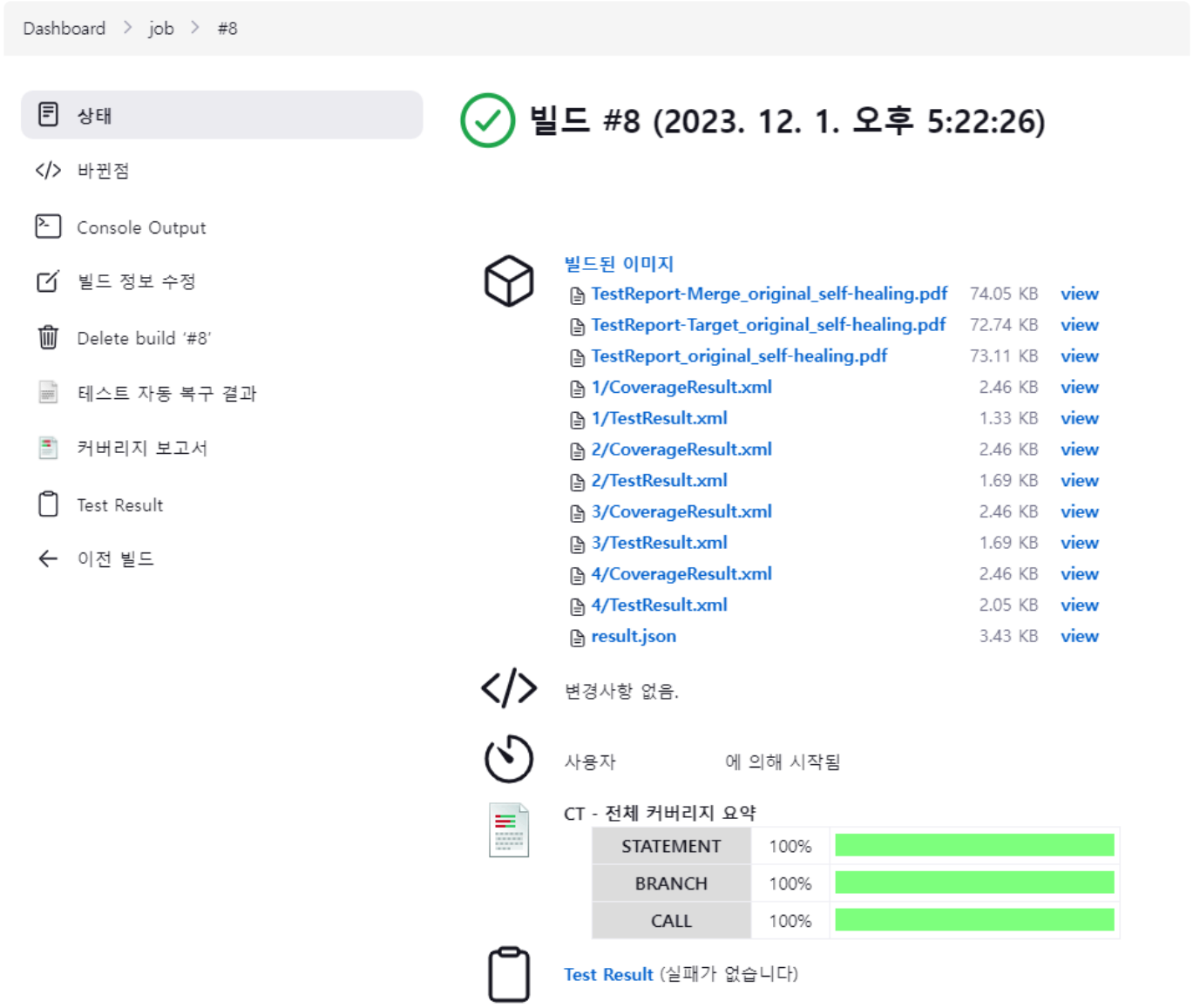
프로젝트 메인 화면

메인 화면 오른쪽에서 테스트 결과 및 코드 커버리지 트렌드를 확인할 수 있으며,
최근 성공한 결과물인 테스트 보고서와 자동 복구(Self-healing)의 회차별 결과 파일을 확인할 수 있습니다.



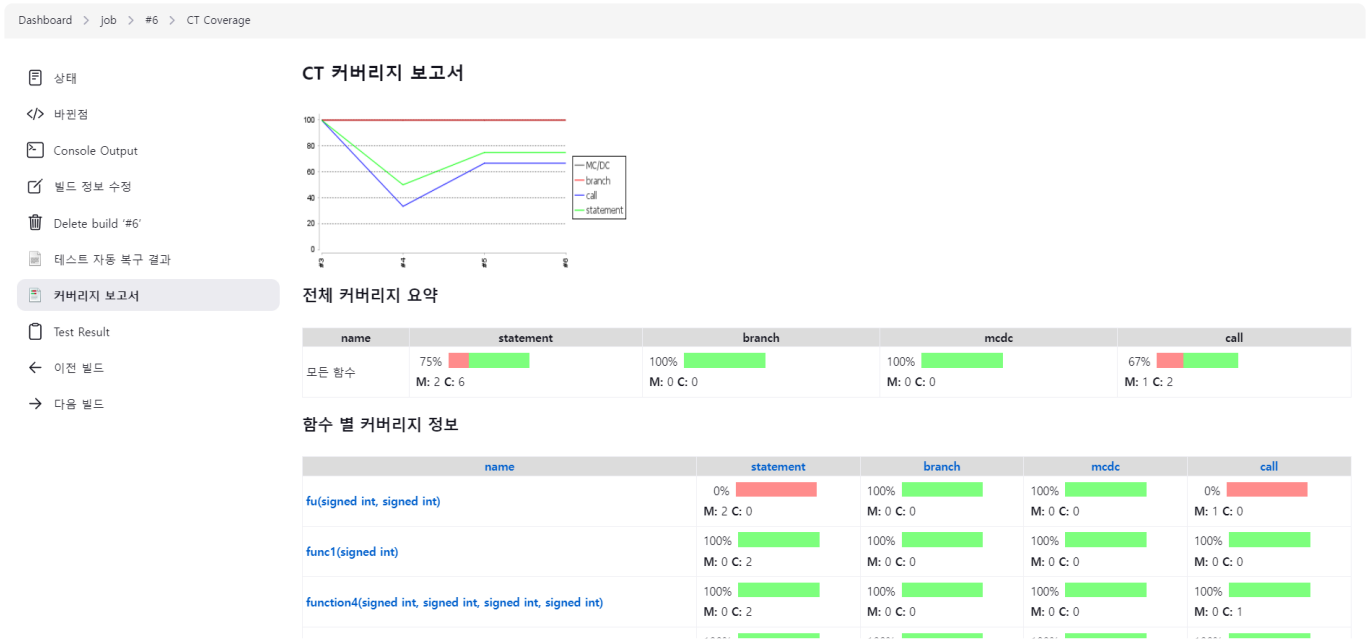
빌드 상세 정보

빌드 히스토리에서 특정 빌드를 클릭하면, 해당 빌드에 대한 상세 정보를 확인할 수 있습니다.
해당 회차에서 수집된 결과물, 커버리지 요약 정보 및 테스트 결과를 확인할 수 있습니다.



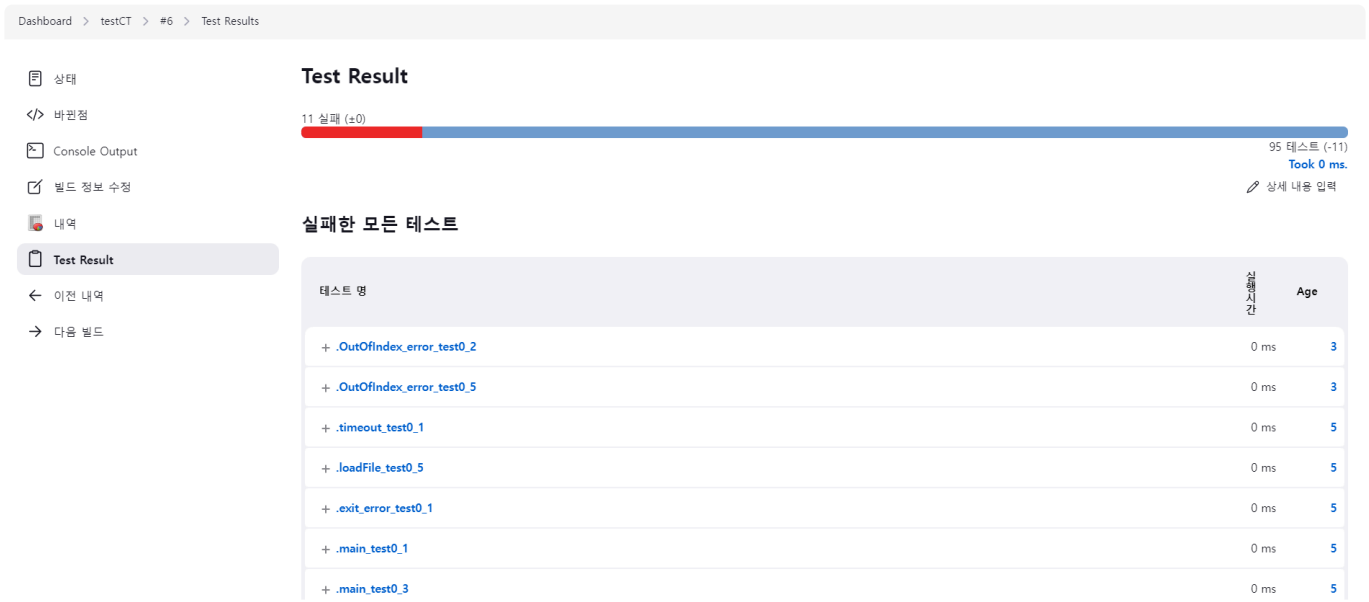
커버리지 보고서

왼쪽 사이드바의 커버리지 보고서 항목을 클릭하면 커버리지 추이 그래프를 더 상세히 확인할 수 있습니다.



테스트 결과

왼쪽 사이드바의 Test Result 항목을 클릭하면 테스트 결과를 더 상세히 확인할 수 있습니다.



5.2. CLI 사용 가이드

Command Line Interface를 사용하여 CT 2023.12 기능들을 사용하는 시나리오를 설명합니다.

- CLI 프로젝트 경로 재설정

5.2.1. CLI 프로젝트 경로 재설정

내보낼 때 소스코드 경로와 가져올 때 소스코드 경로가 다른 경우 아래와 같이 에러가 발생합니다.

```
C:₩Program Files₩Suresoft₩CT 2023.6>csc.exe -e -w "C:₩Users₩sure₩Documents₩cli-workspace" --import -O "--path 'C:₩Users₩sure₩Desktop₩cliExportPath₩zli
b_20230522080714' --include-tch
[MAIN-INFO] 명령어 인자를 분석 중...
[MAIN-INFO] 워크스페이스 설정
[MAIN-INFO] Install Location: C:₩Program Files₩Suresoft₩CT 2023.6
[MAIN-INFO] PRODUCT_VERSION: 2023.6
[MAIN-INFO] 워크스페이스: C:₩Users₩sure₩Documents₩cli-workspace
[MAIN-INFO] initialize()
[MAIN-INFO] 전역 정보를 읽었습니다.
[MAIN-INFO] execution job name: Execute Project Import
INFO
============== IMPORT START ==============


INFO load json file...

[ERROR] Invalid file path included. Please check the path map file. : C:₩Users₩sure₩Desktop₩cliExportPath₩zlib_20230522080714₩PathMappingFile.csv

INFO
```

이 경우 CLI에서는 프로젝트 가져오기 시 맵핑 작업을 통해 경로를 새롭게 지정할 수 있습니다. 맵핑 방법은 아래와 같습니다.

1. 프로젝트를 내보내면 해당 경로에 PathMappingFile.csv 파일이 생성됩니다.
2. PathMappingFile.csv 파일을 편집합니다.
    - Old Path: 프로젝트를 내보낼 때 작성된 경로입니다.
    - New Path: 가져올 때 사용될 경로입니다.
    - status: 경로의 상태를 나타냅니다.
        - 파일의 경로가 정상인 경우 : “OK”
        - 기존 경로(Old Path)가 유효하지 않은데 새 경로(New Path)가 입력되지 않은 경우 : “The old path is invalid. Please enter a new path.”
        - 입력한 새 경로(New Path)가 존재하지 않는 경로일 경우 : “The new path you entered is not valid. Please check the new path.”
        - 내보내기에 소스 코드가 포함된 경우 : “This project contains source files. Do not enter new path If you import with source files.”

    a. 내보내는 PC와 가져오는 PC의 경로가 동일하다면 [New Path] 열은 비워둡니다.

	A	B	C	D
1		Old Path	New Path	status
2	1	C:₩Users₩sure₩Desktop₩zlib₩22crc32.c		
3	2	C:₩Users₩sure₩Desktop₩zlib₩adler32.c		
4	3	C:₩Users₩sure₩Desktop₩zlib₩compress.c		
5	4	C:₩Users₩sure₩Desktop₩zlib₩deflate.c		
6	5	C:₩Users₩sure₩Desktop₩zlib₩gzclose.c		
7	6	C:₩Users₩sure₩Desktop₩zlib₩gzlib.c		
8	7	C:₩Users₩sure₩Desktop₩zlib₩gzread.c		
9	8	C:₩Users₩sure₩Desktop₩zlib₩gzwrite.c		
10	9	C:₩Users₩sure₩Desktop₩zlib₩infback.c		
11	10	C:₩Users₩sure₩Desktop₩zlib₩inffast.c		
12	11	C:₩Users₩sure₩Desktop₩zlib₩inflate.c		
13	12	C:₩Users₩sure₩Desktop₩zlib₩inftrees.c		
14	13	C:₩Users₩sure₩Desktop₩zlib₩trees.c		
15	14	C:₩Users₩sure₩Desktop₩zlib₩uncompr.c		
16	15	C:₩Users₩sure₩Desktop₩zlib₩zutil.c		
17				
18				

    b. 내보내는 PC와 가져오는 PC의 코드 경로가 다르면, [New Path]에 가져오는 PC에서 사용할 경로를 추가합니다.

	A	B	C	D
1		Old Path	New Path	status
2	1	C:₩Users₩sure₩Desktop₩zlib₩adler32.c	C:₩Users₩sure₩Desktop₩newPath₩adler32.c	The old path is invalid. Please enter a new path.
3	2	C:₩Users₩sure₩Desktop₩zlib₩compress.c	C:₩Users₩sure₩Desktop₩newPath₩adler32.c	The old path is invalid. Please enter a new path.
4	3	C:₩Users₩sure₩Desktop₩zlib₩crc32.c	C:₩Users₩sure₩Desktop₩newPath₩crc32.c	The old path is invalid. Please enter a new path.
5	4	C:₩Users₩sure₩Desktop₩zlib₩deflate.c	C:₩Users₩sure₩Desktop₩newPath₩deflate.c	The old path is invalid. Please enter a new path.
6	5	C:₩Users₩sure₩Desktop₩zlib₩gzclose.c	C:₩Users₩sure₩Desktop₩newPath₩gzclose.c	The old path is invalid. Please enter a new path.
7	6	C:₩Users₩sure₩Desktop₩zlib₩gzlib.c	C:₩Users₩sure₩Desktop₩newPath₩gzlib.c	The old path is invalid. Please enter a new path.
8	7	C:₩Users₩sure₩Desktop₩zlib₩gzread.c	C:₩Users₩sure₩Desktop₩newPath₩gzread.c	The old path is invalid. Please enter a new path.
9	8	C:₩Users₩sure₩Desktop₩zlib₩gzwrite.c	C:₩Users₩sure₩Desktop₩newPath₩gzwrite.c	The old path is invalid. Please enter a new path.
10	9	C:₩Users₩sure₩Desktop₩zlib₩infback.c	C:₩Users₩sure₩Desktop₩newPath₩infback.c	The old path is invalid. Please enter a new path.
11	10	C:₩Users₩sure₩Desktop₩zlib₩inffast.c	C:₩Users₩sure₩Desktop₩newPath₩inffast.c	The old path is invalid. Please enter a new path.
12	11	C:₩Users₩sure₩Desktop₩zlib₩inflate.c	C:₩Users₩sure₩Desktop₩newPath₩inflate.c	The old path is invalid. Please enter a new path.
13	12	C:₩Users₩sure₩Desktop₩zlib₩inftrees.c	C:₩Users₩sure₩Desktop₩newPath₩inftrees.c	The old path is invalid. Please enter a new path.
14	13	C:₩Users₩sure₩Desktop₩zlib₩trees.c	C:₩Users₩sure₩Desktop₩newPath₩trees.c	The old path is invalid. Please enter a new path.
15	14	C:₩Users₩sure₩Desktop₩zlib₩uncompr.c	C:₩Users₩sure₩Desktop₩newPath₩uncompr.c	The old path is invalid. Please enter a new path.
16	15	C:₩Users₩sure₩Desktop₩zlib₩zutil.c	C:₩Users₩sure₩Desktop₩newPath₩zutil.c	The old path is invalid. Please enter a new path.
17				

3. 프로젝트 가져오기 시 `--mapping-file` 옵션을 사용하여 PathMappingFile.csv를 적용합니다.
   - 예시 : `-e -w "%workSpacePath%" --import -O "--path '%프로젝트 경로%' --mapping-file '%PathMappingFile.csv 경로% --include-tch'"`

```
C:₩Program Files₩Suresoft₩CT 2023.6>csc.exe -e -w "C:₩Users₩sure₩Documents₩cli-workspace" --import -O "--path 'C:₩Users₩sure₩Desktop₩cliExportPath₩zl
b_20230522080714' --mapping-file 'C:₩Users₩sure₩Desktop₩cliExportPath₩zlib_20230522080714₩PathMappingFile.csv'  --include-tch"
[MAIN-INFO] 워크스페이스 설정
[MAIN-INFO] Install Location: C:₩Program Files₩Suresoft₩CT 2023.6
[MAIN-INFO] PRODUCT_VERSION: 2023.6
[MAIN-INFO] 워크스페이스: C:₩Users₩sure₩Documents₩cli-workspace
[MAIN-INFO] initialize()
[MAIN-INFO] 전역 정보를 읽었습니다.
[MAIN-INFO] execution job name: Execute Project Import
INFO
```

4. 명령이 실행된 후 경로별 결과는 PathMappingFile.csv 파일에서 확인할 수 있습니다.

	A	B	C	D
1		Old Path	New Path	status
2	1	C:₩Users₩sure₩Desktop₩zlib₩adler32.c	C:₩Users₩sure₩Desktop₩newPath₩adler32.c	OK
3	2	C:₩Users₩sure₩Desktop₩zlib₩compress.c	C:₩Users₩sure₩Desktop₩newPath₩adler32.c	OK
4	3	C:₩Users₩sure₩Desktop₩zlib₩crc32.c	C:₩Users₩sure₩Desktop₩newPath₩crc32.c	The new path you entered is not valid. Please check the new path.
5	4	C:₩Users₩sure₩Desktop₩zlib₩deflate.c	C:₩Users₩sure₩Desktop₩newPath₩deflate.c	OK
6	5	C:₩Users₩sure₩Desktop₩zlib₩gzclose.c	C:₩Users₩sure₩Desktop₩newPath₩gzclose.c	OK
7	6	C:₩Users₩sure₩Desktop₩zlib₩gzlib.c	C:₩Users₩sure₩Desktop₩newPath₩gzlib.c	OK
8	7	C:₩Users₩sure₩Desktop₩zlib₩gzread.c	C:₩Users₩sure₩Desktop₩newPath₩gzread.c	OK
9	8	C:₩Users₩sure₩Desktop₩zlib₩gzwrite.c	C:₩Users₩sure₩Desktop₩newPath₩gzwrite.c	OK
10	9	C:₩Users₩sure₩Desktop₩zlib₩infback.c	C:₩Users₩sure₩Desktop₩newPath₩infback.c	OK
11	10	C:₩Users₩sure₩Desktop₩zlib₩inffast.c	C:₩Users₩sure₩Desktop₩newPath₩inffast.c	OK
12	11	C:₩Users₩sure₩Desktop₩zlib₩inflate.c	C:₩Users₩sure₩Desktop₩newPath₩inflate.c	OK
13	12	C:₩Users₩sure₩Desktop₩zlib₩inftrees.c	C:₩Users₩sure₩Desktop₩newPath₩inftrees.c	OK
14	13	C:₩Users₩sure₩Desktop₩zlib₩trees.c	C:₩Users₩sure₩Desktop₩newPath₩trees.c	OK
15	14	C:₩Users₩sure₩Desktop₩zlib₩uncompr.c	C:₩Users₩sure₩Desktop₩newPath₩uncompr.c	OK
16	15	C:₩Users₩sure₩Desktop₩zlib₩zutil.c	C:₩Users₩sure₩Desktop₩newPath₩zutil.c	OK
17				

6. 실제 타깃 환경에서 테스트하기

CT 2023.12를 이용하여 실제 타깃 환경에서 테스트하는 방법은 다음과 같습니다.

- 타깃 테스트 가이드
- 디버거 사용 가이드
- 타깃 빌드 가이드

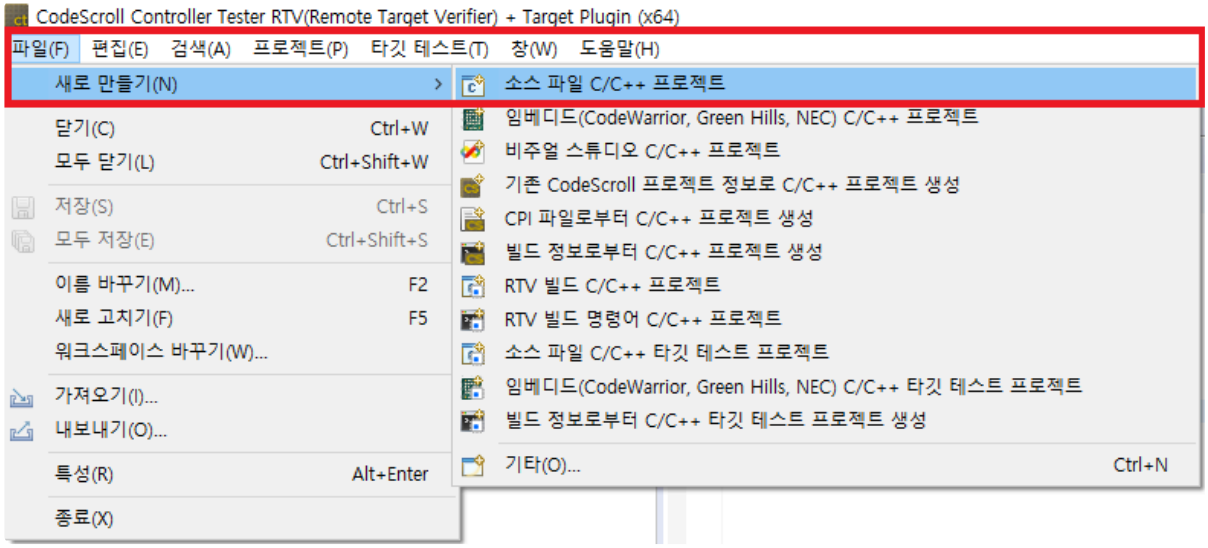
6.1. 타깃 테스트 가이드

CT 2023.12를 사용하여 타깃 테스트를 하는 시나리오를 설명합니다.

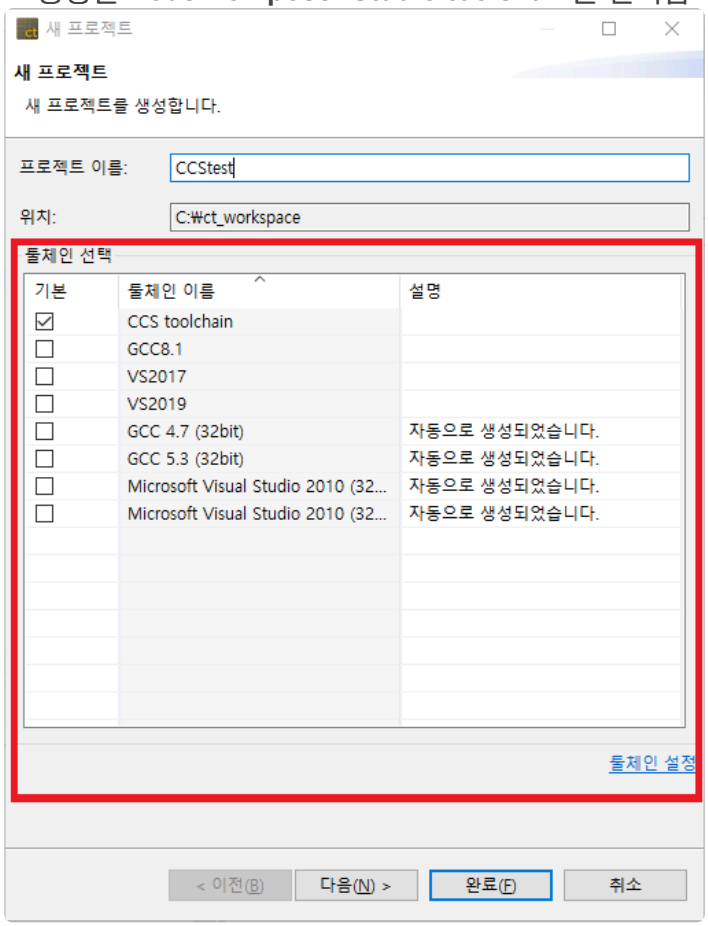
- Texas Instruments Code Composer Studio
- STM32cubeIDE
- Wind River Workbench

6.1.1. Texas Instruments Code Composer Studio

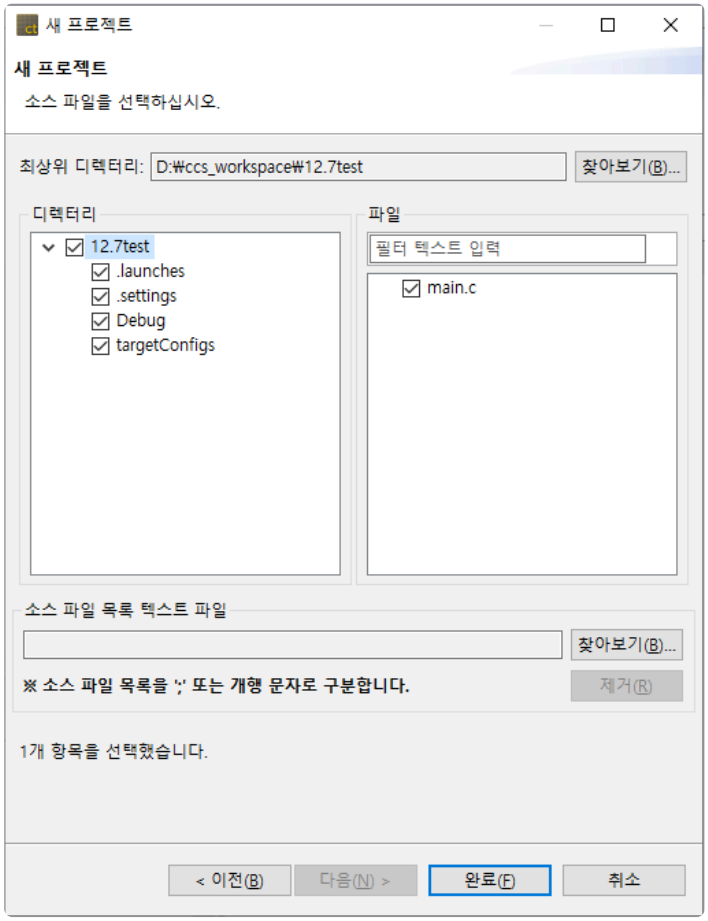
1. CT 2023.12 프로젝트를 생성합니다.



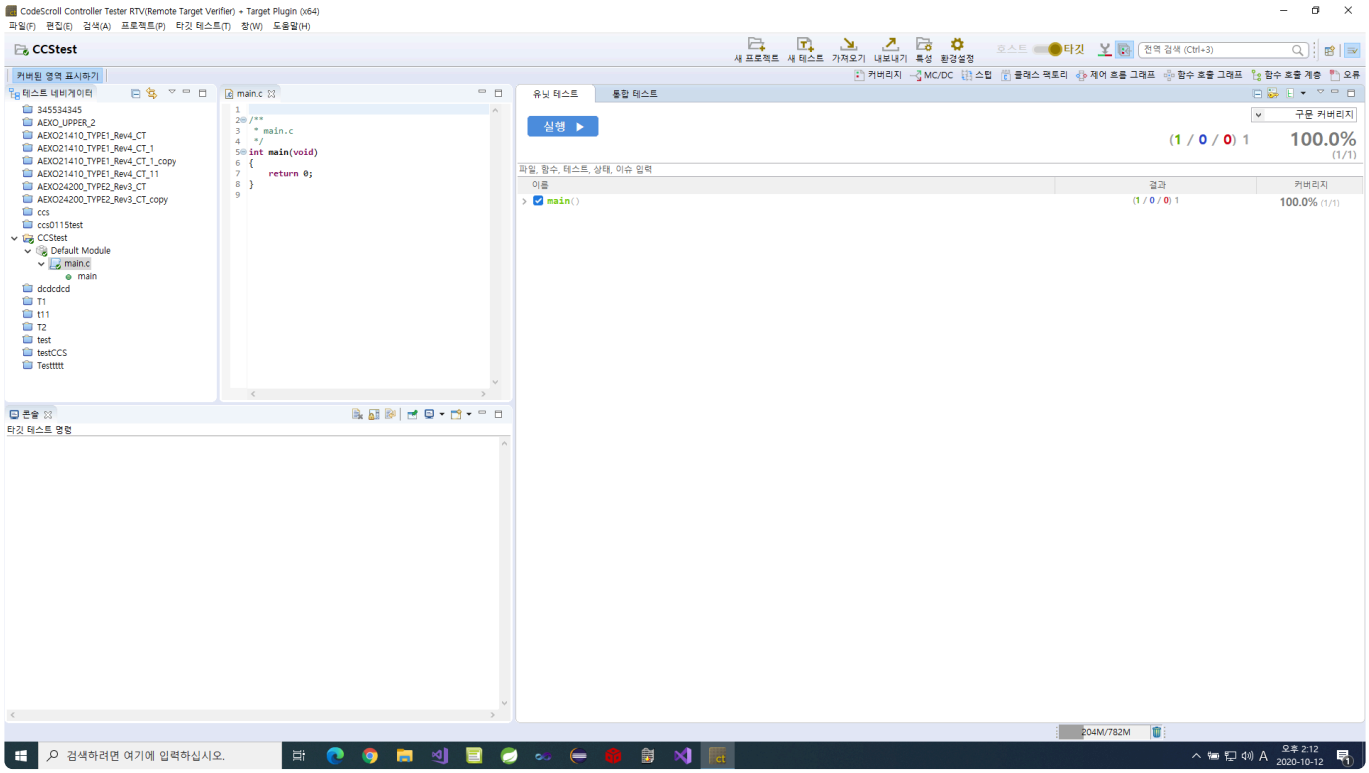
2. 생성한 Code Composer Studio toolchain을 선택합니다.



3. 테스트할 소스 파일을 선택합니다.



4. 해당 설정을 모두 마치고 완료를 클릭하면 프로젝트가 생성됩니다.



5. 빌드 자동화를 위해 **CT 2023.12**의 타깃 환경 설정이 필요합니다. 자세한 사항은 본 문서 타깃 빌드 가이드의 하위 항목인 **Texas Instruments Code Composer Studio**를 참고하시기 바랍니다.

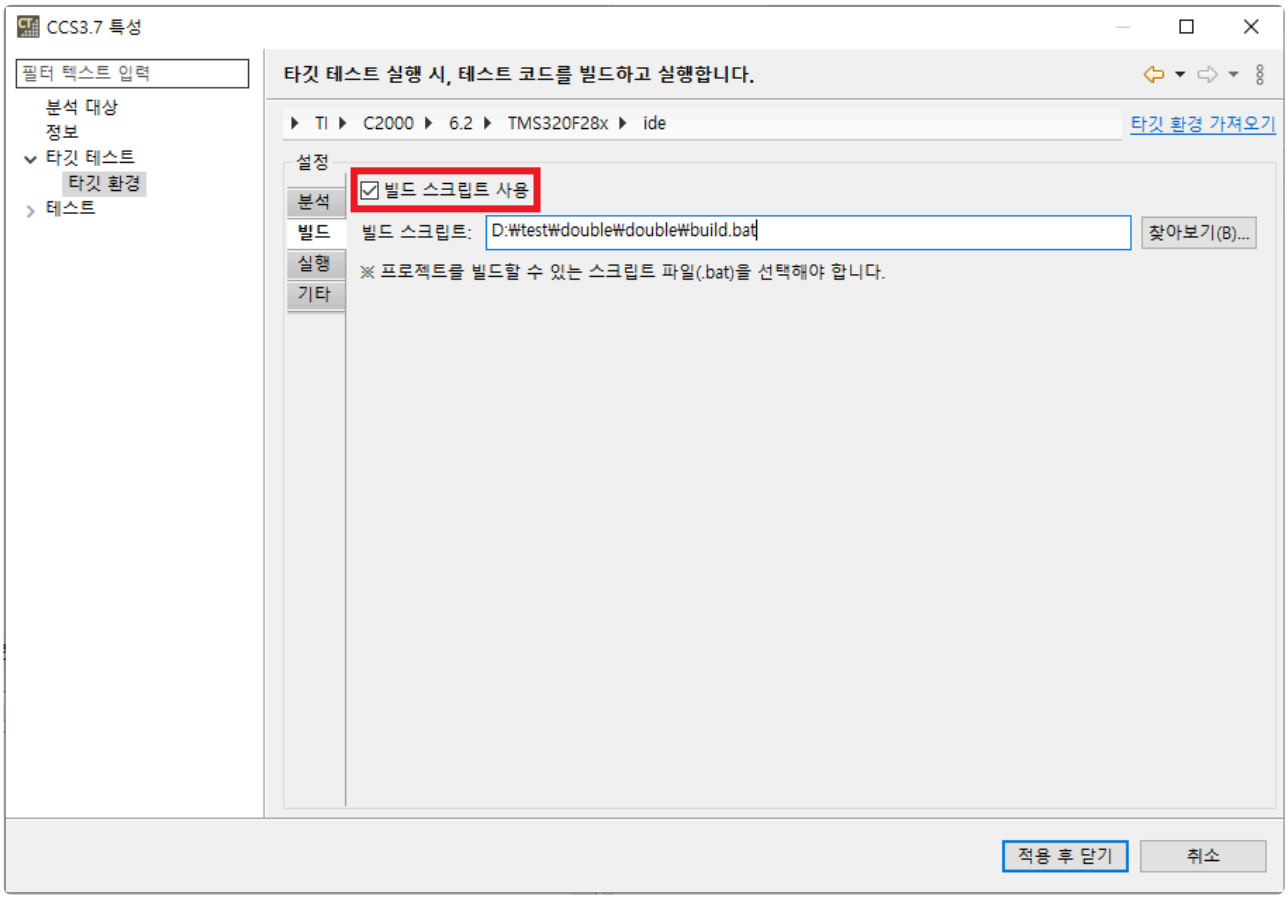
- 타깃 환경 설정 작성 후에 빌드를 실행하면 바이너리가 정상적으로 생성되는 지 확인해야 합니다. 정상적으로 생성되지 않은 경우에는 빌드 구조가 기본값과 다르게 설정되어 있어서 명령어를 정상적으

로 생성해주지 못한 경우일 수 있습니다.

- 이러한 경우에는 타깃 환경 설정의 빌드 탭에서 빌드 스크립트 사용 기능을 활용하여 빌드를 진행할 수 있습니다.

```
cd "%Code Composer Studio 프로젝트 워크스페이스 경로%\Debug"
"C:\\ti\\ccs%ccs_version%\\utils\\bin\\gmake" -k clean
"C:\\ti\\ccs%ccs_version%\\utils\\bin\\gmake" -k all
```

- 위의 예시와 같이 Code Composer Studio에서 빌드 시 console에 출력해주는 빌드 명령어를 이용하여 빌드 스크립트를 .bat 파일로 저장 후에, 타깃 환경 설정의 빌드 탭의 빌드 스크립트에 해당 .bat 파일을 입력합니다. gmake의 위치는 버전별로 상이할 수 있습니다.



- 그 후 정상적으로 Controller Tester에서 빌드 시 바이너리가 생성되는 지 확인합니다.

6. 디버거를 사용하기 위해 **Code Composer Studio, CT 2023.12**에서 설정을 합니다. 자세한 사항은 본 문서 디버거 사용 가이드의 하위 항목인 **Texas Instruments Code Composer Studio**를 참고하시기 바랍니다.

6.1.2. STM32cubeIDE

이 문서에서는 STM32 계열의 타깃에 대해 STM32cubeIDE를 사용하여 타깃 테스트를 진행하는 방법에 대해 서술합니다.

적용 예시 환경은 다음과 같으며, ST-Link 디버거를 사용하였습니다.

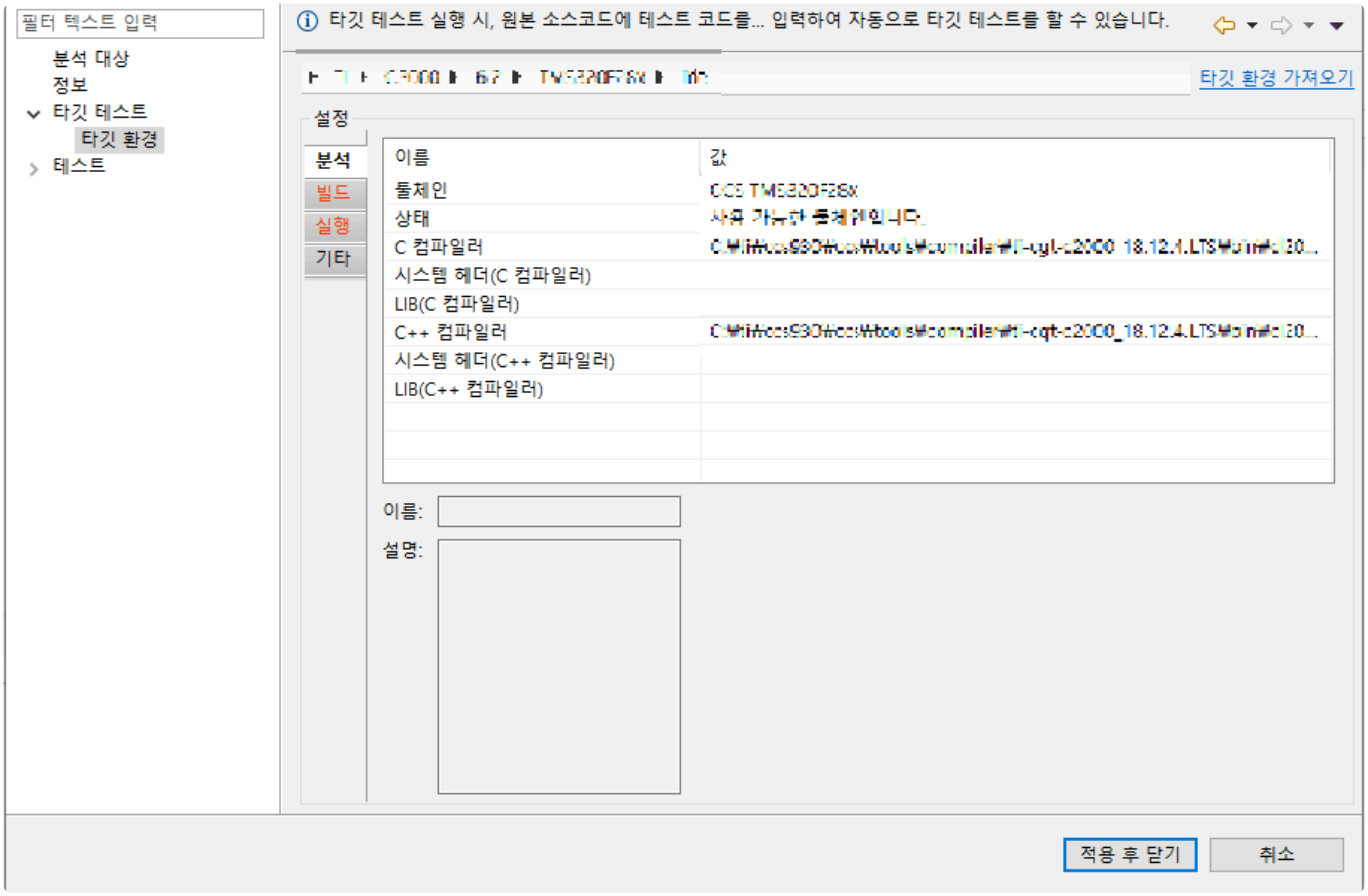
No.	개발 환경(OS)		빌드 환경(OS)		개발 언어	통합 개발 환경(IDE)		컴파일러		빌드 방식	타깃(실행 환경)	
	종류	버전	종류	버전		종류	버전	종류	버전	(Makefile, IDE)	아키텍처	칩셋(Chipset)
1	Windows	10	Windows	10	C	Stm32cubeide	1.6.1	Arm-none-eabi-gcc	GNU Tools for STM32 9-2020-q2-update.20201001-1621) 9.3.1 20200408	IDE	ARM Cortex-M7, 32Bit MCU	SMT32F7 Series
2	Windows	10	Windows	10	C++	Stm32cubeide	1.6.1	Arm-none-eabi-g++	GNU Tools for STM32 9-2020-q2-update.20201001-1621) 9.3.1 20200408	IDE	ARM Cortex-M7, 32Bit MCU	SMT32F7 Series

✻ 본 가이드는 타깃 테스트 가이드이므로, 프로젝트 생성 및 분석이 완료되었다는 것을 전제로 합니다.

! 가이드 문서의 각 과정에서 발생하는 문제들은 CT Target Plug-in 문제 해결 가이드 를 통하여 해결할 수 있습니다.

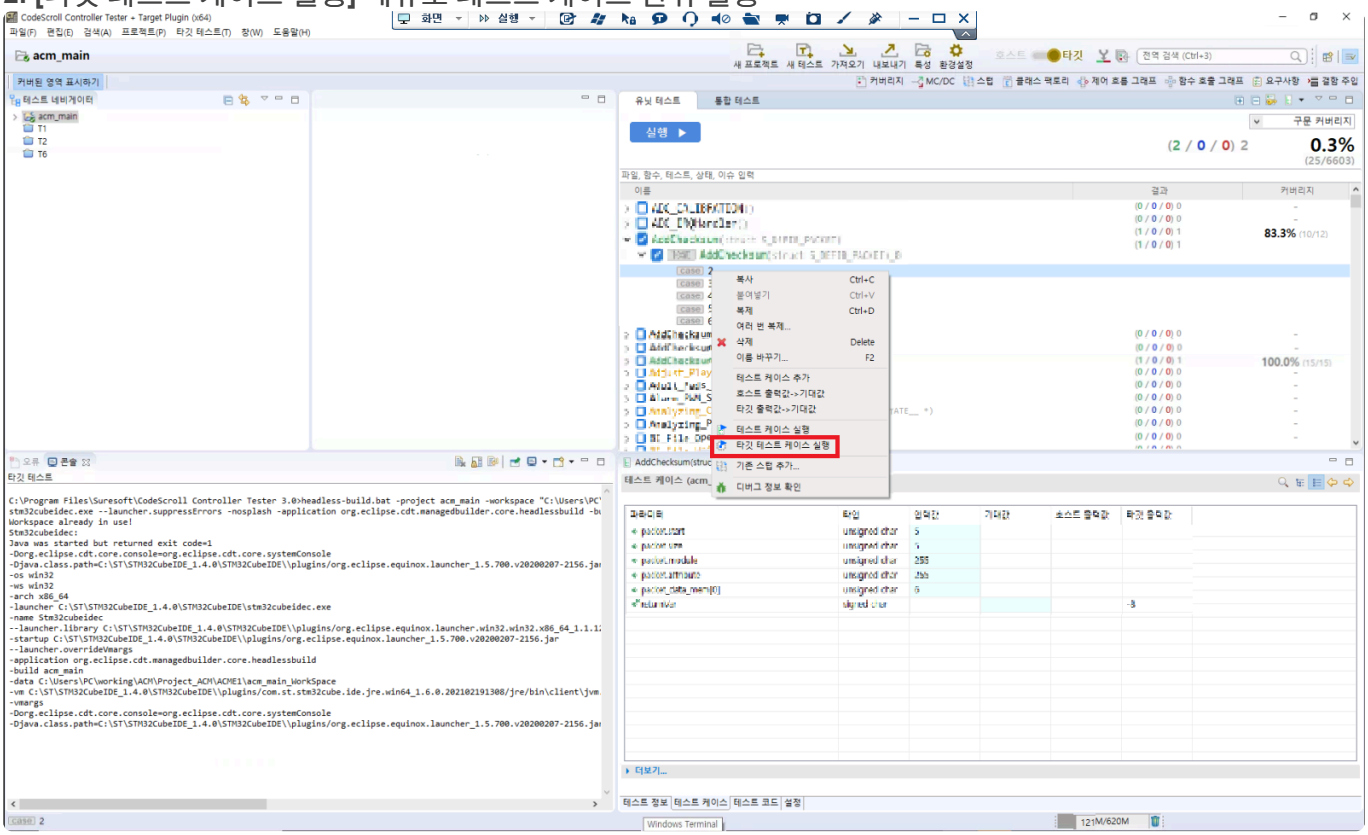
타깃 테스트 적용 및 실행 순서

1. 타깃 환경 설정

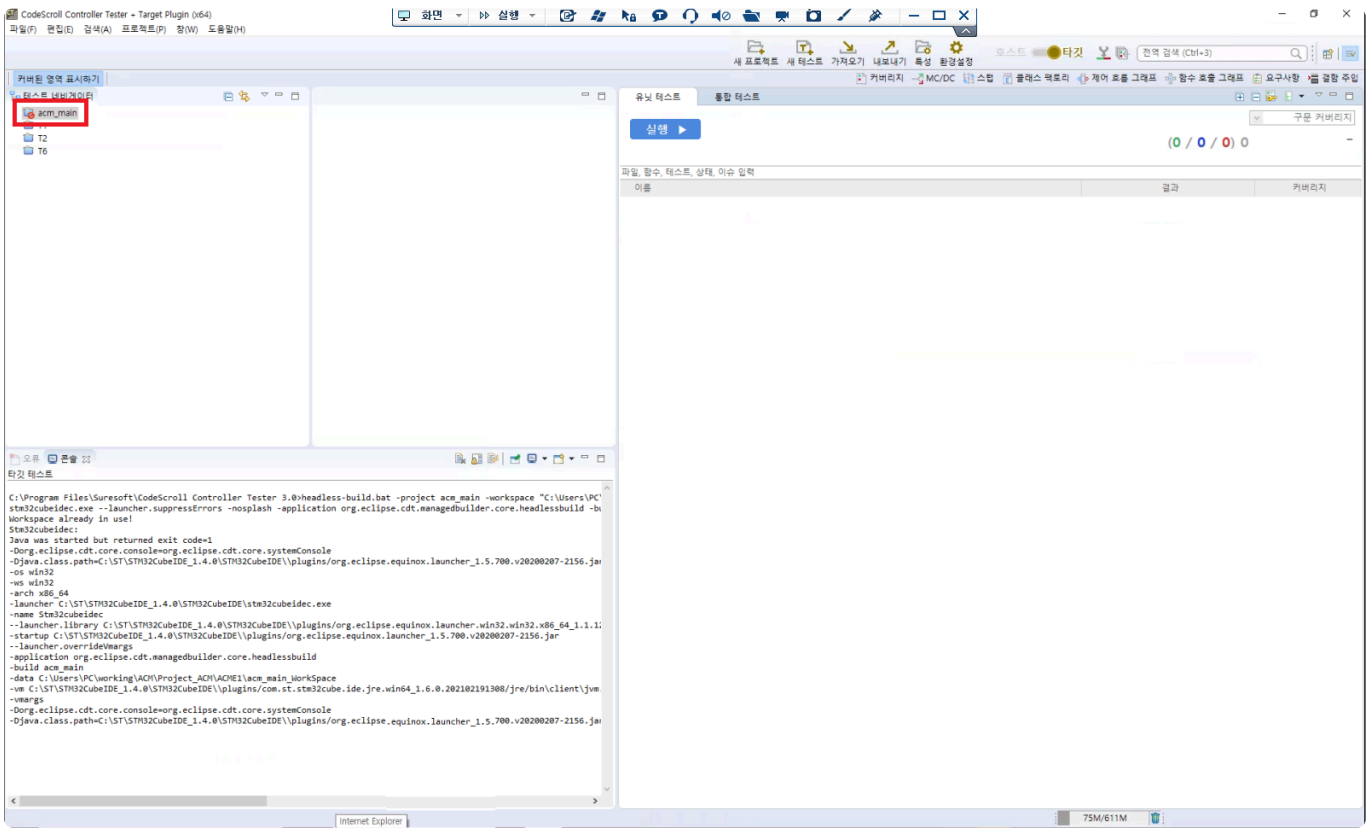


- [프로젝트 우클릭] -> [특성] -> [타깃 테스트] -> [타깃 환경] 설정에서 분석 탭만 작성하여 적용 후 닫기를 합니다. 해당 타깃 테스트 문서는 수동 빌드 방식으로 진행되어 다른 탭들은 테스트에 영향을 미치지 않습니다.

2. [타깃 테스트 케이스 실행] 메뉴로 테스트 케이스 단위 실행

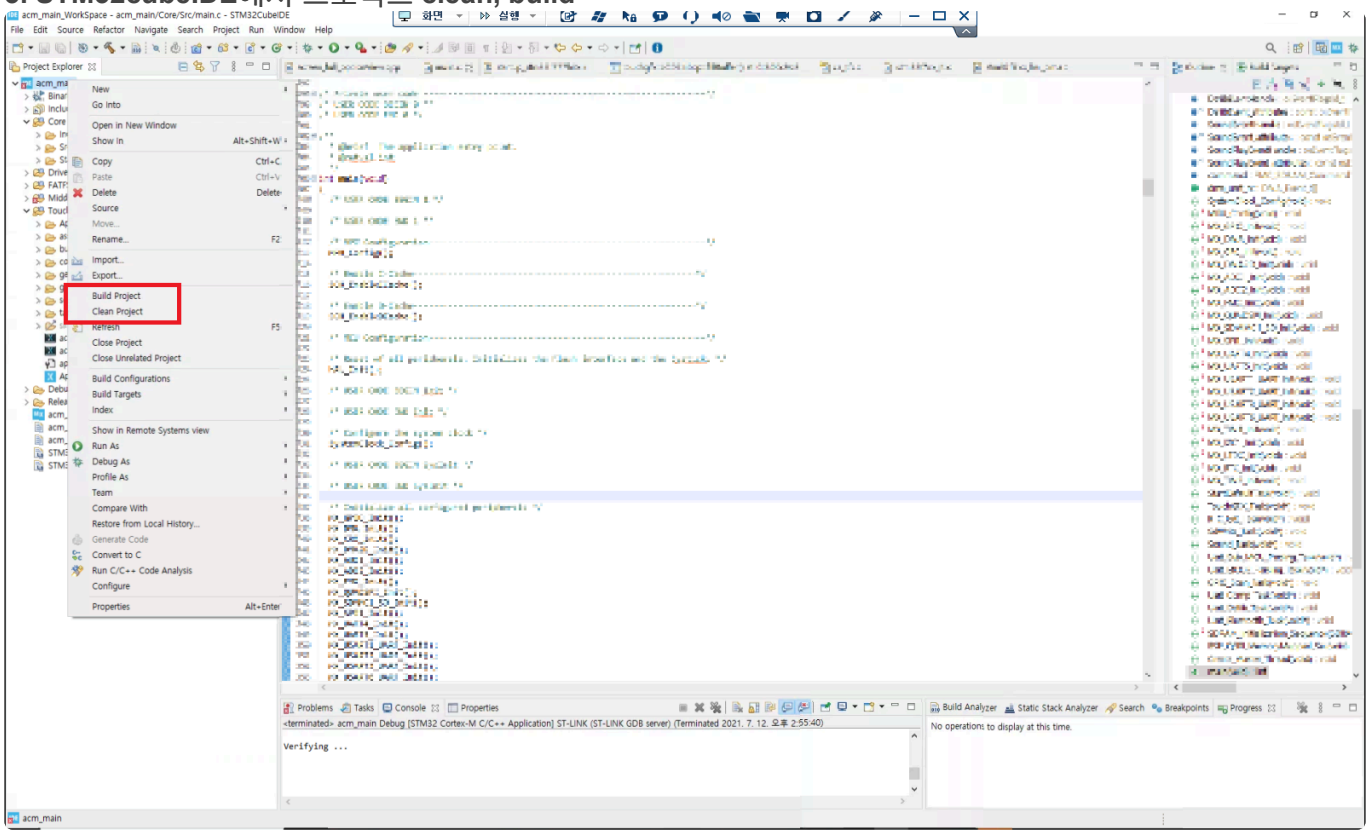


- 정확한 테스트를 위해 테스트 케이스 단위로 실행합니다.



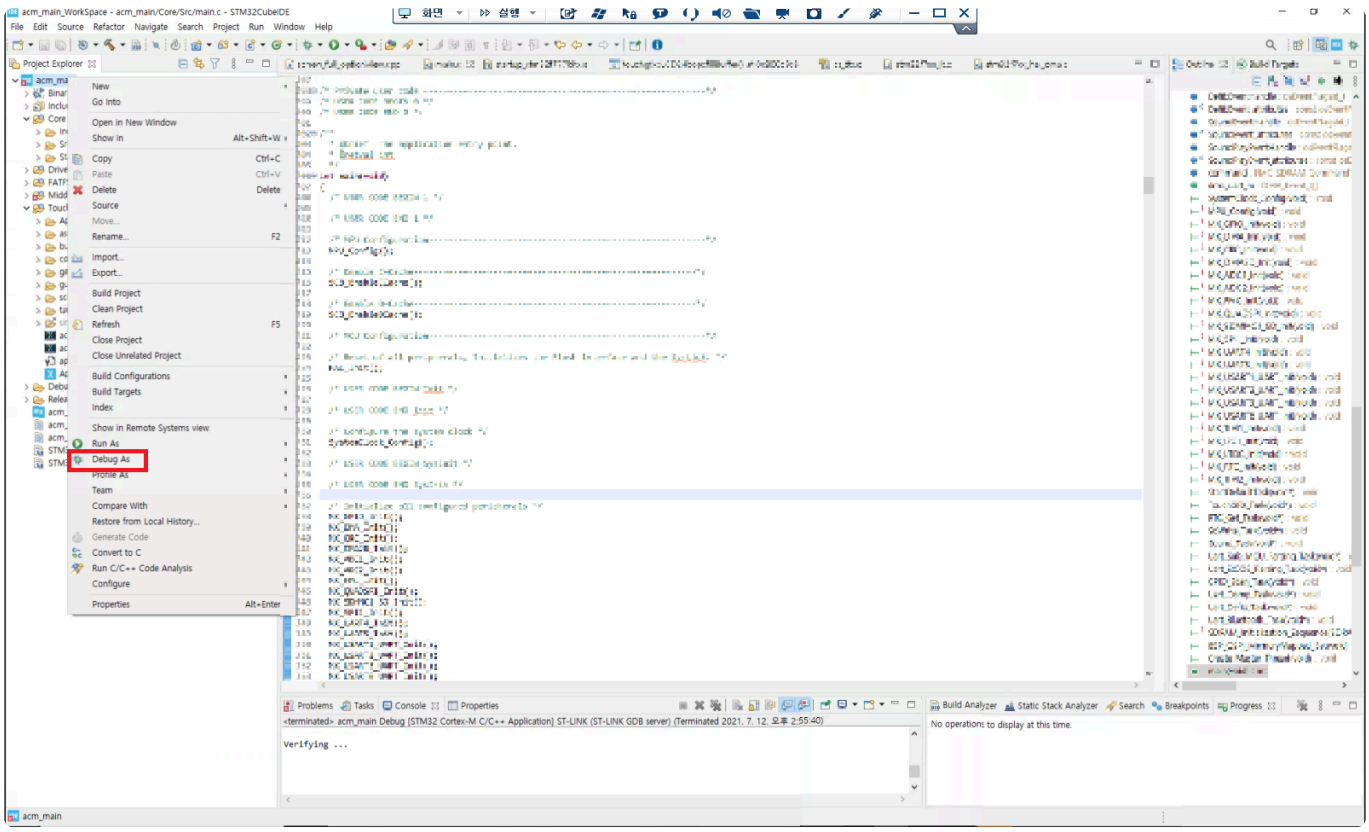
- 1,2번 내용을 거치면 CT 2023.12의 프로젝트가 위 상태처럼 잠기게 됩니다.

3. STM32cubeIDE에서 프로젝트 clean, build



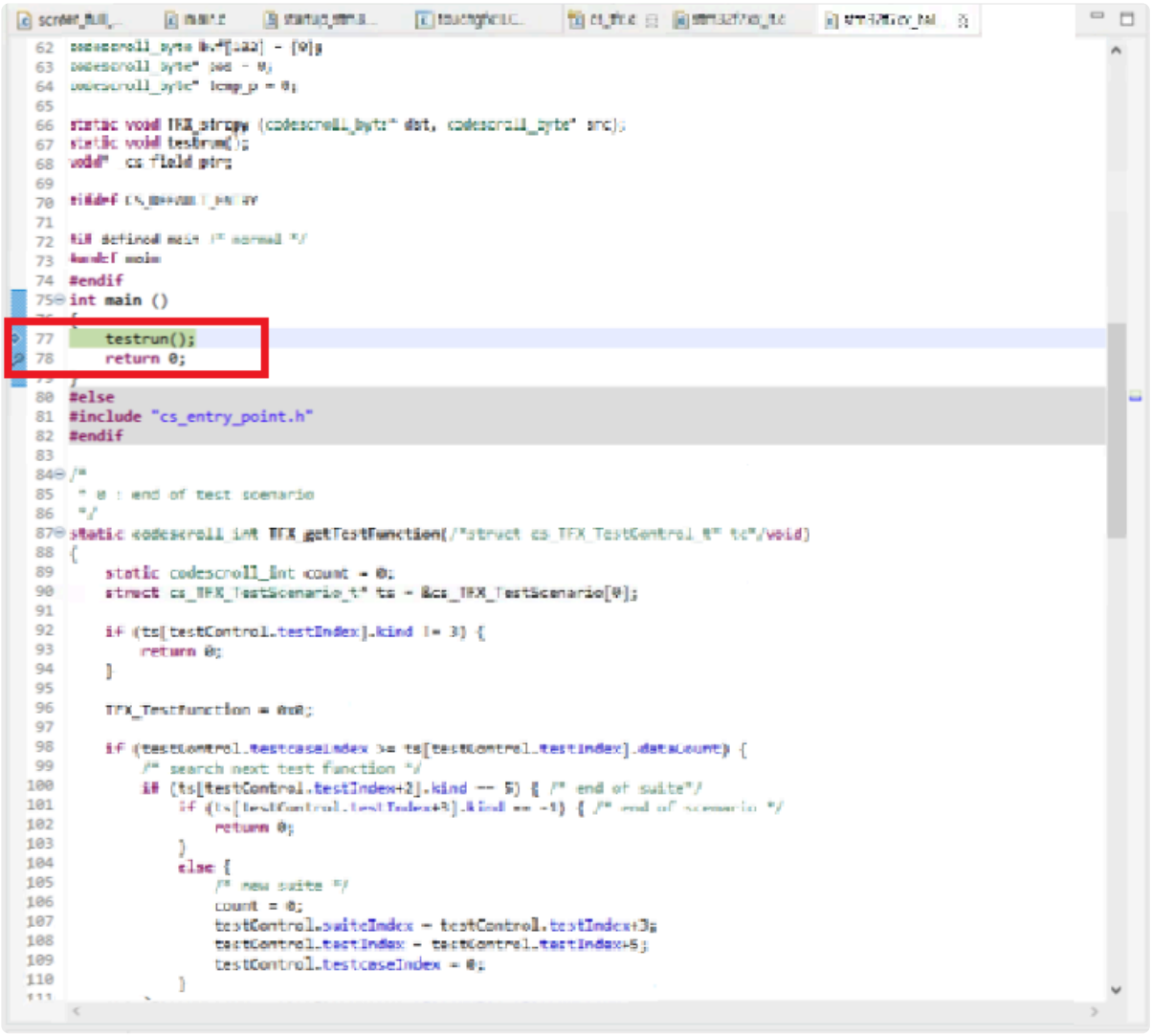
- 내보내기 된 코드를 STM32cubeIDE에서 clean 후 build합니다.

4. STM32cubeIDE에서 Debug



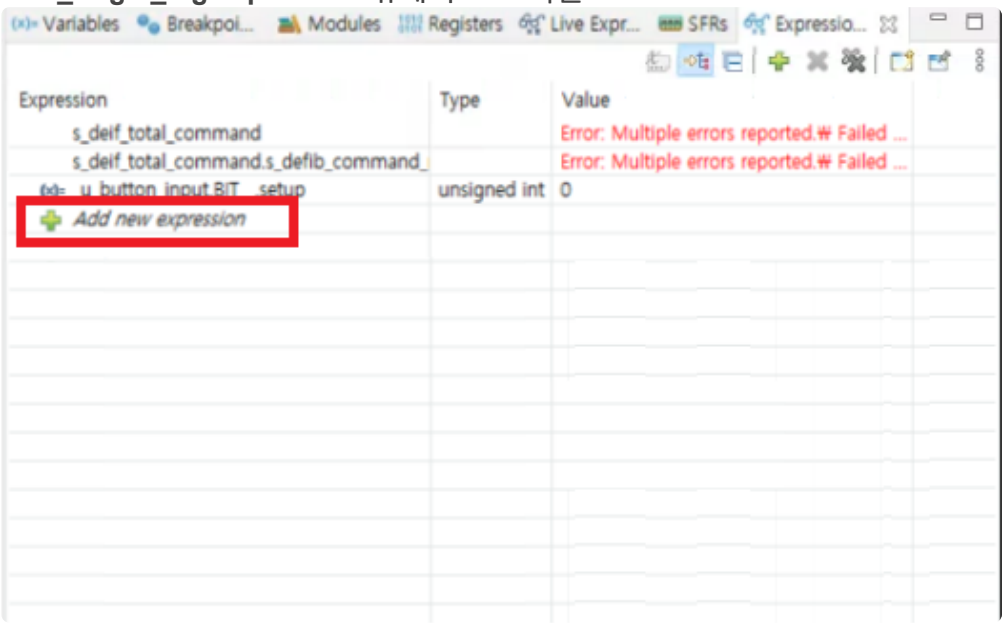
- build가 정상적으로 되었다면, debug를 실행합니다.

5. return 0;에 break point 설정 후 실행

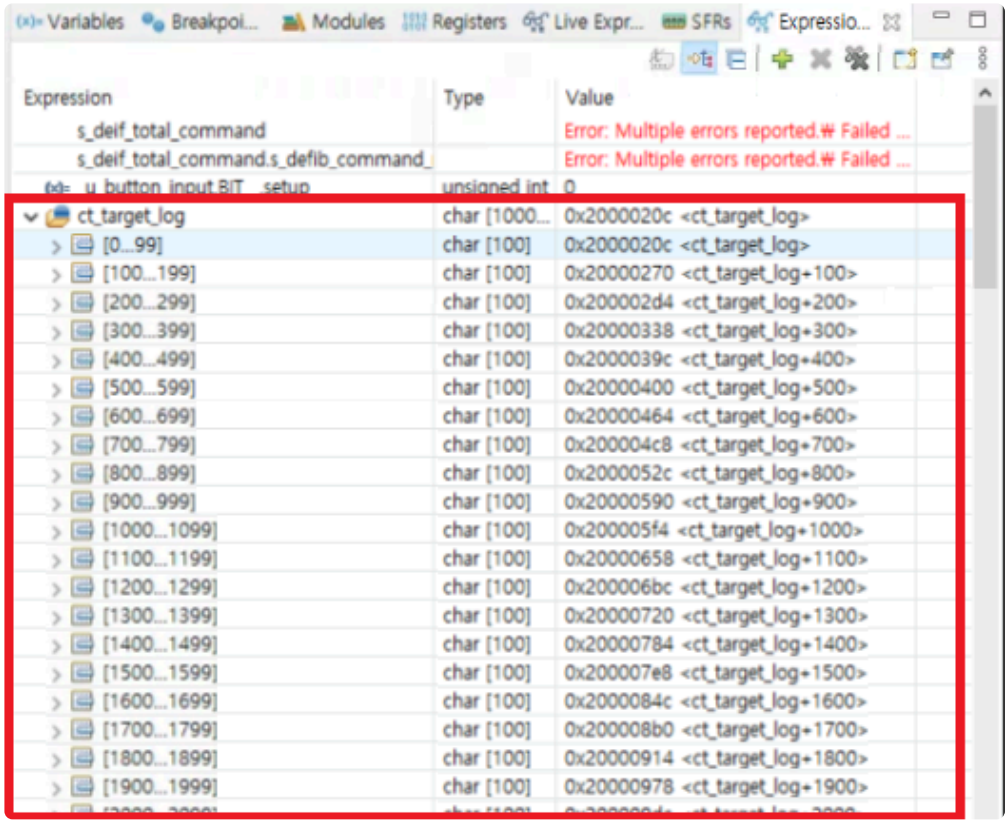


- 시작 지점은 cs_tfx.c의 main 입니다. testrun();이 끝나는 시점인 return 0; 앞에 break point를 걸어줍니다.

6. ct_target_log expression 뷰에서 로그 확인

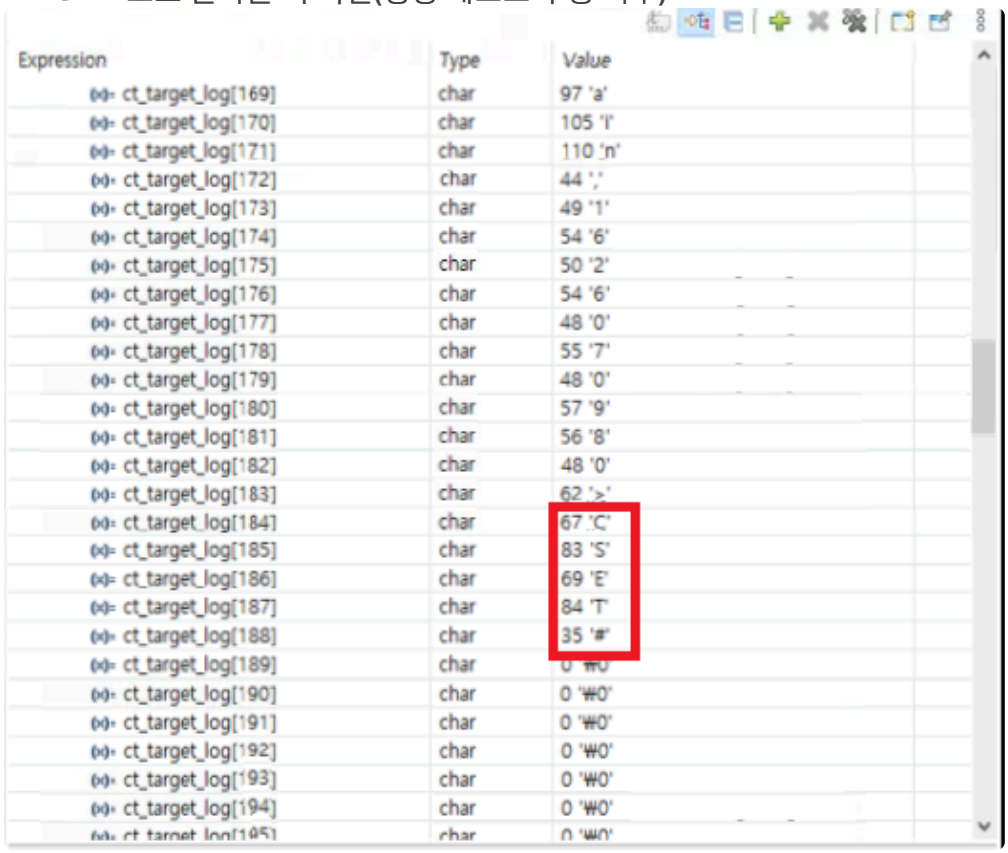


- expression 뷰에서 Add new expression을 클릭하여 로그 인터페이스에 작성한 로그를 담고 있는 배열(ct_target_log)을 추가합니다.



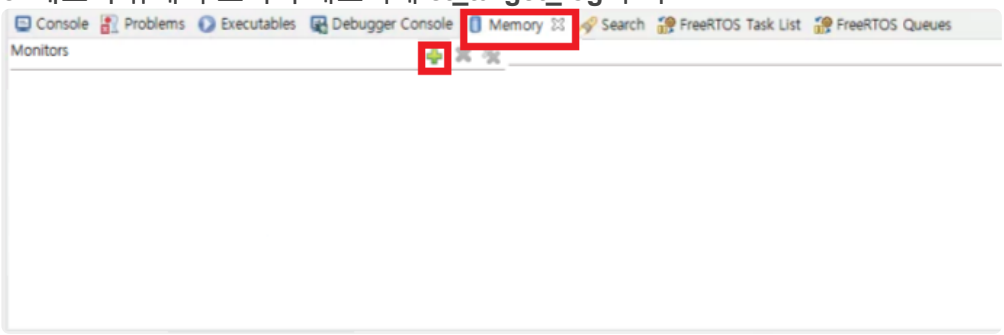
- 이와 같이 ct_target_log의 내용을 확인할 수 있습니다.

7. CSET#으로 끝나는 지 확인(정상 테스트 수행 여부)

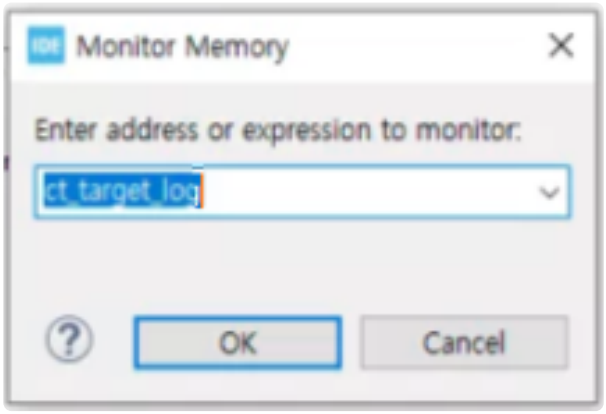


- log의 마지막 부분이 CSET#으로 종료되어야 정상적으로 테스트가 종료되었다고 판단할 수 있습니다. 따라서 expression 뷰에서 테스트의 정상 실행 여부를 한 번 확인할 수 있습니다.

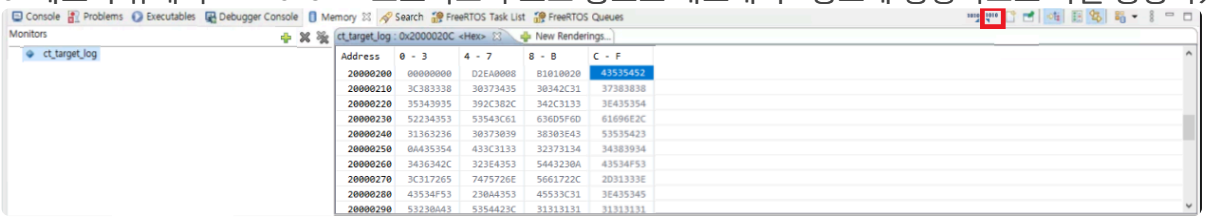
8. 메모리 뷰에서 모니터 메모리에 **ct_target_log** 추가



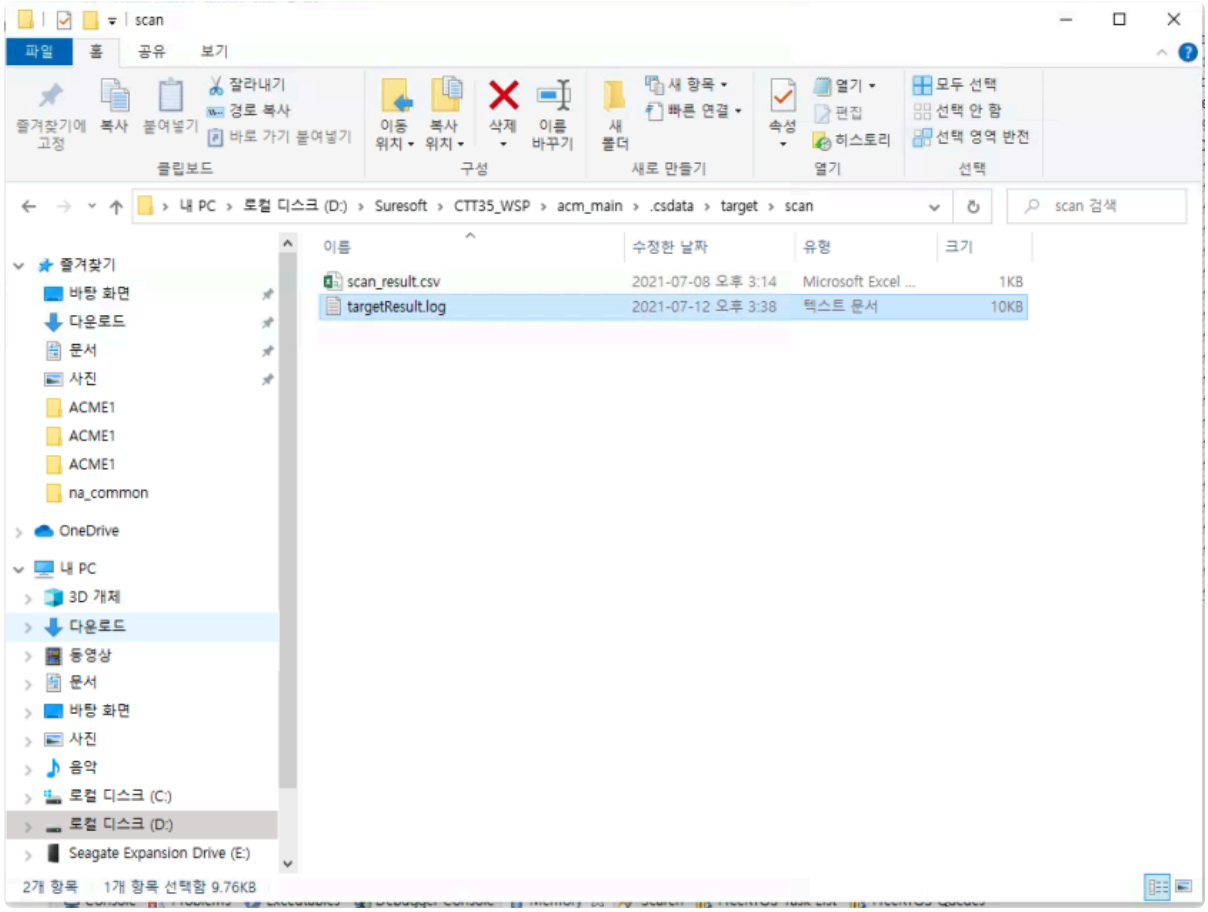
- 메모리 덤프를 위해 memory 뷰에서 Moniters에 +을 클릭하여 ct_target_log를 추가합니다.



9. 메모리 뷰에서 **CT 2023.12** 프로젝트의 로그 경로로 내보내기 / 경로에 정상적으로 파일 생성되었는지 확인

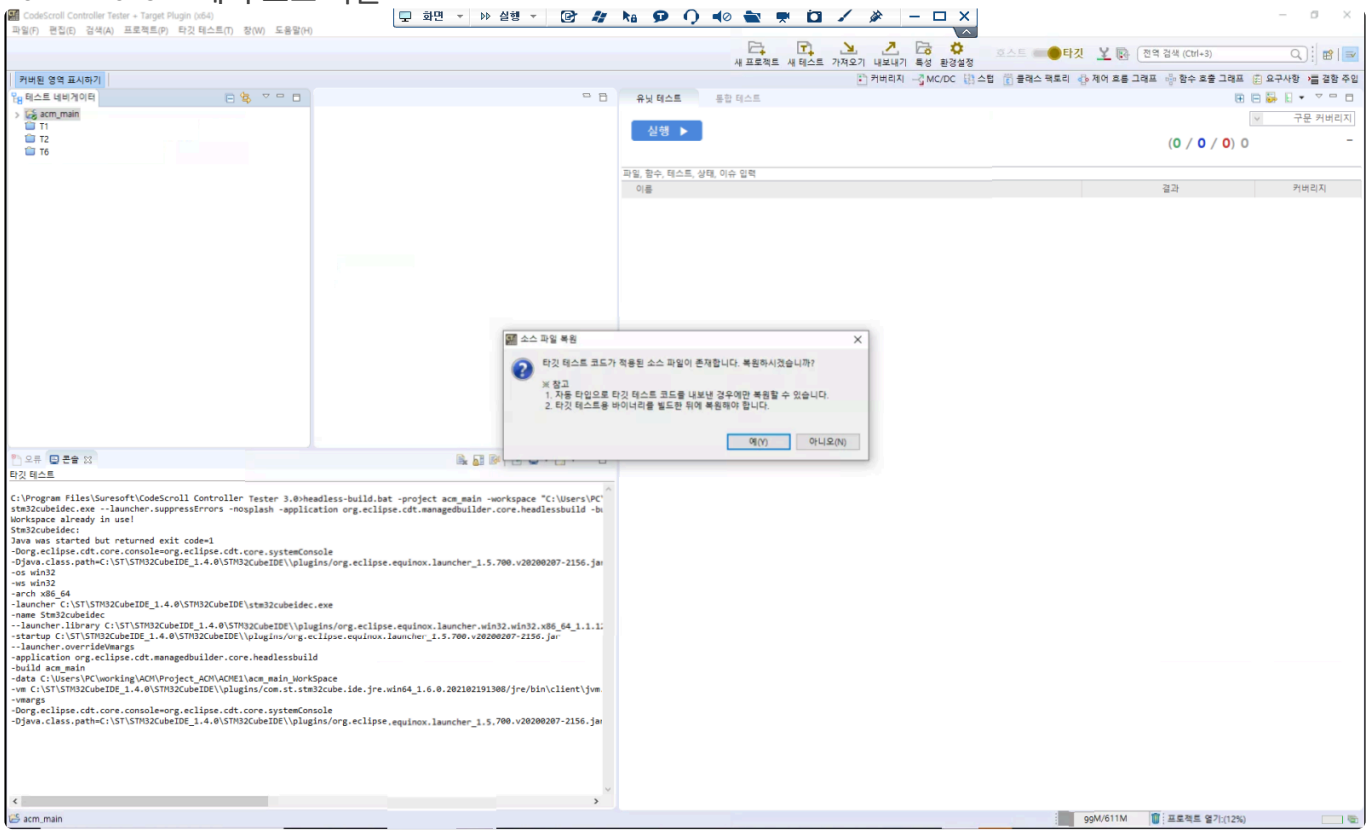


- 메모리 뷰에서 export 버튼을 눌러 ct_target_log의 메모리를 파일로 내려받습니다.
- Format은 RAW Binary, Start address는 expression 뷰의 ct_target_log 시작 주소, Length는 ct_target_log의 배열 크기를 지정합니다. (로그의 길이가 Length보다 짧아도 테스트에 영향을 주지 않습니다.)



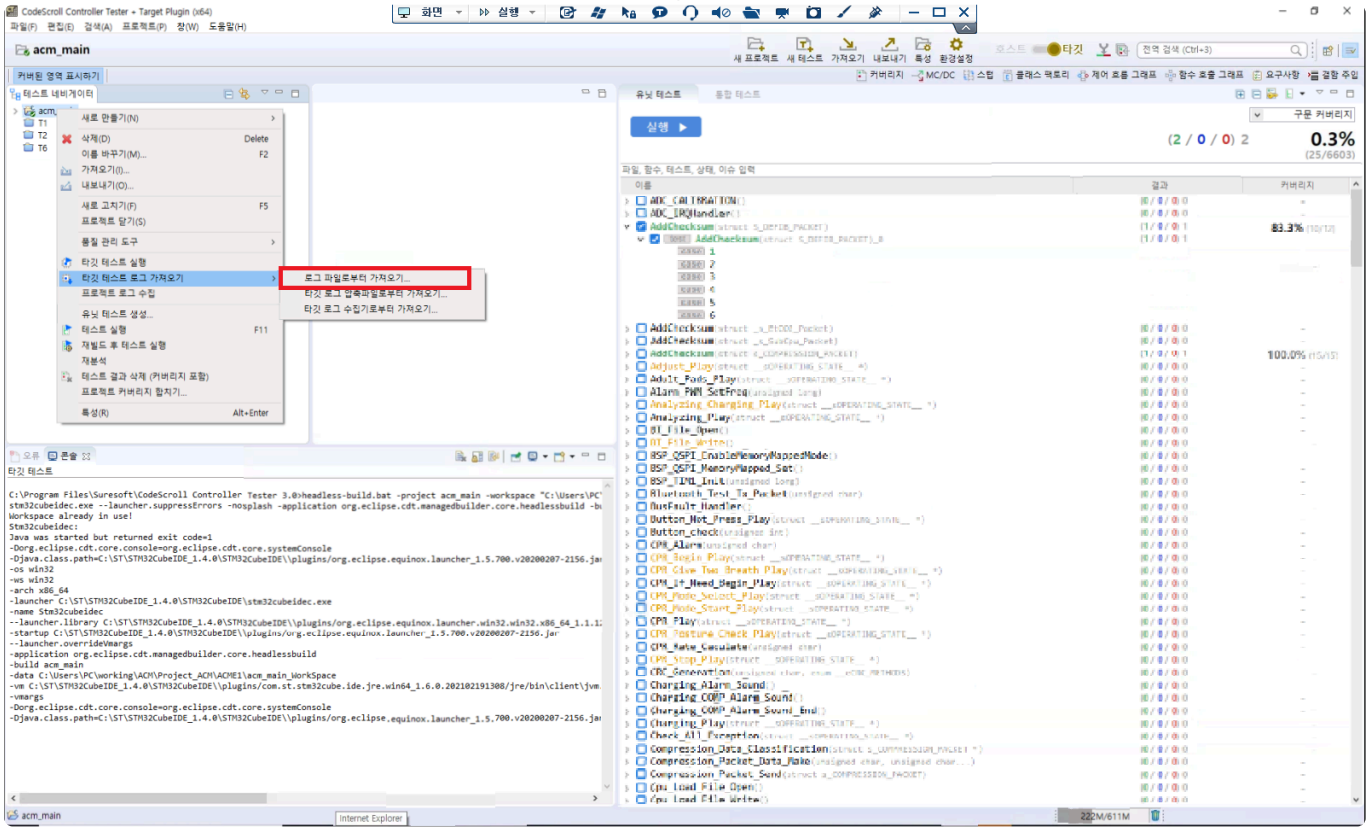
- 지정한 경로에 파일이 정상적으로 만들어졌는 지 확인합니다.

10. CT 2023.12에서 코드 복원

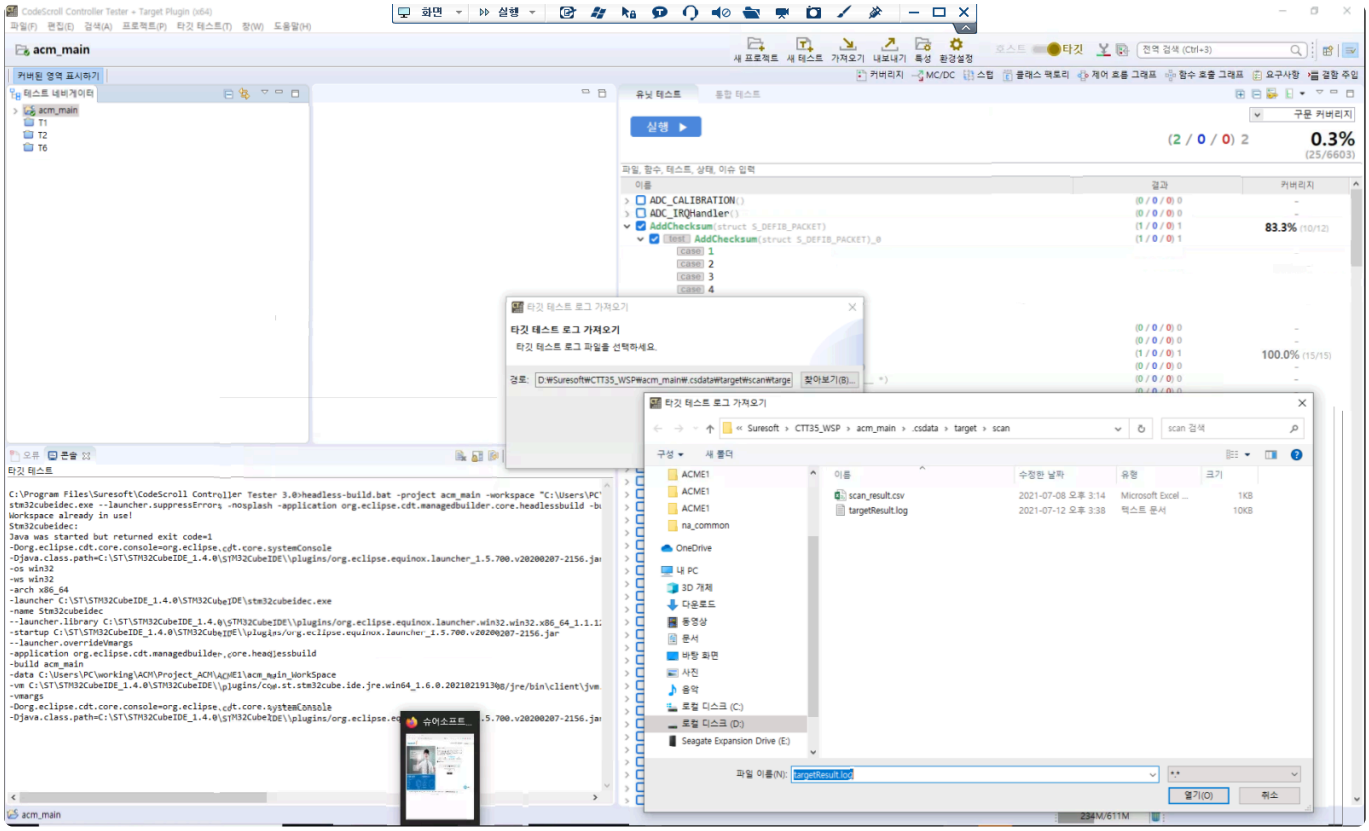


- 타깃 테스트 로그를 가져오기 위해 CT 2023.12에서 코드를 복원합니다.

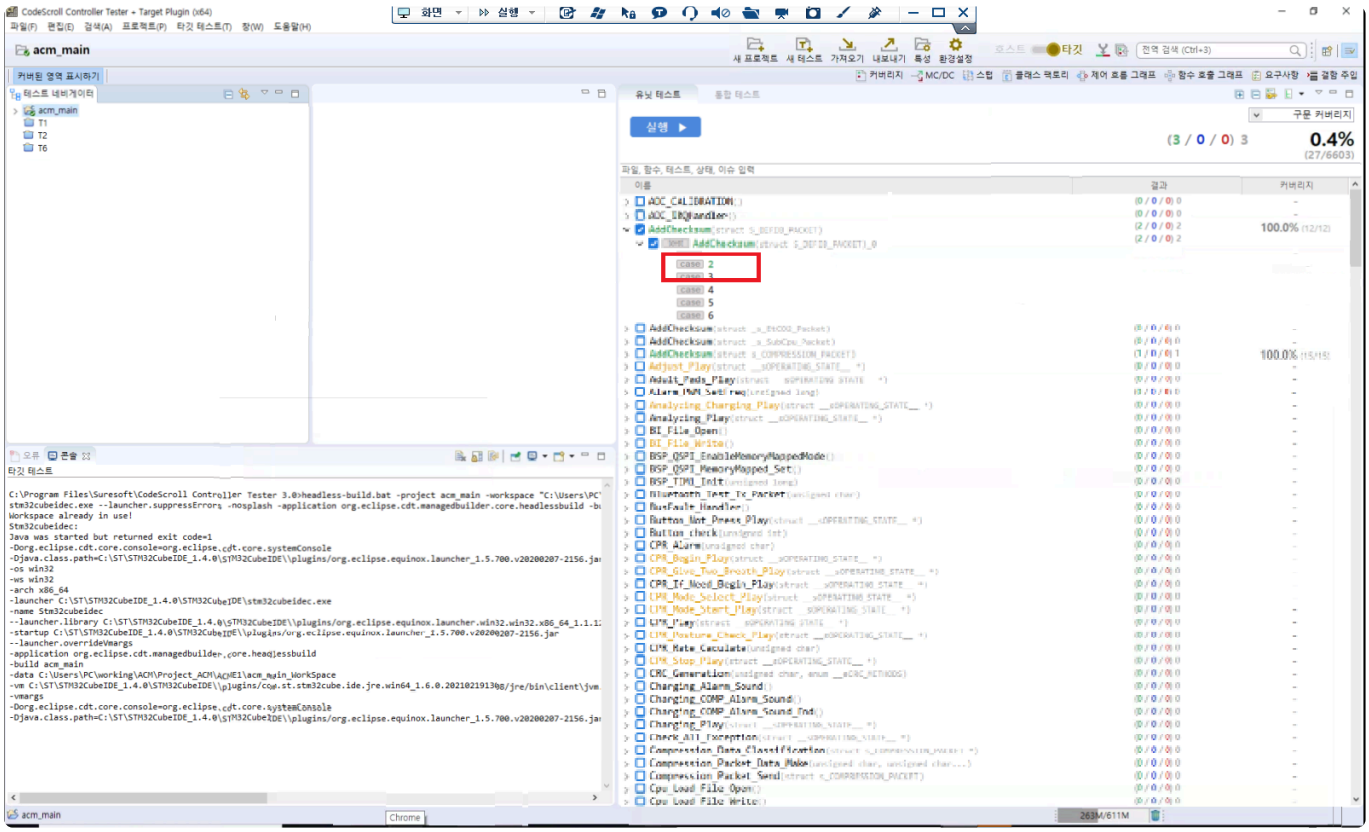
11. 타깃 테스트 로그 가져오기 -> 로그파일 경로로부터 가져오기 -> 완료



- [타깃 테스트 로그 가져오기] -> [로그 파일로부터 가져오기]를 선택합니다.



- 9번에서 생성된 파일을 가져옵니다.



- 테스트 케이스가 성공적으로 수행되어 커버리지가 측정되었음을 확인할 수 있습니다.

6.1.3. Wind River Workbench

이 문서에서는 VxWorks 6.9 타깃 실행 환경에 대해 Wind River Workbench IDE를 사용하여 타깃 테스트를 진행하는 방법에 대해 서술합니다.

적용 예시 환경은 다음과 같으며, 로그 인터페이스는 TCP 소켓 통신을 사용하였습니다.

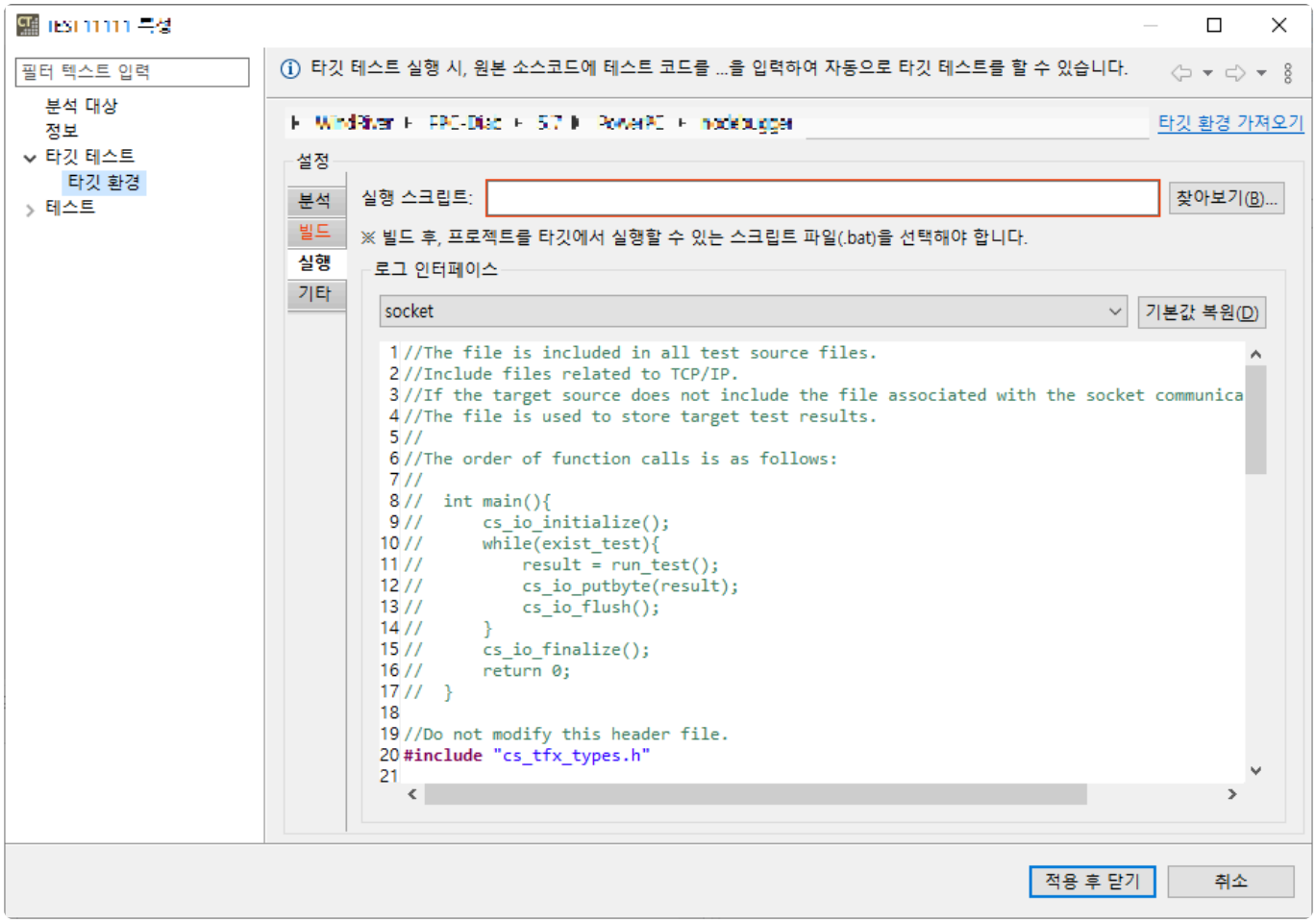
No.	개발 환경(OS)		빌드 환경(OS)		개발 언어	통합 개발 환경(IDE)		컴파일러		빌드 방식	타깃(실행 환경)	
	종류	버전	종류	버전		종류	버전	종류	버전		아키텍처	칩셋(Chipset)
1	Windows	10	VxWorks	6.9	C	Workbench	3.3	GNU	PPC85XXe500v2	Makefile generated by the IDE	x86	P2020NXN2KFC

> 본 가이드는 타깃 테스트 가이드이므로, 프로젝트 생성 및 분석이 완료되었다는 것을 전제로 합니다.

> 가이드 문서의 각 과정에서 발생하는 문제들은 Controller Tester Target Plug-in 문제 해결 가이드 를 통하여 해결할 수 있습니다.

타깃 테스트 적용 및 실행 순서

1. 타깃 환경 설정

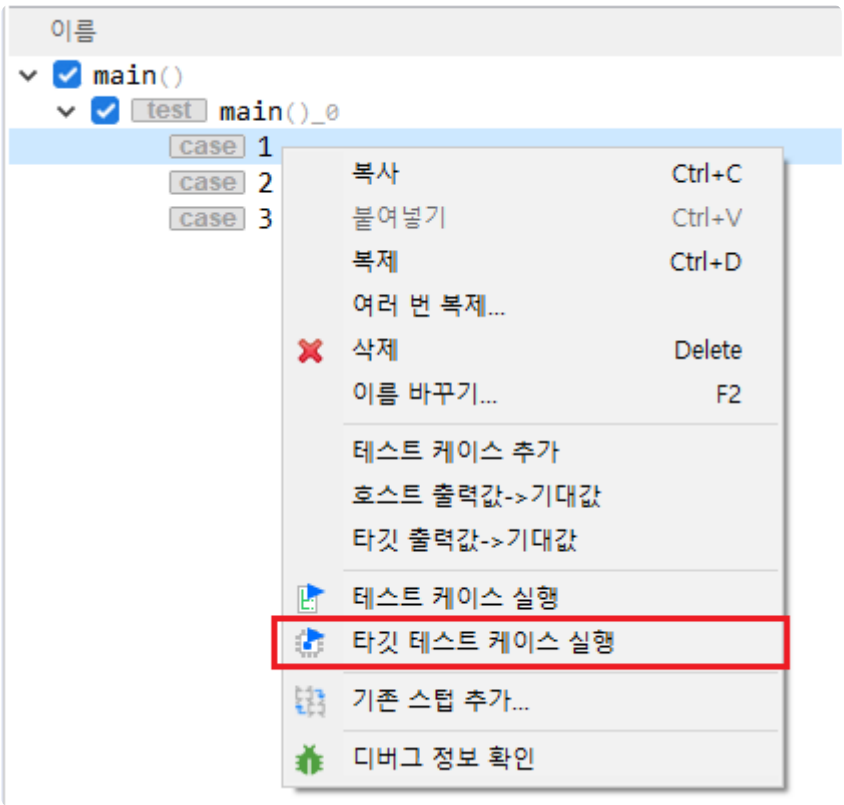


- [프로젝트 우클릭] -> [특성] -> [타깃 테스트] -> [타깃 환경] 설정에서 실행 탭의 로그 인터페이스만 작성하여 적용 후 닫기를 합니다. 이 문서에서 다루는 환경은 빌드 및 실행을 수동으로 진행하므로, 다른 탭들은 테스트에 영향을 미치지 않습니다. 아래 코드는 예시 환경(VxWorks 6.9)에 적용된 로그 인터페이스 입니다.

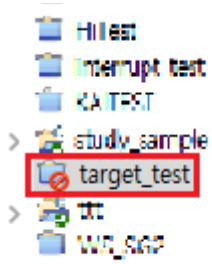
```
//Do not modify this header file.
#include "cs_tfx_types.h"

//Below is an example.

#define AF_INET        2
#define SOCK_STREAM 1
#define htons(x) (x)

struct in_addr {
        unsigned int s_addr;
};

struct sockaddr_in {
        unsigned char sin_len;
        unsigned char sin_family;
        unsigned short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];
};

int sock;
struct sockaddr_in addr;

//This function called at test start.
void cs_io_initialize()
{
 sock = socket(AF_INET, SOCK_STREAM, 0);
 if (sock < 0)
 {
  printf("create socket failed\n");
 }

 memset (&addr, 0, sizeof(addr));
 addr.sin_addr.s_addr = inet_addr("211.116.222.180"); // CT 2023.12가 설치된 PC
의 IP 주소
 addr.sin_port = htons(2019); // 타깃 로그 수집기에서 사용할 포트
 addr.sin_family = AF_INET;
 addr.sin_len = sizeof(addr);

 if ((connect(sock, (struct sockaddr_in *)&addr, sizeof(addr))) < 0) {
  printf("connect failed");
```

```
 }
 else
  printf("connected to server!\n");

 printf("Controller Tester Init End!!!!\n");
}

//This function called at test end.
void cs_io_finalize()
{
 //Close socket.
 close(sock);
}

void cs_io_flush()
{

}

//This function prints the test result
void cs_io_putbyte(char v)
{
 //Send the target test result.
 printf("%c",v);
 send(sock, &v, 1, 0);
}
```

2. 타깃 테스트 실행

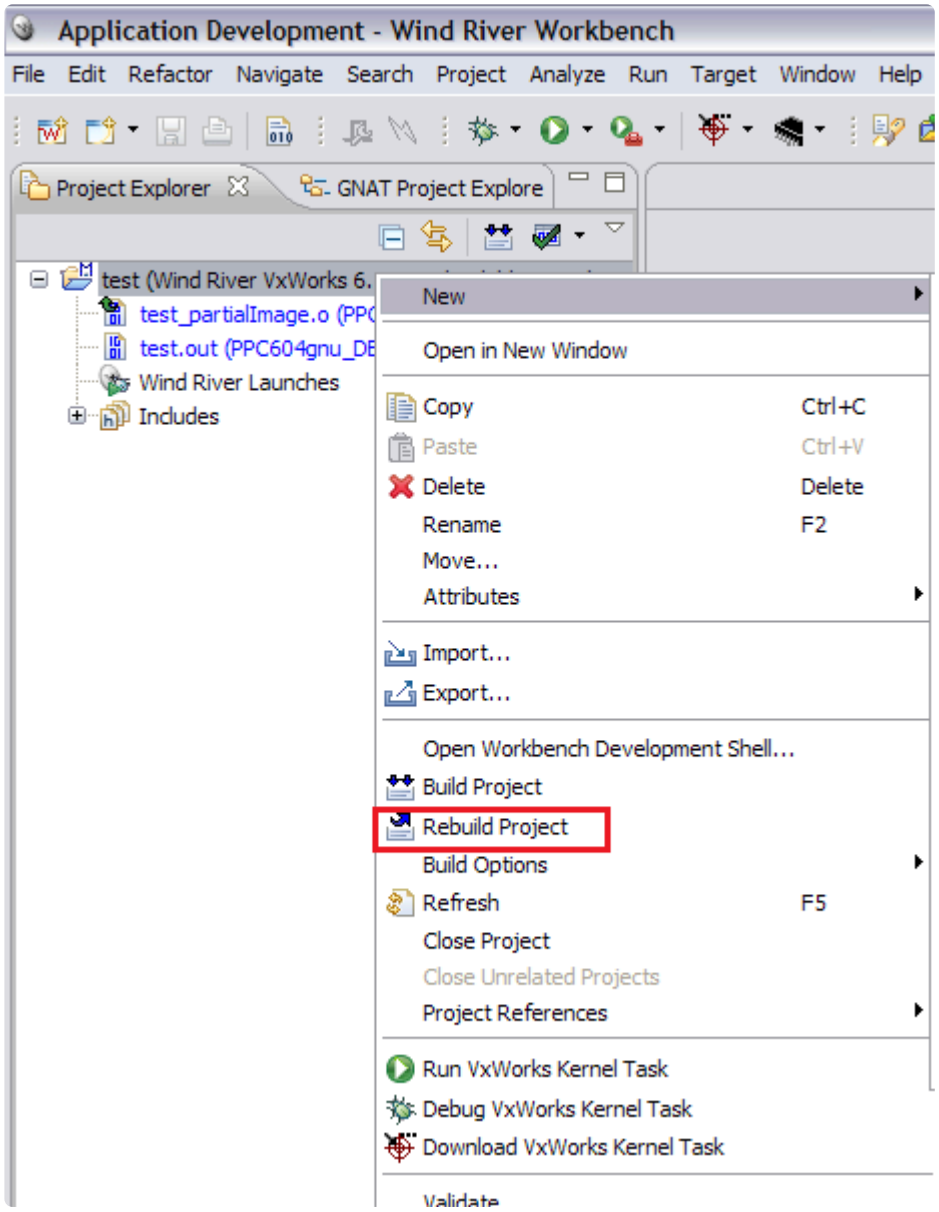


- 정확한 테스트를 위해서 [타깃 테스트 케이스 실행] 메뉴로 테스트 케이스 단위 실행합니다. 메모리 여유가 있을 경우, 여러 테스트 또는 함수를 한 번에 테스트할 수 있습니다.



- 타깃 테스트 실행 시 CT 2023.12의 프로젝트가 위 상태처럼 잠기게 됩니다.

3. 빌드



- 내보내기 된 코드를 Wind River Workbench IDE에서 Rebuild Project 합니다.

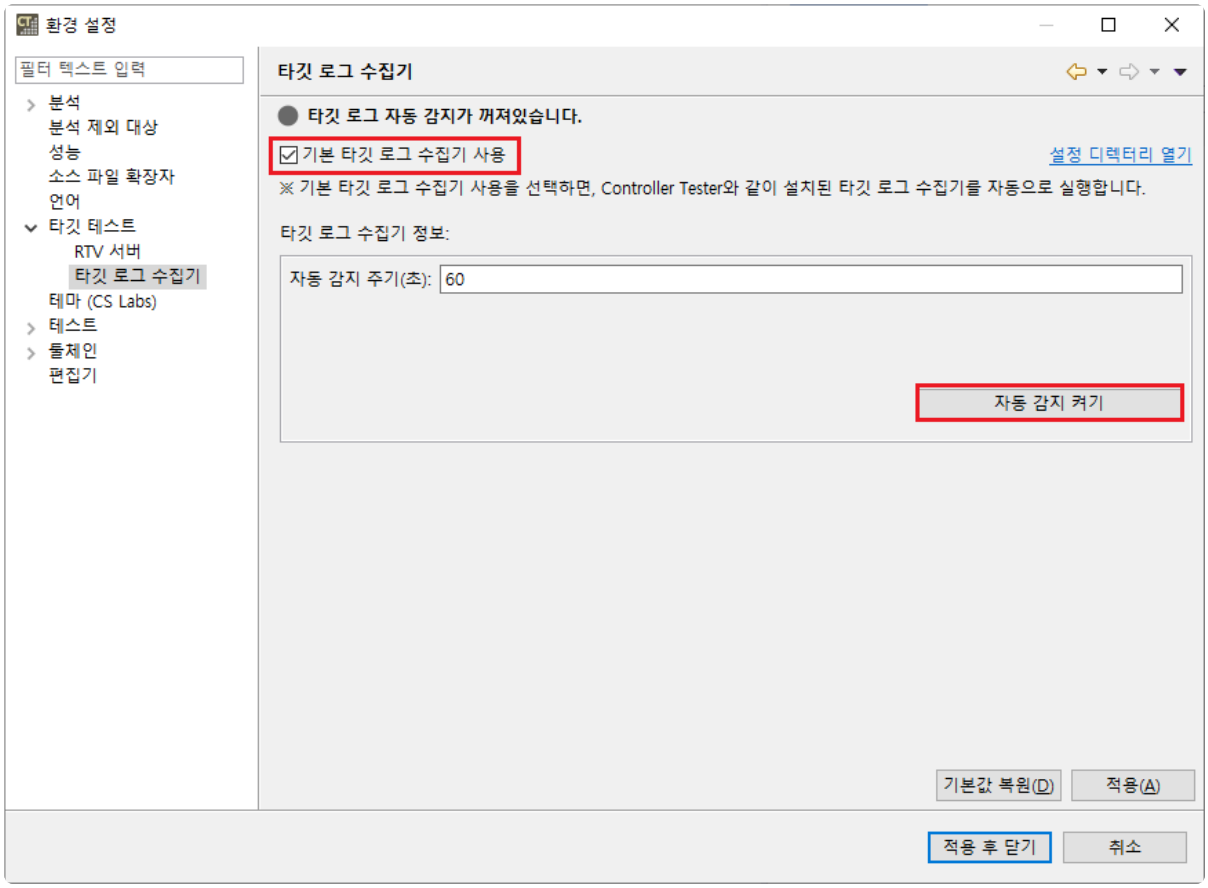
4. 타깃 로그 수집기 설정

- 로그 인터페이스에서 작성한 포트에 맞춰서 타깃 로그 수집기를 설정 후 실행합니다.
- 타깃 로그 수집기는 %appdata%\CodeScroll\TargetLogCollector\TargetLogCollector.exe에 설치되어 있습니다.
- cmd에서 타깃 로그 수집기를 최초 실행하면 %appdata%\CodeScroll\TargetLogCollector\setting.ini가 생성됩니다. 해당 파일로 타깃 로그 수집기 설정을 할 수 있습니다.
- 아래 예시는 로그 인터페이스 예시에 맞게 설정한 타깃 로그 수집기 설정입니다.

```
[LogReceiveServer]
; TCP, UDP server port
port=2019
; tcp, udp, uart or serial
protocol=tcp
; timeout(second)
timeout=60
```

```
lastString=CSET#
; serial(UART) port (Windows: COM#, Linux: /dev/ttyS#)
serialPort=COM1
; serial port setting
baudRate=9600
dataBits=8
stopBits=1
parity=0
flowControl=0

[ScanLog]
; log directory(default: scan/log)
dir=
; log file extension(if empty, scan everything)
fileExtension=log
; begin character when filtering the string of log (ascii code with value sepa
rator ;)
beginCharacter=4;5;6
; end character when filtering the string of log (ascii code with value separa
tor ; , 10 is LF)
endCharacter=10

[LogSendServer]
; send log to Controller Tester
port=2020
```

5. CT 2023.12의 타깃 로그 수집기 설정

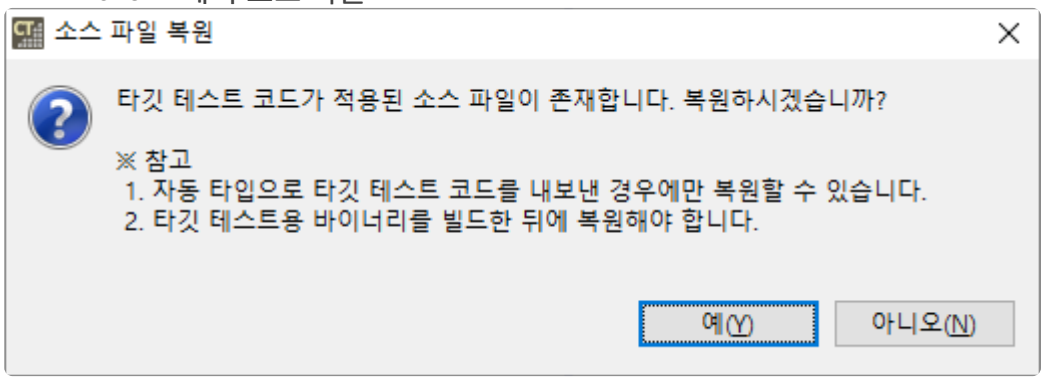


- CT 2023.12의 [환경설정] -> [타깃 테스트] -> [타깃 로그 수집기]에서 기본 타깃 로그 수집기 사용을 선택 후 자동 감지를 켭니다.

6. 타깃에서 바이너리 실행

- 빌드된 바이너리를 VxWorks OS에 옮겨서 타깃에 올려 실행합니다.

7. CT 2023.12에서 코드 복원



- 타깃 테스트 로그를 가져오기 위해 CT 2023.12에서 코드를 복원합니다.

8. 타깃 테스트 로그 가져오기

- 설정한 자동 감지 주기에 따라 작성된 로그를 자동으로 불러옵니다.

6.2. 디버거 사용 가이드

CT 2023.12 타깃 테스트 수행 시 디버거 사용 방법에 대한 가이드 문서입니다.

- Lauterbach TRACE32
- PLS Universal Debug Engine
- iSYSTEM winIDEA Debugger
- IAR Embedded Workbench C-SPY Debugger
- Texas Instruments Code Composer Studio
- Microchip MPLAB IDE

6.2.1. Lauterbach TRACE32

CT 2023.12는 TRACE32 디버거를 사용하여 타깃 테스트를 할 수 있습니다.
CT 2023.12는 TRACE32의 cmm 스크립트를 사용하여 타깃 환경에서 테스트를 실행하고 결과를 가져옵니다.
TRACE32에서 지원하는 타깃 목록은 Lauterbach 홈페이지 에서 확인할 수 있습니다.

- cmm 스크립트 자동 생성 지원 타깃 목록
- Step1: CT에서 타깃 환경 설정
- Step2: 타깃 테스트 실행

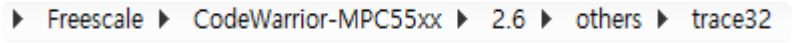
6.2.1.1. cmm 스크립트 자동 생성 지원 타깃 목록

CT 2023.12는 cmm 스크립트 파일을 자동으로 생성하거나 사용자에게 입력받습니다.
cmm 스크립트를 자동으로 생성할 수 있는 경우에는 타깃의 칩 이름만 입력하면 됩니다. cmm 스크립트를 자동으로 생성할 수 없는 경우에는 사용자가 직접 cmm 스크립트 파일 경로를 입력해야 합니다.
현재 cmm 스크립트 자동 생성을 지원하는 타깃은 아래와 같습니다.

PowerPC	mpc5554, mpc5553, mpc5534, mpc556x, mpc551x, mpc560xe, spc560bxx, spc560pxx, spc560sxx, mpc560xb, mpc560xp, mpc560xs, spc563m54, mpc5632m, spc563m60, mpc5633m, spc563m64, mpc5634m, mpc564xs, mpc5668, mpc5674, mpc5644a, spc564a80, mpc5642a, spc564a70, mpc567xk, spc56hk, mpc5643l, spc56el60, spc56el70, mpc5644b, mpc5644c, spc564b64, spc56ec64, mpc5645b, spc564b70, mpc5645c, spc56ec70, mpc5646b, spc564b74, mpc5646c, spc56ec74, mpc5676r, spc56ap, mpc5746m, mpc5744k, spc574k74, mpc5777m, spc57hm90, mpc574xp, mpc574xg, mpc574xr, mpc577xk, mpc5777c, spc570s, mpc5726l, spc572l, spc574s, spc58ne, spc58eg, spc58nn, spc582b, spc58ec, spc58nh, spc584b, s32r274, s32r264, s32r372
ARM	mkw01, mkw20, mkv30, mkv40, mkv10, mkv50, mkm30, mkl0, mkl10, mkl20, mkl30, mkl40, mkl80, mk0, mk10, mk20, mk30, mk40, mk50, mk60, mk70, mk80, mac57d54h, mac71×1, mac71×2, mac71×4, mac71×5, mac71×6, mac72×1, lpc51u68, lpc54xx, lpc8xx, lpc11xx, lpc12xx, lpc13xx, lpc17xx, lpc18xx, lpc21xx, lpc22xx, lpc23xx, lpc24xx, lpc28xx, lpc29xx, lpc40xx, lpc43xx, imxrt1064, xmc1100, xmc1200, xmc1300, xmc1400, xmc4100, xmc4200, xmc4300, xmc4400, xmc4500, xmc4700, xmc4800, tle98, s3fm02g, s32k, s6e1a, s6e1c, s6j3
tricore	tc2dx, tc21x, tc22x, tc23x, tc26x, tc27x, tc29x, tc35x, tc37x, tc38x, tc39x, tc116x, tx1167, tx1197, tc1724, tc1728, tc1736, tc1762, tc1764, tc1766, tc1767, tc1782, tc1784, tc1791, tc1792, tc1793, tc1796, tc1797, tc1798

6.2.1.2. Step1: CT에서 타깃 환경 설정

CT 2023.12의 타깃 환경 설정 페이지에서 디버거를 선택합니다. 프로젝트에 선택된 툴체인에 따라 지원하는 디버거 목록만 표시됩니다.
디버거를 TRACE32로 설정합니다.



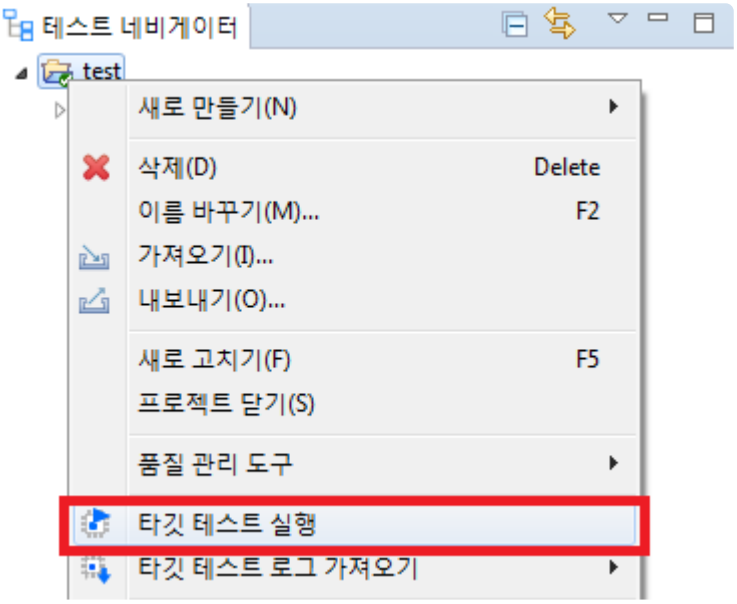
선택한 정보에 따라 설정 항목들이 표시됩니다. TRACE32 디버거를 사용하는 경우에 설정해야하는 항목은 아래 표와 같습니다.
설정 중 일부는 필수 항목입니다.

trace32_exe_file_path	TRACE32 실행파일 경로입니다. 타깃마다 실행파일이 다르기 때문에, 사용하는 타깃의 실행파일이 맞는지 확인해야 합니다. 필수 항목입니다.
target_binary_path	타깃 환경에 로드하기 위한 바이너리 파일 경로입니다. 사용하는 IDE 또는 빌드 스크립트에서 타깃 바이너리 파일이 생성되는 경로를 확인하고 입력합니다. 필수 항목입니다.
chip	사용하는 타깃의 칩 이름을 입력합니다. cmm 스크립트를 자동 생성할 때 사용되기 때문에 정확한 칩 이름을 입력해야 합니다.
user_defined_cmm_script_file_path	사용자 정의 cmm 스크립트 파일 경로입니다. cmm 스크립트 자동 생성을 지원하지 않는 타깃인 경우에는 디버거와 타깃 사용 환경을 설정하는 스크립트를 작성하거나, 사용하고 있는 cmm 스크립트 파일 경로를 입력해야 합니다.

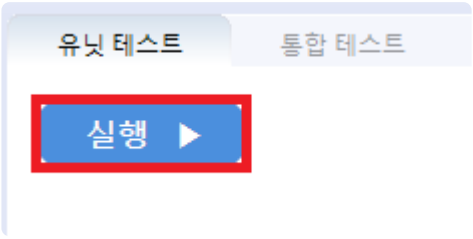
6.2.1.3. Step2: 타깃 테스트 실행

타깃 테스트를 실행하기 전에 실행 중인 TRACE32 프로그램을 종료해야 합니다.
테스트 네비게이터 뷰의 프로젝트 컨텍스트 메뉴에서 [타깃 테스트 실행]을 선택하거나 테스트 뷰의 [실행] 버튼을 클릭하여, 타깃 테스트를 실행할 수 있습니다.

- [타깃 테스트 실행]



- [실행]



타깃 테스트를 실행할 때 TRACE32 프로그램이 실행됩니다. 정상적으로 테스트가 완료되면 TRACE32 프로그램이 자동으로 종료됩니다.

6.2.1.4. 디버거를 통해 타깃 테스트 디버깅

1. 타깃으로 설정 후 유닛 테스트 뷰에서 테스트 케이스 우클릭 후 ‘디버그 정보 확인’ 클릭
2. 사용자 프로젝트를 직접 빌드 혹은 Controller Tester 프로젝트에서 ‘타깃 환경’ 설정에 등록한 빌드 스크립트 수행
3. 빌드 성공하는지 확인
4. CT 2023.12에서 프로젝트를 열어 원본 소스 복원
5. Trace32 실행 후 cmm 스크립트 파일(start.cmm)을 열어 ‘debug’를 실행(CT_프로젝트_경로/.csdata/target/start.cmm)
6. ‘step’ 버튼을 클릭하면 target.cmm 스크립트 첫 라인으로 이동
7. target.cmm 파일의 Go.Hll에 breakpoint 지정
8. Trace32의 Var > Show Function 클릭
9. 테스트 대상 함수 검색 후 더블 클릭
10. 함수 시작부에 breakpoint 지정
11. ‘step’ 버튼을 클릭하면 디버깅 포인트가 10번에서 지정한 위치로 이동하는 것을 확인
12. Var > Show Local…. 클릭하면 지역변수의 값이 변하는 것을 확인 가능
13. 디버깅 포인트까지 실행

6.2.2. PLS Universal Debug Engine (UDE)

CT 2023.12는 UDE 디버거를 사용하여 타깃 테스트를 할 수 있습니다.
CT 2023.12는 UDE에서 지원하는 디버깅 스크립트를 사용하여 타깃 환경에서 테스트를 실행하고 결과를 가져옵니다.
UDE에 연결하여 사용할 수 있는 타깃 목록은 PLS 홈페이지 에서 확인할 수 있습니다.

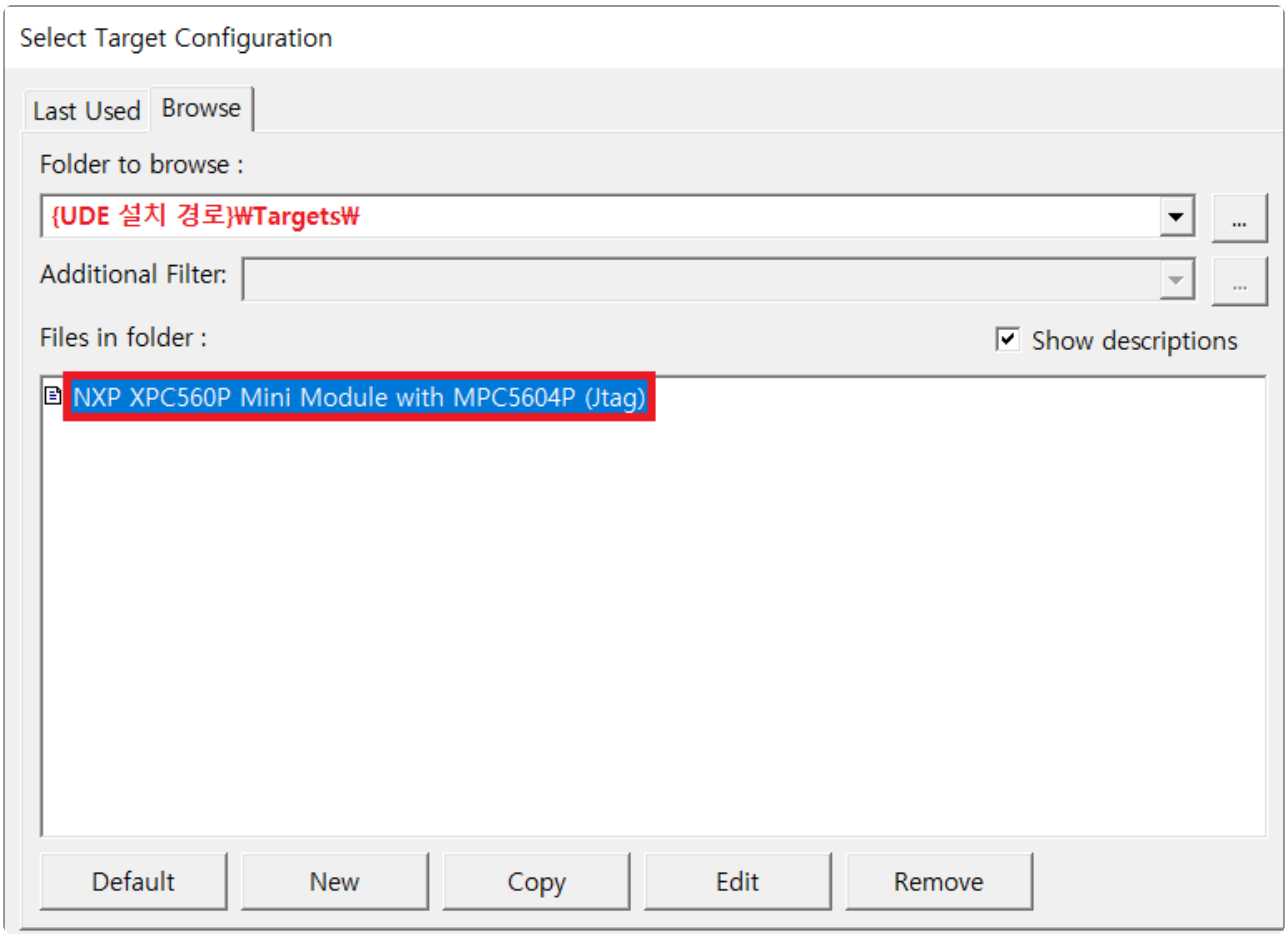
CT 2023.12에서는 UDE 워크스페이스 정보를 사용하여 타깃 테스트를 수행합니다. 그렇기 때문에 사용자는 타깃 테스트 수행 전에 먼저 워크스페이스를 생성해야 합니다.

- Step1: UDE IDE에서 워크스페이스 생성
- Step2: CT에서 타깃 환경 설정
- Step3: 타깃 테스트 실행

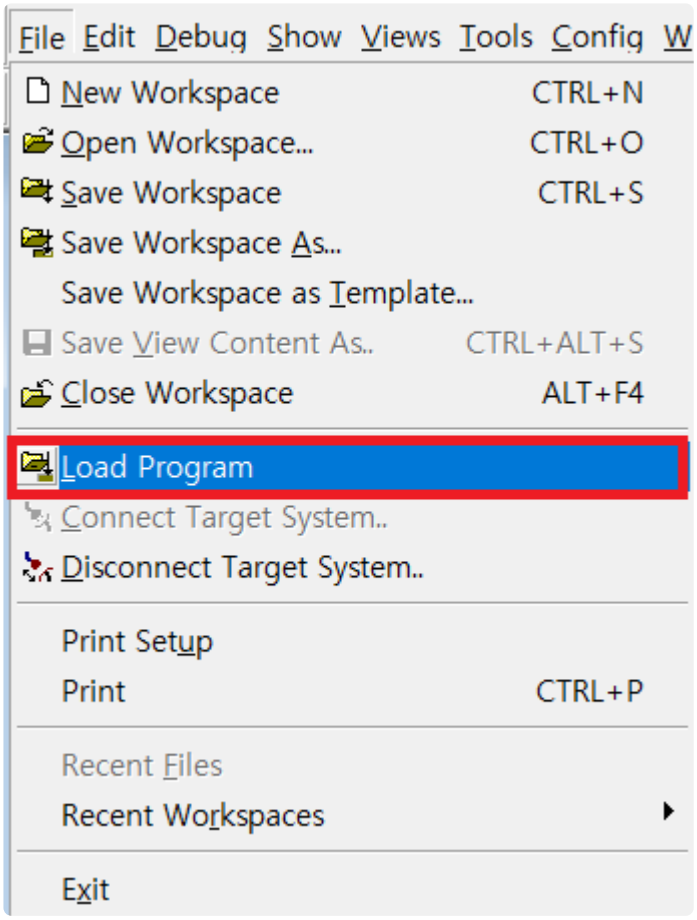
6.2.2.1. Step1: UDE IDE에서 워크스페이스 생성

UDE는 UDE desktop IDE에서 UDE 워크스페이스를 생성할 수 있습니다.

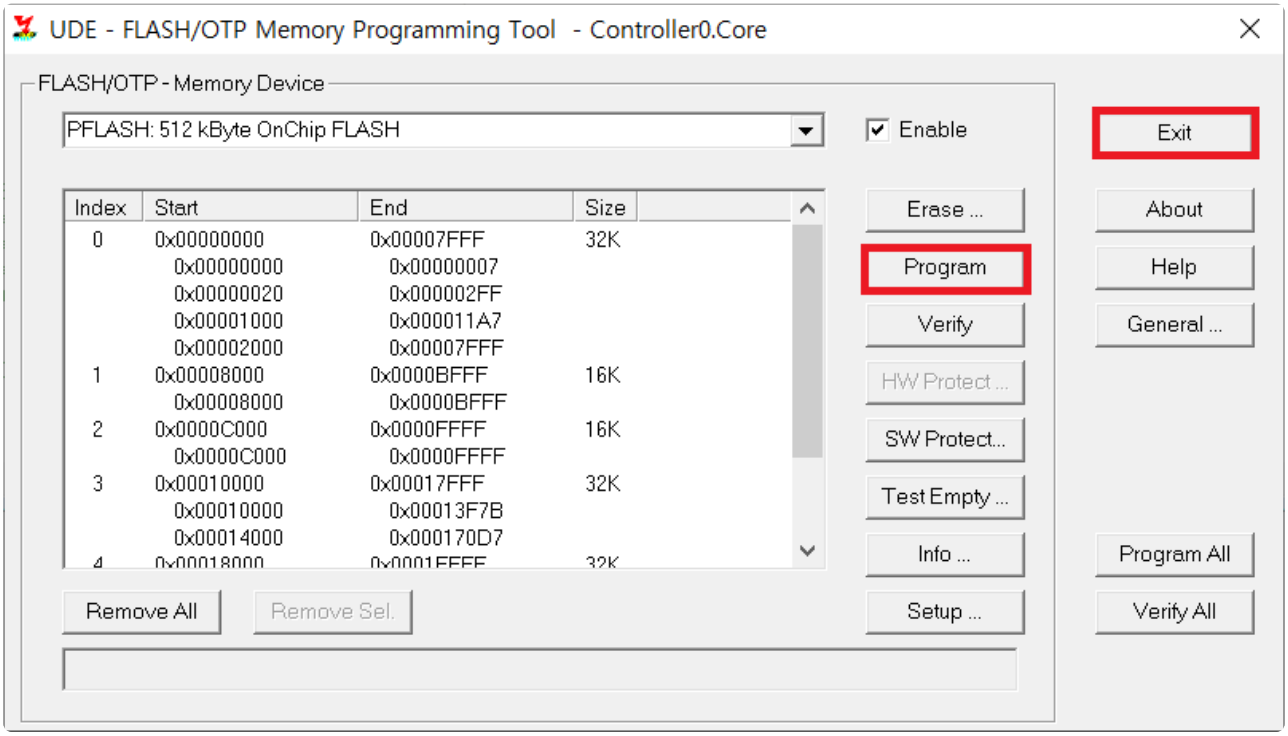
1. 사용하는 타깃에 맞는 설정 파일을 선택하여 워크스페이스를 생성합니다.



2. 상단 메뉴에서 [File] > [Load Program] 버튼을 클릭하면 바이너리 파일을 로드할 수 있습니다. 이때 테스트 코드로부터 빌드된 바이너리 파일을 선택합니다.



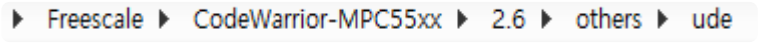
3. 이후 안내에 따라 [Program] 버튼을 누르면 타깃 설정에 따라 바이너리 파일이 타깃에 로드됩니다. 로드가 정상적으로 끝나면 워크스페이스 설정이 완료됩니다. [Exit]를 클릭하여 창을 빠져나옵니다.



자세한 사항은 UDE에서 제공하는 매뉴얼을 참조하십시오.

6.2.2.2. Step2: CT에서 타깃 환경 설정

CT 2023.12의 타깃 환경 설정 페이지에서 디버거를 선택합니다. 프로젝트에 선택된 툴체인에 따라 지원하는 디버거 목록만 표시됩니다.
디버거를 UDE 로 설정합니다.



선택한 정보에 따라 설정 항목들이 표시됩니다. UDE 디버거를 사용하는 경우에 설정해야하는 항목은 아래 표와 같습니다.
설정 중 일부는 필수 항목입니다.

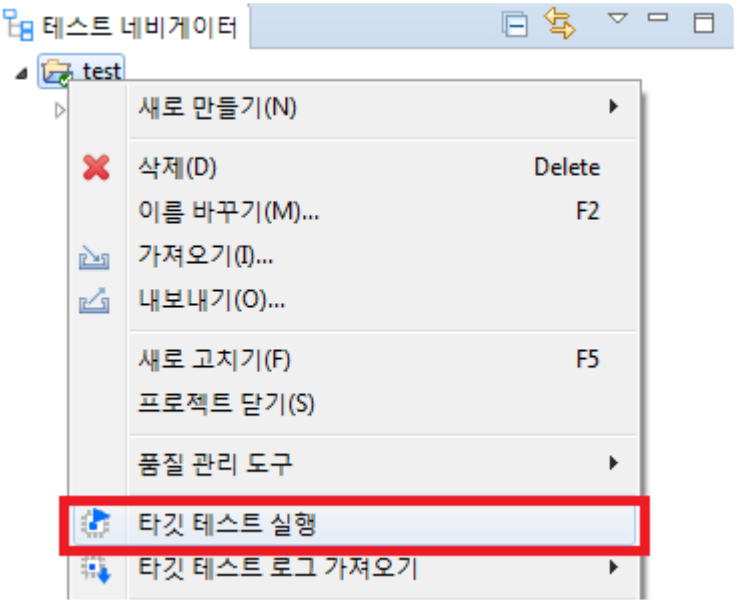
target_binary_path	타깃 환경에 로드하기 위한 바이너리 파일 경로입니다. 사용하는 IDE 또는 빌드 스크립트에서 타깃 바이너리 파일이 생성되는 경로를 확인하고 입력합니다. 필수 항목입니다.
ude_project_file	UDE IDE에서 생성된 워크스페이스 프로젝트 파일(.wsx) 경로입니다. 필수 항목입니다.

CT 2023.12에서 사용하는 기본 스크립트 언어는 visual basic script입니다.
타깃 환경 설정이 끝나면 [OK] 또는 [Finish]버튼을 클릭합니다. 타깃 테스트를 수행할 준비가 끝났습니다.

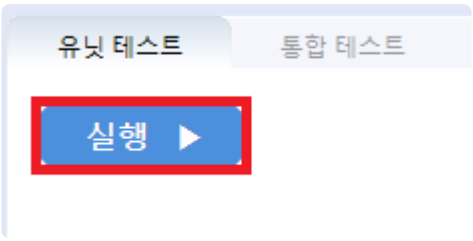
6.2.2.3. Step3: 타깃 테스트 실행

타깃 테스트를 실행하기 위해서는 UDE IDE를 종료한 상태여야 합니다.
테스트 네비게이터 뷰의 프로젝트 컨텍스트 메뉴에서 [타깃 테스트 실행]을 선택하거나 테스트 뷰의 [실행] 버튼을 클릭하여, 타깃 테스트를 실행할 수 있습니다.

- [타깃 테스트 실행]



- [실행]



> UDE 디버깅 스크립트는 C++, .NET, Perl 등의 언어로 작성할 수 있습니다. 스크립트로 작성할 수 있는 다른 지원 언어들에 대해서는 UDE 매뉴얼에 포함되는 UDE Automation Basics 설명서를 확인하십시오.

6.2.2.4. 디버거를 통해 타깃 테스트 디버깅

1. 타깃으로 설정 후 유닛 테스트 뷰에서 테스트 케이스 우클릭 후 ‘디버그 정보 확인’ 클릭
2. 사용자 프로젝트를 직접 빌드 혹은 CT 프로젝트에서 ‘타깃 환경’ 설정에 등록한 빌드 스크립트 수행
3. 빌드 성공하는지 확인
4. CT에서 프로젝트를 열어 원본 소스 복원
5. Pls Ude 실행 후 프로젝트 선택(.wsx파일)
6. 2번에서 빌드된 output파일을 선택
7. 좌측 네비게이션에 output 파일에 들어 있는 소스 파일과 함수 정보가 표시되는 것을 확인
8. 테스트 대상 함수가 있는 소스파일 선택 후 함수에 디버깅 포인트 지정
9. F5버튼을 누르면 entry point에서부터 실행

6.2.3. iSYSTEM winIDEA Debugger

CT 2023.12는 winIDEA 디버깅 스크립트를 통해 자동으로 타깃 환경에서 테스트를 실행하고 결과를 가져오는 기능을 제공합니다.
winIDEA가 지원하는 타깃 목록은 iSYSTEM 홈페이지 에서 확인할 수 있습니다.

디버깅 스크립트의 실행은 winIDEA 설치 시 함께 설치되는 python SDK를 필요로 합니다. 만일 설치되어 있지 않은 경우 iSYSTEM SDK 설치 페이지 에서 다운로드받을 수 있습니다. 또한, 사용하는 winIDEA 버전이 해당 SDK를 지원하고 있는지 확인해야 합니다. Controller Tester에서 기본으로 제공하는 디버깅 스크립트는 python 3.3 버전을 기준으로 합니다.

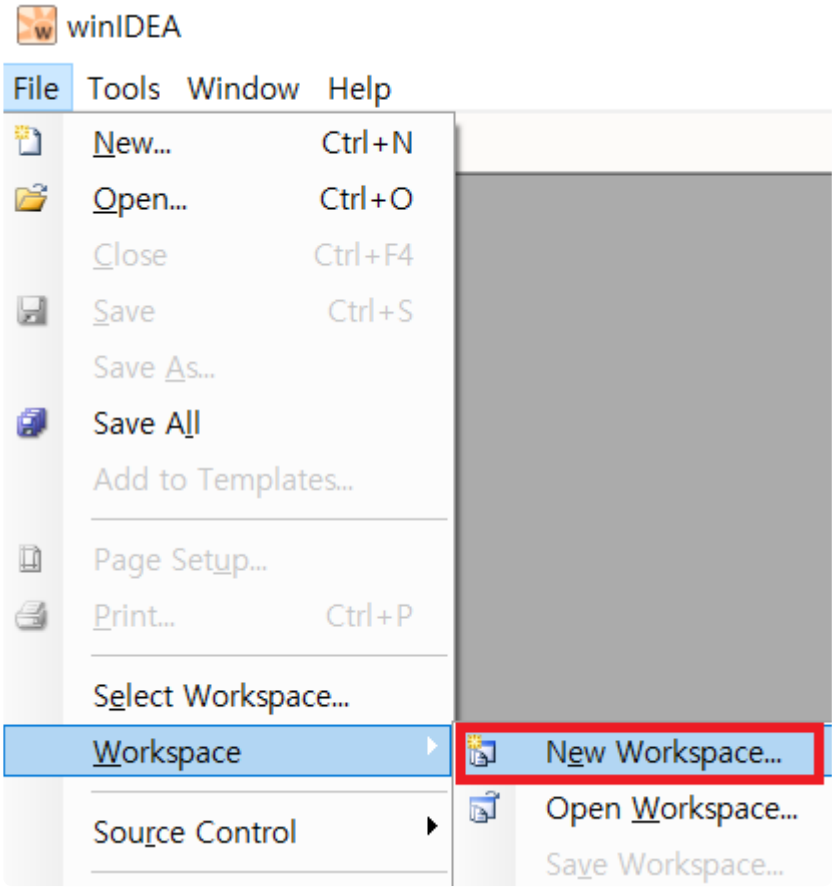
이 문서에서는 winIDEA에서 프로젝트를 생성하는 것부터 CT 2023.12에서 타깃 테스트를 실행하기까지의 과정을 예시와 함께 설명합니다.
예시에서는 iSYSTEM BlueBox iC5000 Unit 디버거와 NXP의 MPC56xx 타깃을 사용합니다.

6.2.3.1. iSYSTEM winIDEA 를 사용하기 위한 사전 준비

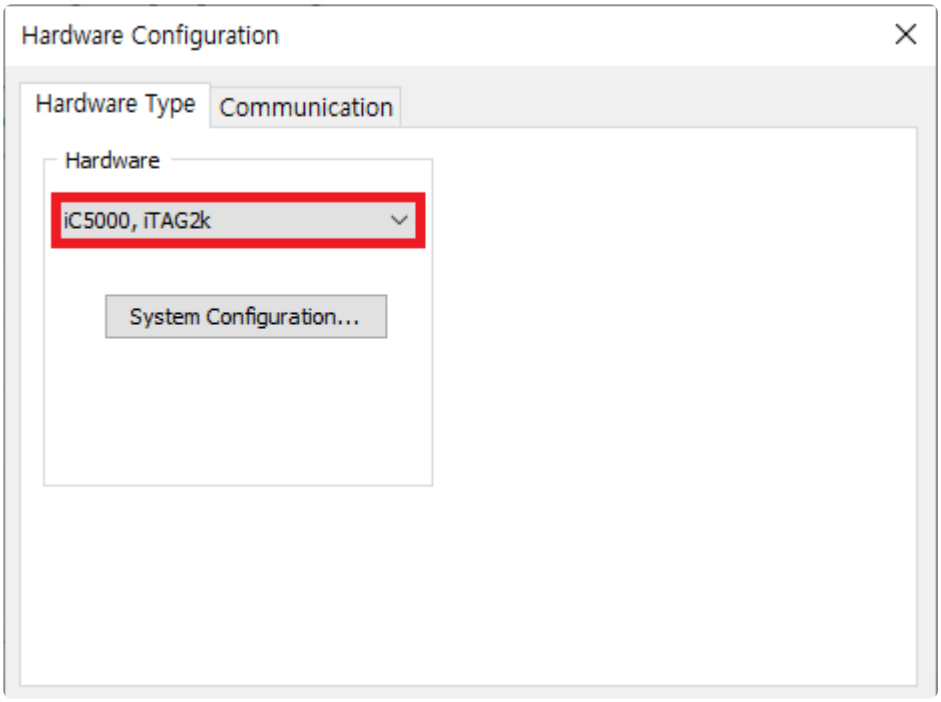
CT 2023.12에서 winIDEA를 사용하여 타깃 테스트를 하려면 winIDEA와 연결하여 사용 가능한 디버거가 필요합니다.
사용자는 타깃 테스트 수행 전에 winIDEA 워크스페이스를 생성하고 사용할 디버거를 CT 2023.12가 설치된 PC와 연결해야 합니다.

6.2.3.2. Step1: winIDEA 워크스페이스 생성 및 설정

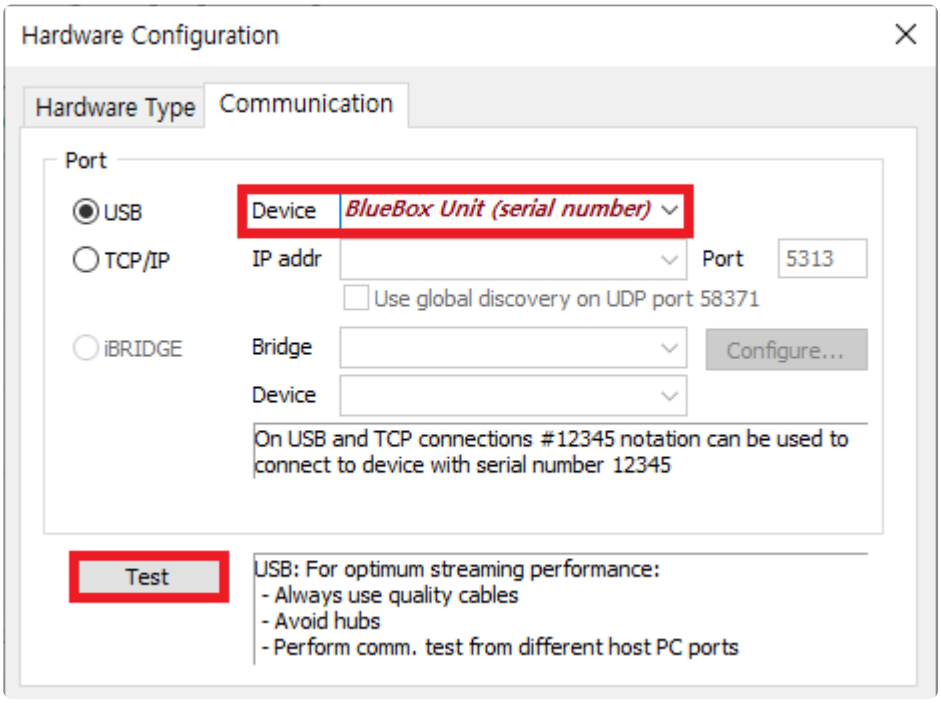
1. winIDEA 실행 후, 상단 메뉴에서 [File] > [Workspace] > [New Workspace…]를 선택하여 새로운 워크스페이스를 생성합니다. 생성한 워크스페이스를 CT 타깃 테스트에 사용하기 위해서는 추가 환경 설정이 필요합니다.



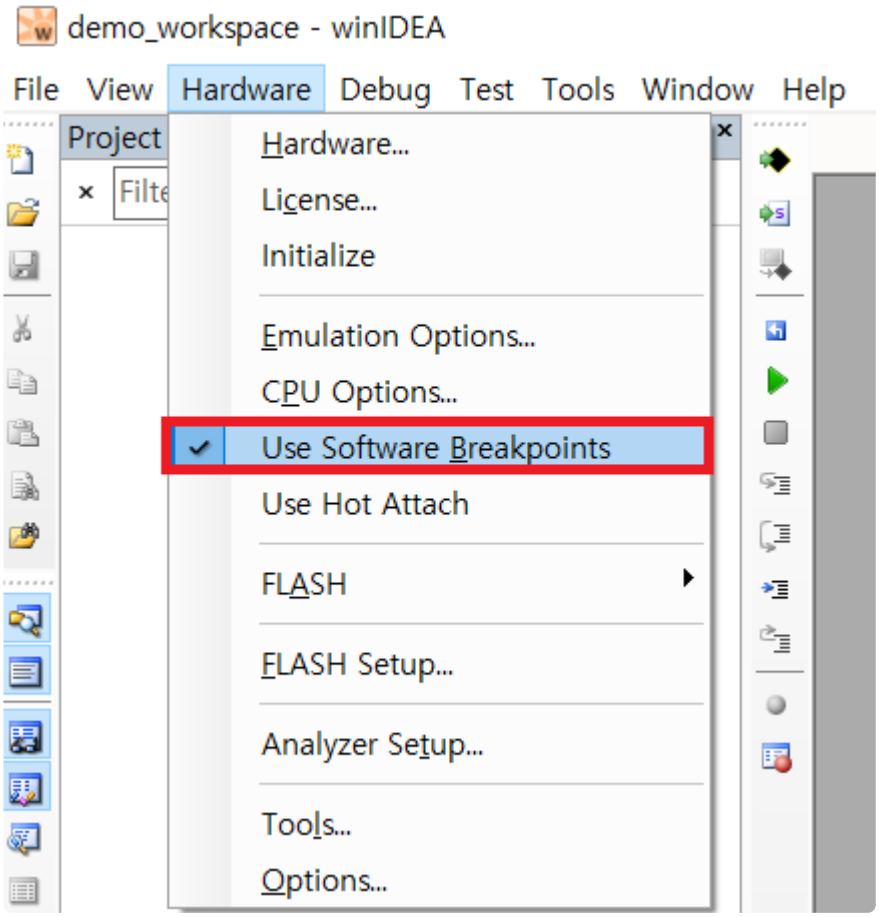
2. 첫 번째로 상단 메뉴에서 [Hardware] > [Hardware…] 를 클릭한 후, [Hardware Type] 탭에서 연결된 BlueBox의 종류를 선택합니다.

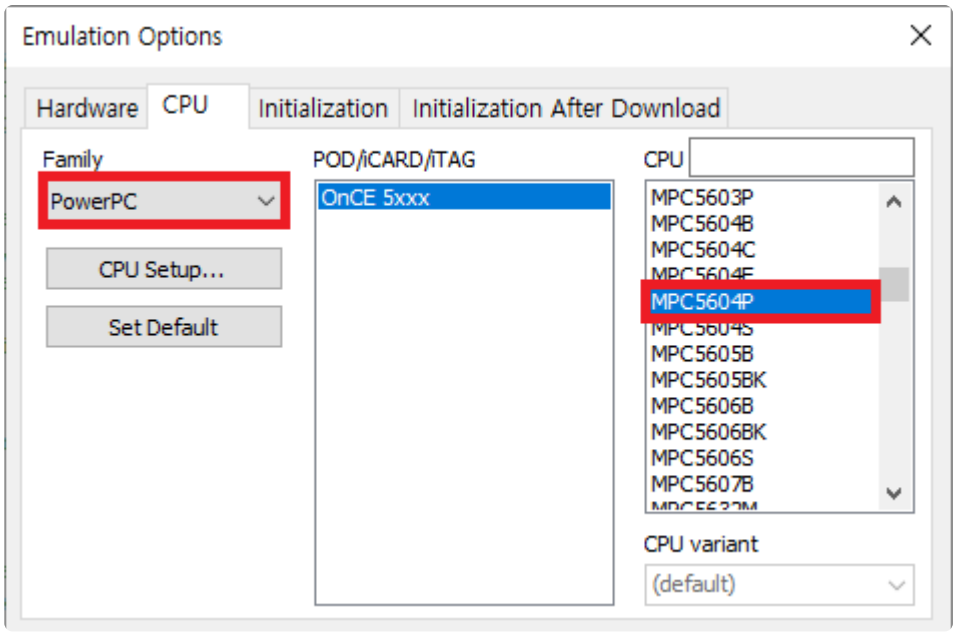


3. 다음으로 [Communication] 탭에서 통신 방식 설정하고 [Test] 버튼을 눌러 기기 연결을 확인합니다. 통신 방식에 따른 디버거 장비 연결 방법에 대한 설명은 iSYSTEM BlueBox 매뉴얼을 참조하십시오.



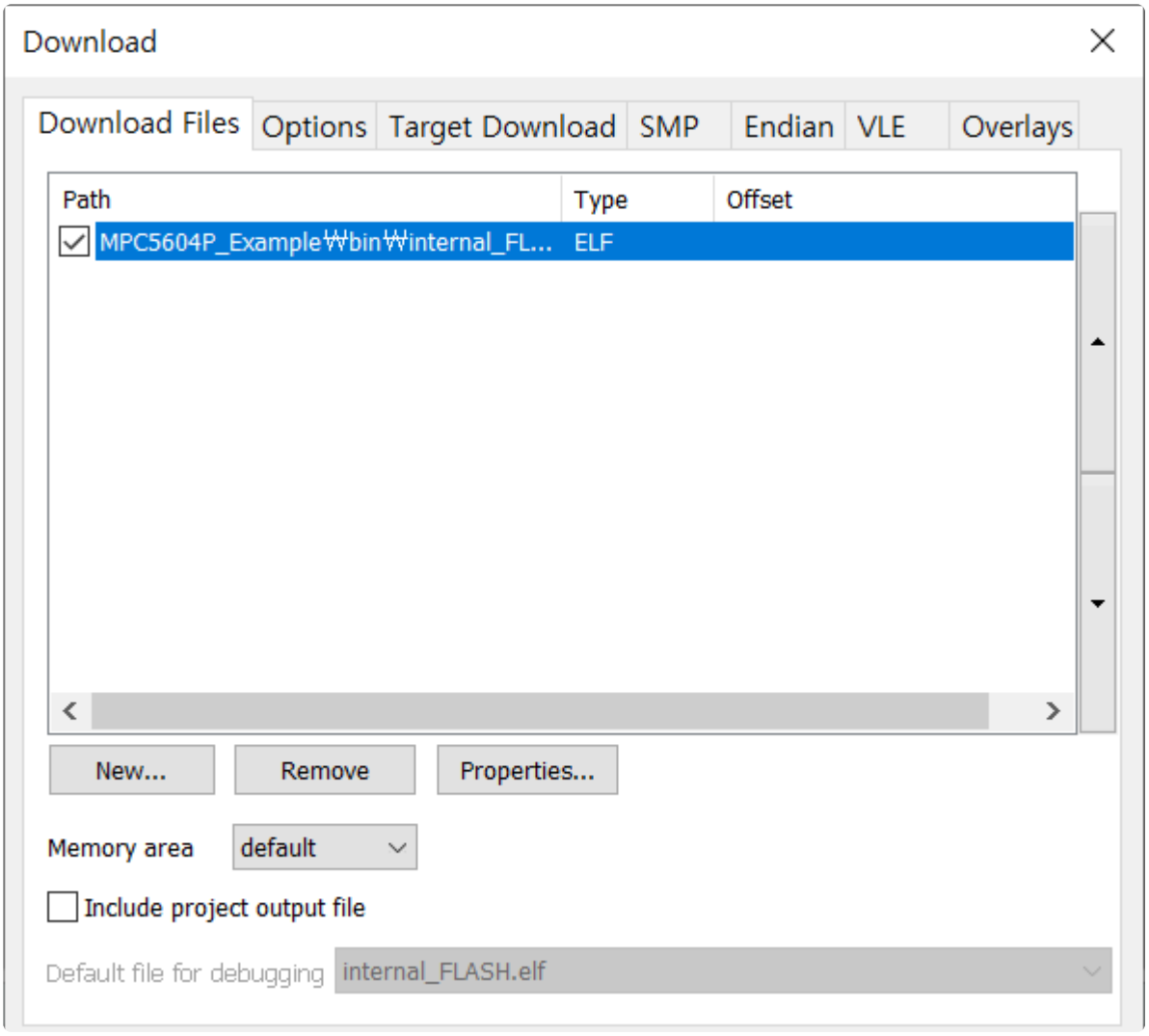
4. 상단 메뉴 중 [Hardware] > [Use Software Breakpoints] 를 클릭해서 활성화시킨 후, [Hardware] > [Emulation Options…] 의 [CPU] 항목에서 사용할 타깃 종류를 선택합니다.



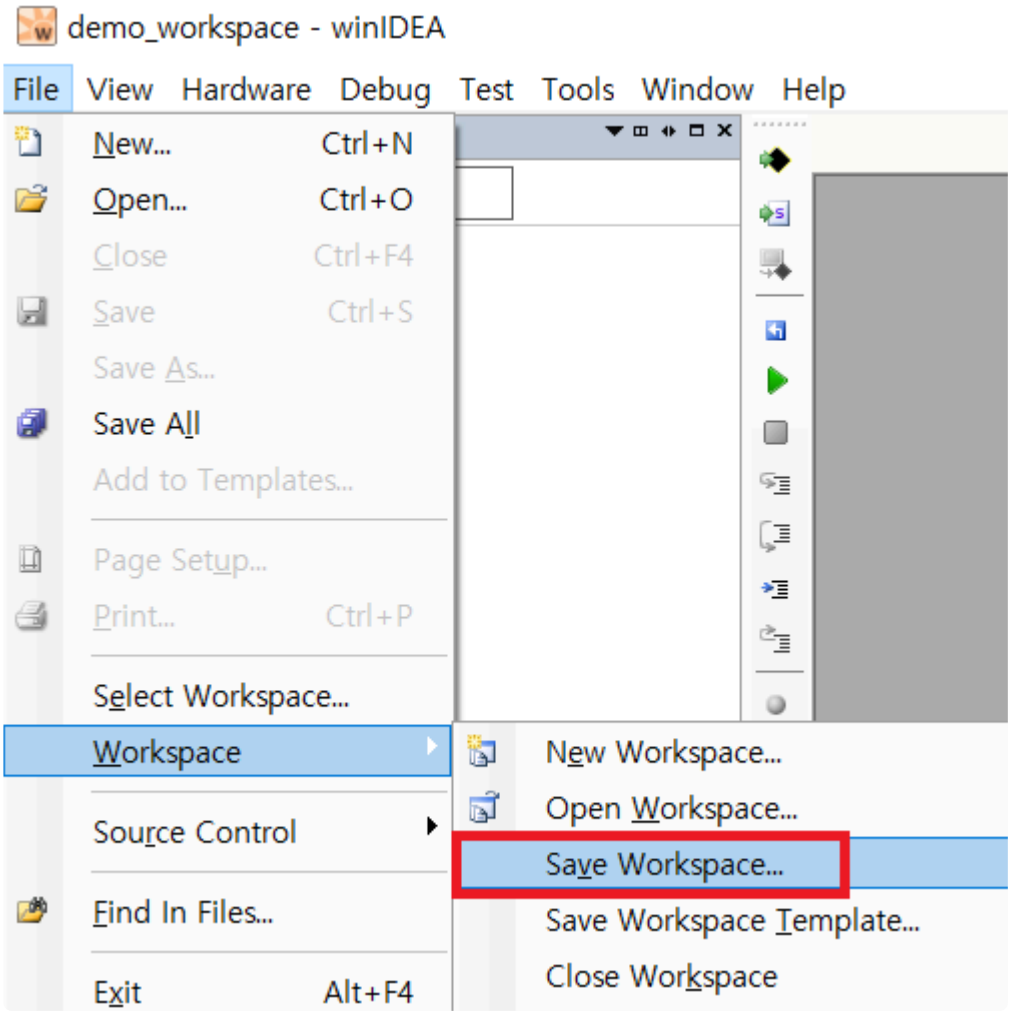


타깃 별로 설정해야 하는 세부 옵션이 다를 수 있습니다.

5. 기기 설정을 끝내고 나면 테스트할 바이너리 경로를 등록해야 합니다. 먼저 테스트 대상 소스 코드를 빌드하여 바이너리를 생성합니다. 그런 다음 winIDEA의 상단 메뉴 [Debug] > [Files for Download…] 에서 [New…]를 선택하고 생성한 바이너리 경로를 추가해줍니다.



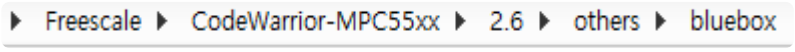
6. 워크스페이스 설정이 끝나고, 워크스페이스를 저장하여 winIDEA 워크스페이스 파일(.xjrf)을 생성합니다. 워크스페이스 파일은 CT 2023.12에서 winIDEA를 이용한 타깃 테스트 설정 시 사용됩니다.



이제 타깃 테스트를 하기 위한 winIDEA 워크스페이스 생성이 끝났습니다.

6.2.3.3. Step2: CT에서 타깃 환경 설정

CT 2023.12의 타깃 테스트 프로젝트 생성 마법사 또는 프로젝트 프로퍼티의 타깃 환경 설정에서 디버거를 선택합니다. 프로젝트 생성 시 선택한 툴체인에 따라 선택 가능한 디버거 목록이 달라집니다.
디버거를 BlueBox 로 설정합니다.



선택한 정보에 따라 설정 항목들이 표시됩니다. BlueBox를 사용하는 경우에 설정해야 하는 항목은 아래 표와 같습니다.
필수로 입력해야 하는 항목은 CT 2023.12에서 붉은 색으로 표시됩니다.

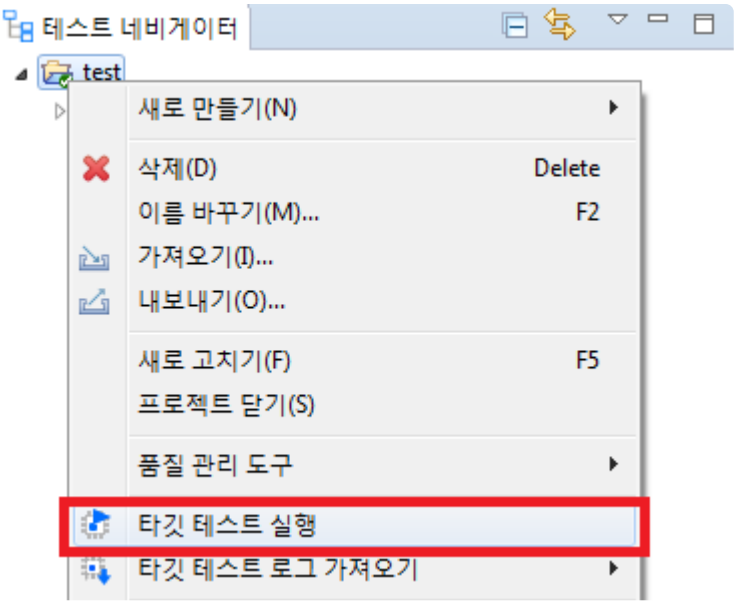
winidea_binary_path	winIDEA 실행 파일(winIDEA.exe) 경로입니다. 필수 항목입니다.
winidea_workspace_file_path	winIDEA에서 생성한 워크스페이스 파일(.xjrf) 경로입니다. 필수 항목입니다.

Controller Tester에서 사용하는 기본 스크립트 언어는 python입니다. 사용자 정의 디버깅 스크립트를 사용하는 경우, python으로 작성해야 원활한 타깃 테스트가 가능합니다. 다른 언어를 사용하여 디버깅 스크립트를 작성하는 경우에는 iSYSTEM 홈페이지 를 참고하여 추가 SDK 설치를 진행해야 합니다.
타깃 환경 설정이 끝나면 [OK] 또는 [Finish]버튼을 클릭합니다. 타깃 테스트를 수행할 준비가 끝났습니다.

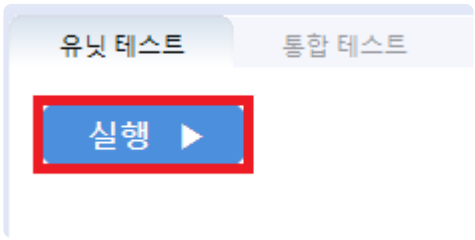
6.2.3.4. Step3: 타깃 테스트 실행

테스트 네비게이터 뷰의 프로젝트 컨텍스트 메뉴에서 [타깃 테스트 실행]을 선택하거나 테스트 뷰의 [실행] 버튼을 클릭하여, 타깃 테스트를 실행할 수 있습니다.

- [타깃 테스트 실행]



- [실행]



winIDEA가 실행 중인 상태에서는 타깃 테스트를 수행할 수 없습니다. CT 2023.12에서 타깃 테스트를 실행하기 전에 반드시 winIDEA를 종료해야 합니다.

6.2.3.5. 디버거를 통해 타깃 테스트 디버깅

1. 타깃으로 설정 후 유닛 테스트 뷰에서 테스트 케이스 우클릭 후 ‘디버그 정보 확인’ 클릭
2. 사용자 프로젝트를 직접 빌드 혹은 CT 프로젝트에서 ‘타깃 환경’ 설정에 등록한 빌드 스크립트 수행
3. 빌드 성공하는지 확인
4. CT에서 프로젝트를 열어 원본 소스 복원
5. winIDEA 실행 후 빌드한 프로젝트 포함하는 워크스페이스 선택 (.xjrf파일)
6. [Debug] > [Download] 선택하여 바이너리 파일 타깃에 다운로드
7. 상단에 Run 버튼을 누르면 디버깅 모드
8. [Project] > [Functions] 항목 더블 클릭하여 해당 함수 위치로 이동 후, 원하는 곳에 디버깅 포인트 설정
9. F5 눌러 디버깅 진행

6.2.4. IAR Embedded Workbench C-SPY Debugger

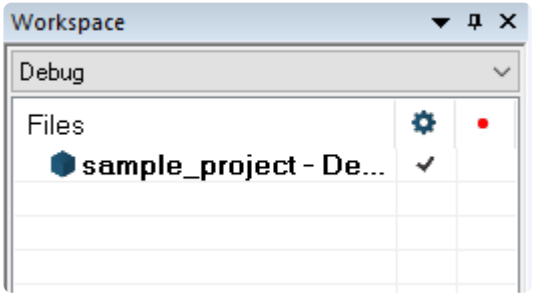
CT 2023.12는 IAR Embedded Workbench C-SPY 디버깅 기능을 통해 자동으로 타깃 환경에서 테스트를 실행하고 결과를 가져오는 기능을 제공합니다.
C-SPY에서 지원하는 타깃 목록은 IAR 홈페이지 에서 확인할 수 있습니다.

CT 2023.12에서 IAR Embedded Workbench C-SPY로 타깃 테스트를 하려면 C-SPY 디버깅 기능을 사용할 수 있는 디버깅 프로브가 필요합니다. 사용자는 타깃 테스트 수행 전에 IAR Embedded Workbench 프로젝트를 생성하고 사용할 디버깅 프로브를 CT 2023.12가 설치된 PC와 연결해야 합니다.
IAR에서 제공하는 디버깅 프로브 목록은 IAR 홈페이지 에서 확인할 수 있습니다.

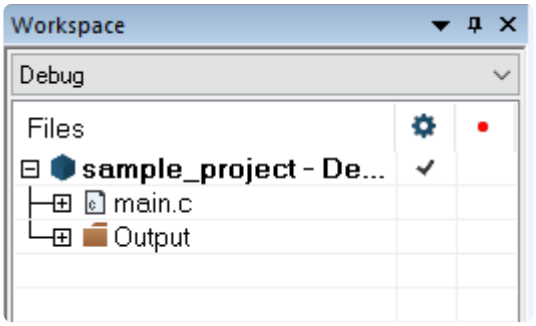
- Step1: IAR Embedded Workbench 프로젝트 생성
- Step2: IAR 프로젝트 설정
- Step3: CT에서 타깃 환경 설정
- Step4: 타깃 테스트 실행

6.2.4.1. Step1: IAR Embedded Workbench 프로젝트 생성

1. [File] > [New Workspace]를 클릭하여 새 워크스페이스를 생성한 후 [Project] > [Create New Project...]를 클릭하여 프로젝트 파일(.ewp)을 생성합니다. 프로젝트 파일을 생성하고 나면 IAR Embedded Workbench의 [Workspace] 뷰에 프로젝트 이름이 표시됩니다.



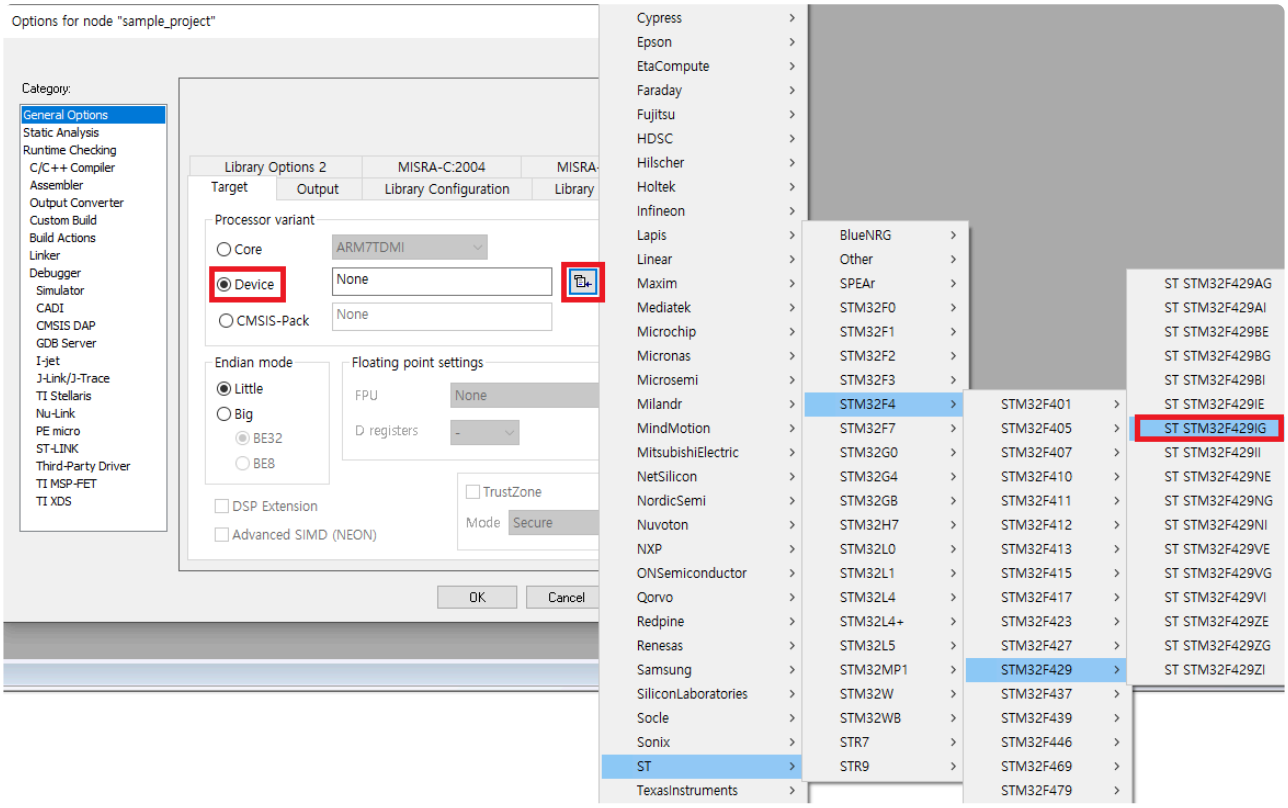
2. 다음으로 프로젝트에 시험 대상 소스 파일을 추가해야 합니다. 프로젝트를 우클릭을 하여 [Add] > [Add Files...]를 선택하고, 시험 대상 소스 파일을 추가합니다. 추가 후 [Workspace] 뷰에서 계층 구조로 추가된 소스 파일을 확인할 수 있습니다.



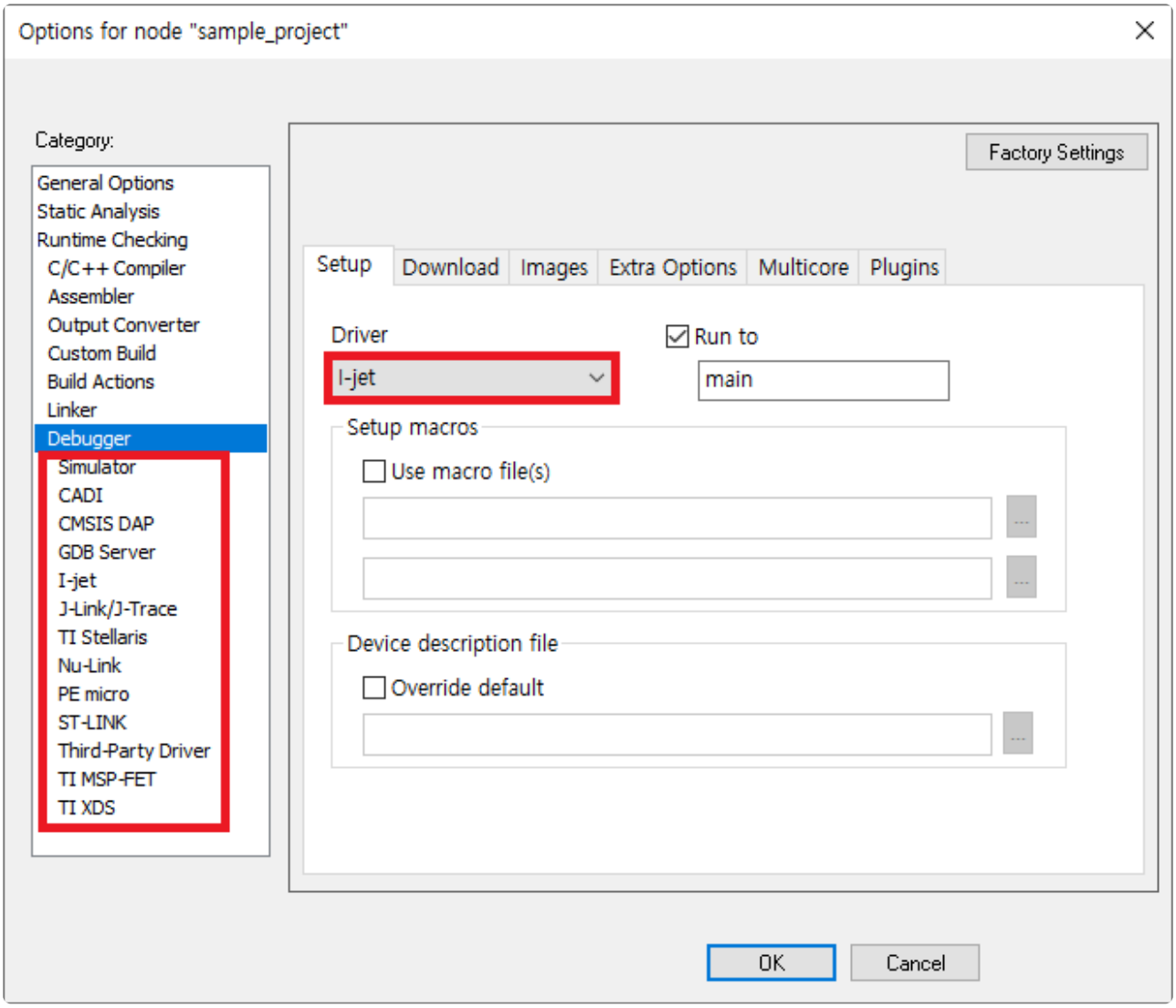
6.2.4.2. Step2: IAR 프로젝트 설정

프로젝트를 생성했다면 C-SPY 디버깅 기능 사용을 위한 프로젝트 설정을 해야 합니다. 생성한 프로젝트를 우클릭하여 [Options...]를 선택합니다.

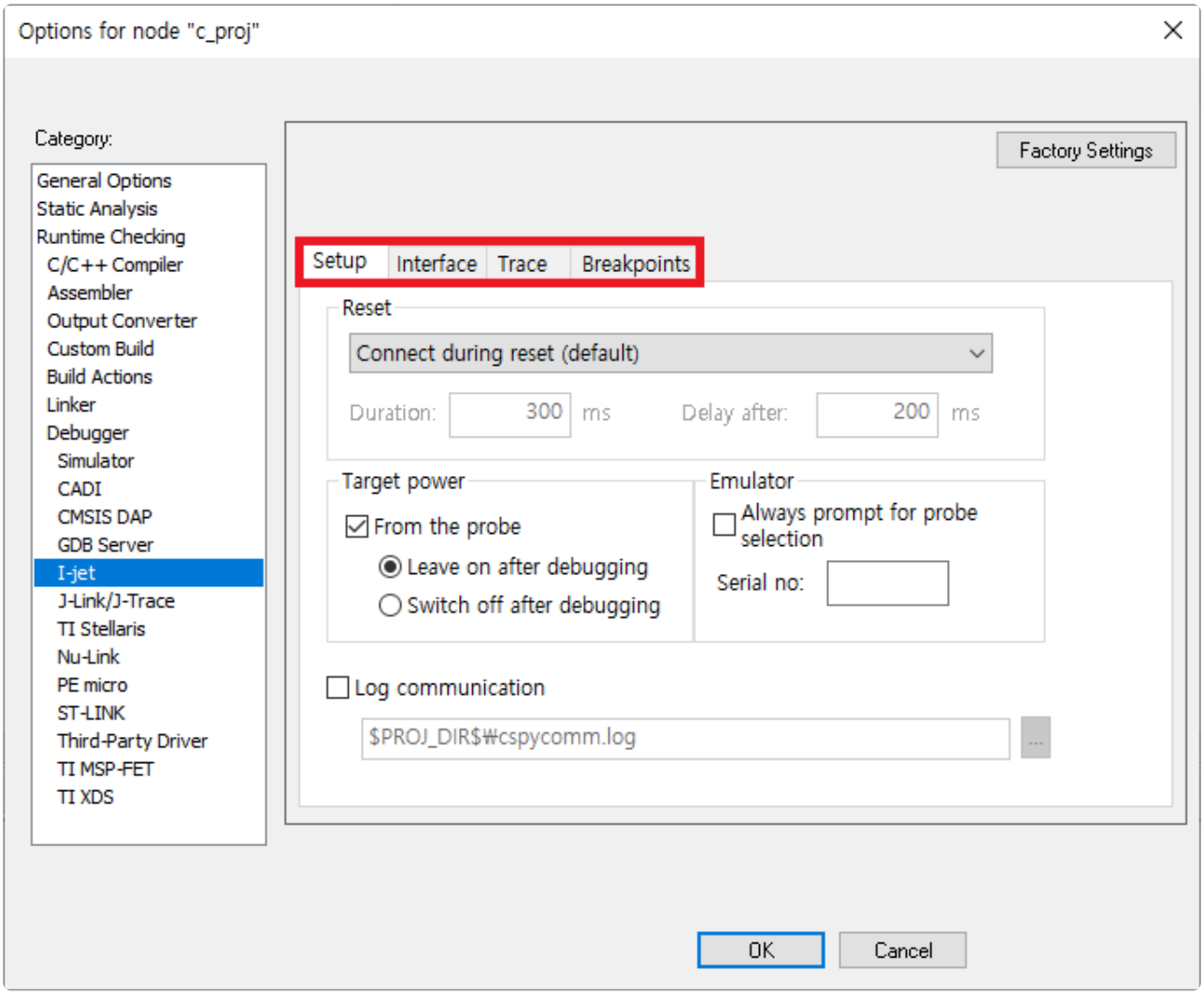
1. 먼저, [General Options]에서 [Processor variant]를 설정합니다. 예를 들어, ARM의 STM32F429IG 타깃의 경우 Device를 선택한 후 우측의 타깃 목록에서 해당 타깃에 맞는 이름을 고릅니다.



2. 두번째로 [Debugger] 카테고리로 이동하여, [Driver] 항목에서 사용하려는 디버깅 프로브를 선택합니다. 선택한 디버깅 프로브와 PC를 연결한 방식에 따라 [Debugger] 카테고리 하단의 디버깅 프로브 항목에서 세부 내용을 설정합니다.



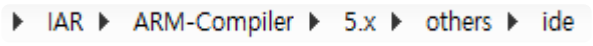
3. I-jet를 선택한 경우, [Debugger] 카테고리 하단의 [I-jet] 항목을 선택하여 세부 내용을 설정합니다. 각 설정 탭에 대한 설명은 사용하고자 하는 IAR 디버거 메뉴얼을 참조하십시오.



이제 타깃 테스트를 하기 위한 IAR 프로젝트 생성 및 설정이 끝났습니다.

6.2.4.3. Step3: CT에서 타깃 환경 설정

CT 2023.12의 타깃 테스트 프로젝트 생성 마법사 또는 프로젝트 프로퍼티의 타깃 환경 설정에서 디버거를 선택합니다. 프로젝트 생성 시 선택한 툴체인에 따라 선택 가능한 디버거 목록이 달라집니다.
IAR 툴체인을 사용하여 프로젝트 생성 시, IAR C-SPY 디버깅 기능을 사용하기 위해서는 디버거를 ide로 설정해야 합니다.



선택한 정보에 따라 설정 항목들이 표시됩니다. C-SPY를 사용하는 경우에 설정해야 하는 항목은 아래 표와 같습니다.
필수로 입력해야 하는 항목은 CT 2023.12에서 붉은 색으로 표시됩니다.

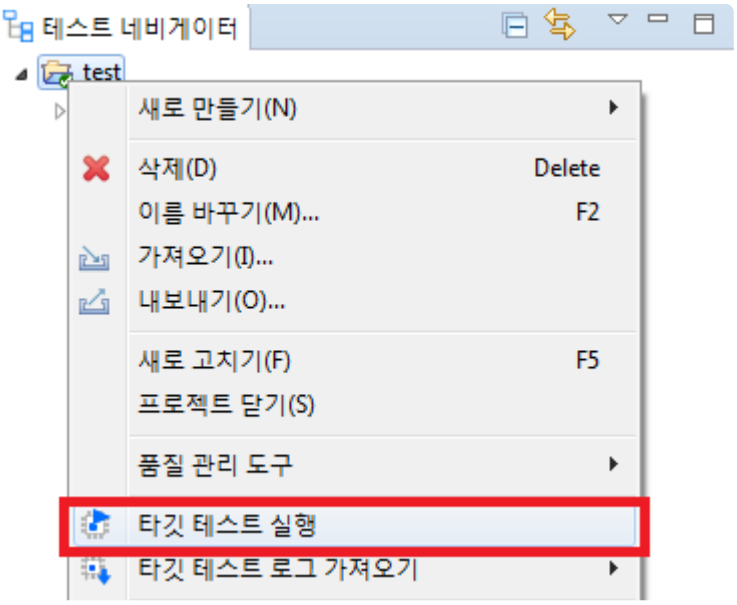
cspy_debug_general_xcl_file_path	IAR Embedded Workbench C-SPY 디버거를 사용 시 필요한 debug.general.xcl 파일 경로입니다. IAR 프로젝트 생성 시 프로젝트 파일(.ewp)이 저장된 위치의 [setting] 폴더에 자동 생성됩니다. 필수 항목입니다.
cspy_debug_driver_xcl_file_path	IAR Embedded Workbench C-SPY 디버거를 사용 시 필요한 debug.driver.xcl 파일 경로입니다. IAR 프로젝트 생성 시 프로젝트 파일(.ewp)이 저장된 위치의 [setting] 폴더에 자동 생성됩니다. 필수 항목입니다.

타깃 환경 설정이 끝나면 [OK] 또는 [Finish]버튼을 클릭합니다. 타깃 테스트를 수행할 준비가 끝났습니다.

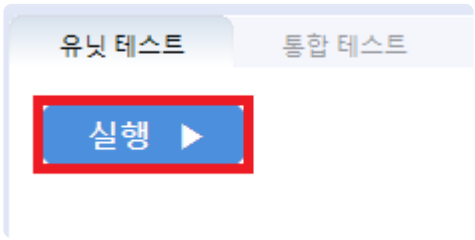
6.2.4.4. Step4: 타깃 테스트 실행

테스트 네비게이터 뷰의 프로젝트 컨텍스트 메뉴에서 [타깃 테스트 실행]을 선택하거나 테스트 뷰의 [실행] 버튼을 클릭하여, 타깃 테스트를 실행할 수 있습니다.

- [타깃 테스트 실행]



- [실행]



6.2.4.5. 디버거를 통해 타깃 테스트 디버깅

1. 타깃으로 설정 후 ‘유닛 테스트’ 뷰에서 테스트 케이스 우클릭 후 ‘디버그 정보 확인’ 클릭. Output 파일이 떨어진 것을 확인
2. 사용자 프로젝트를 직접 빌드 혹은 CT 프로젝트에서 ‘타깃 환경’ 설정에 등록한 빌드 스크립트 수행
3. 빌드 성공하는지 확인
4. IAR Workbench 실행 후 빌드한 프로젝트 포함하는 워크스페이스 선택 (.eww파일)
5. workspace 뷰에서 테스트 대상 함수가 있는 소스파일 선택 후, 라인 왼쪽을 클릭하여 디버깅 포인트 지정
6. workspace 뷰에서 프로젝트를 우클릭, Options…을 열어 Debugger 항목에서 Run to 옵션 체크 확인 및 ‘main’으로 지정되어 있는지 확인
7. 상단의 Download and Debug 버튼을 누르면 main부터 실행
8. F5 눌러서 디버깅 포인트까지 진행하여 디버깅

6.2.5. Texas Instruments Code Composer Studio (CCSv4 and greater)

CT 2023.12는 Code Composer Studio의 디버거를 사용하여 타깃 테스트를 할 수 있습니다. Code Composer Studio가 지원하는 디버깅 스크립트(4.x버전부터)를 사용하여 타깃 환경에서 테스트를 실행하고 결과를 가져옵니다. Code Composer Studio에 연결하여 사용할 수 있는 디버깅 장비 목록은 Code Composer Studio 매뉴얼을 확인하십시오.

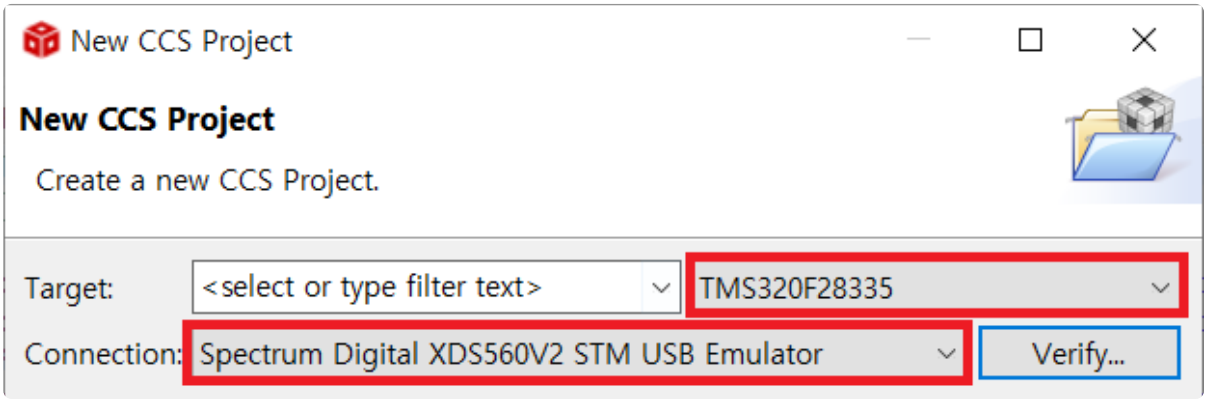
이 문서는 Code Composer Studio 디버거 사용법을 다음과 같이 세 단계로 나누어 설명합니다.

- Step1: Code Composer Studio에서 프로젝트 생성
- Step2: CT에서 타깃 환경 설정
- Step3: 타깃 테스트 실행

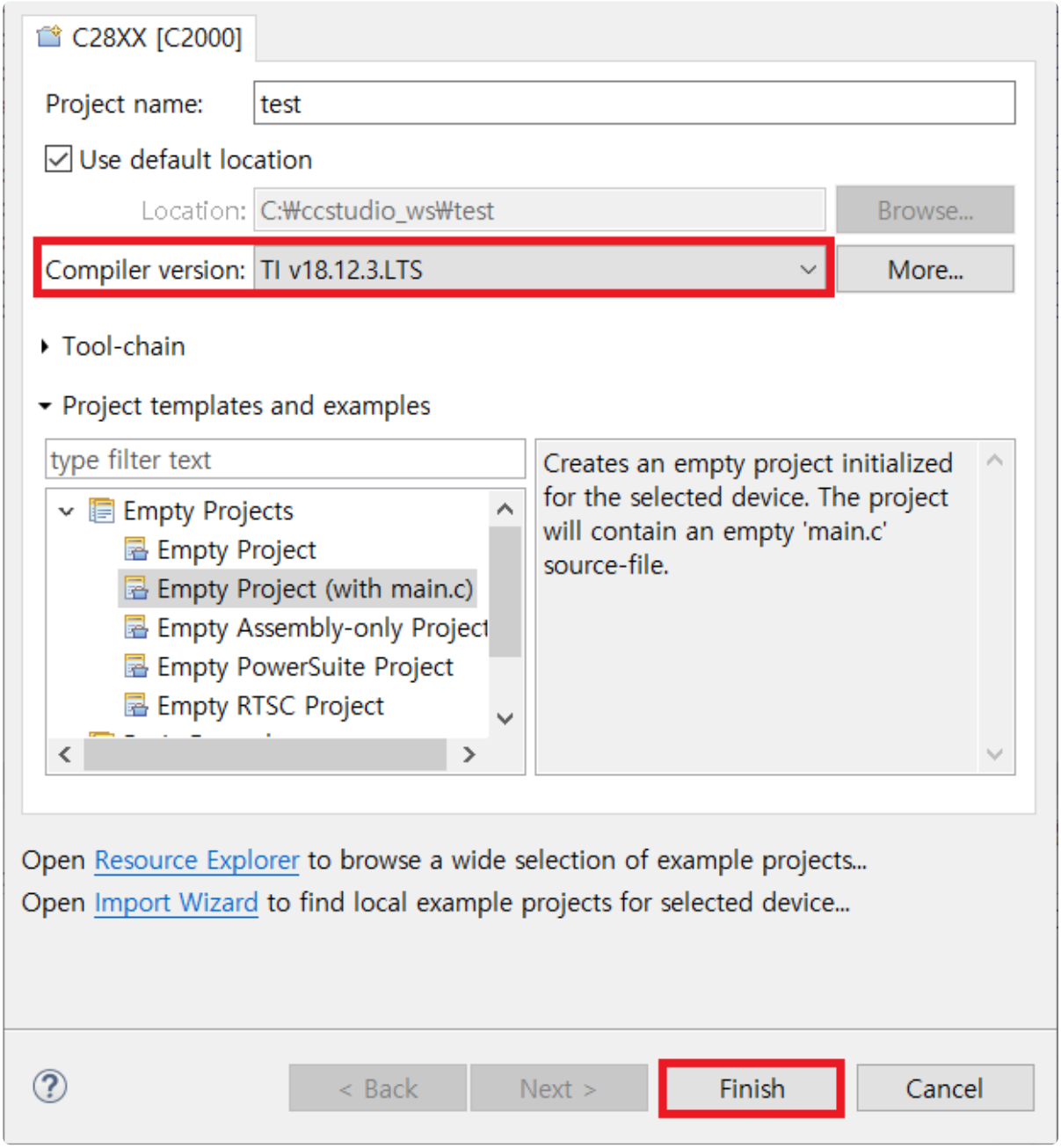
예시에서 사용한 디버거는 Spectrum Digital의 XDS560v2이고 타깃은 Texas Instruments의 TMS320입니다.

6.2.5.1. Step1: Code Composer Studio에서 프로젝트 생성

1. Code Composer Studio를 실행하고 새 프로젝트를 생성합니다. 최상위 메뉴에서 [File]-[New]를 선택한 뒤에 원하는 형태의 프로젝트를 선택합니다. 여기서는 [CCS Project]를 클릭하여 프로젝트를 생성합니다. 사용하는 타깃과 디버거 정보를 입력한 후 [Verify]를 클릭하면, 정상적으로 연결되었는지 확인할 수 있습니다.



2. 디버거와 타깃 연결을 확인한 후 나머지 설정도 입력합니다. 예시에서는 C2000 Ti 컴파일러를 사용합니다. [Finish]를 클릭하면 CCS 프로젝트가 워크스페이스에 생성됩니다.



Code Composer Studio는 Texas Instruments에서 기본으로 제공하는 디버거 외에 몇가지 디버거를 더 지원합니다.

- TI XDS USB (CCStudio default)
- BlackHawk JTAG emulator
- Spectrum digital
- MSP430 USB
- MSP432 USB
- Tiva/Stellaris ICDI

CT 2023.12는 Code Composer Studio에서 지원하는 디버거를 javascript로 제어합니다. Code Composer Studio의 프로젝트 설정 화면에서 타깃과 디버거 관련 상세 정보를 확인할 수 있습니다.

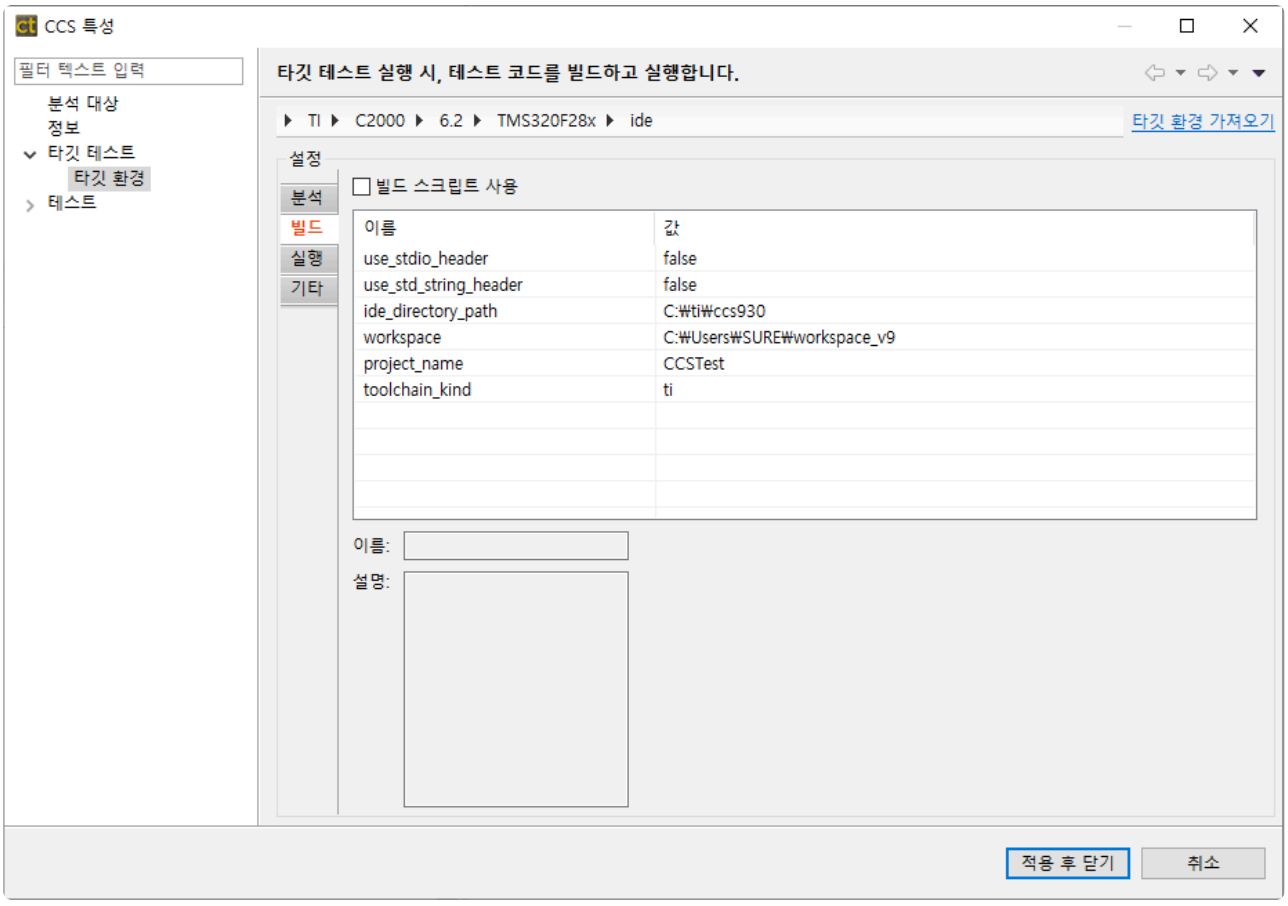
6.2.5.2. Step2: CT에서 타깃 환경 설정

1. CT 2023.12 프로젝트를 생성합니다. 프로젝트 생성에 대한 자세한 내용은 본 문서의 Texas Instruments Code Composer Studio 를 참고하시기 바랍니다.

2. 테스트 네비게이터의 프로젝트 우클릭 후 [특성] > [타깃 테스트] > [타깃 환경]을 선택합니다. [타깃 환경]에서 타깃 환경 설정을 할 수 있습니다. 프로젝트를 생성할 때 선택한 툴체인에 따라 선택 가능한 디버거와 설정 항목이 달라집니다.

3. CT 2023.12의 타깃 환경 설정 페이지에서 디버거를 선택합니다. 예시에서는 Code Composer Studio 디버거를 사용하기 때문에 IDE 디버거를 선택합니다.
   ▶ TI ▶ C2000 ▶ 6.2 ▶ TMS320F28x ▶ ide

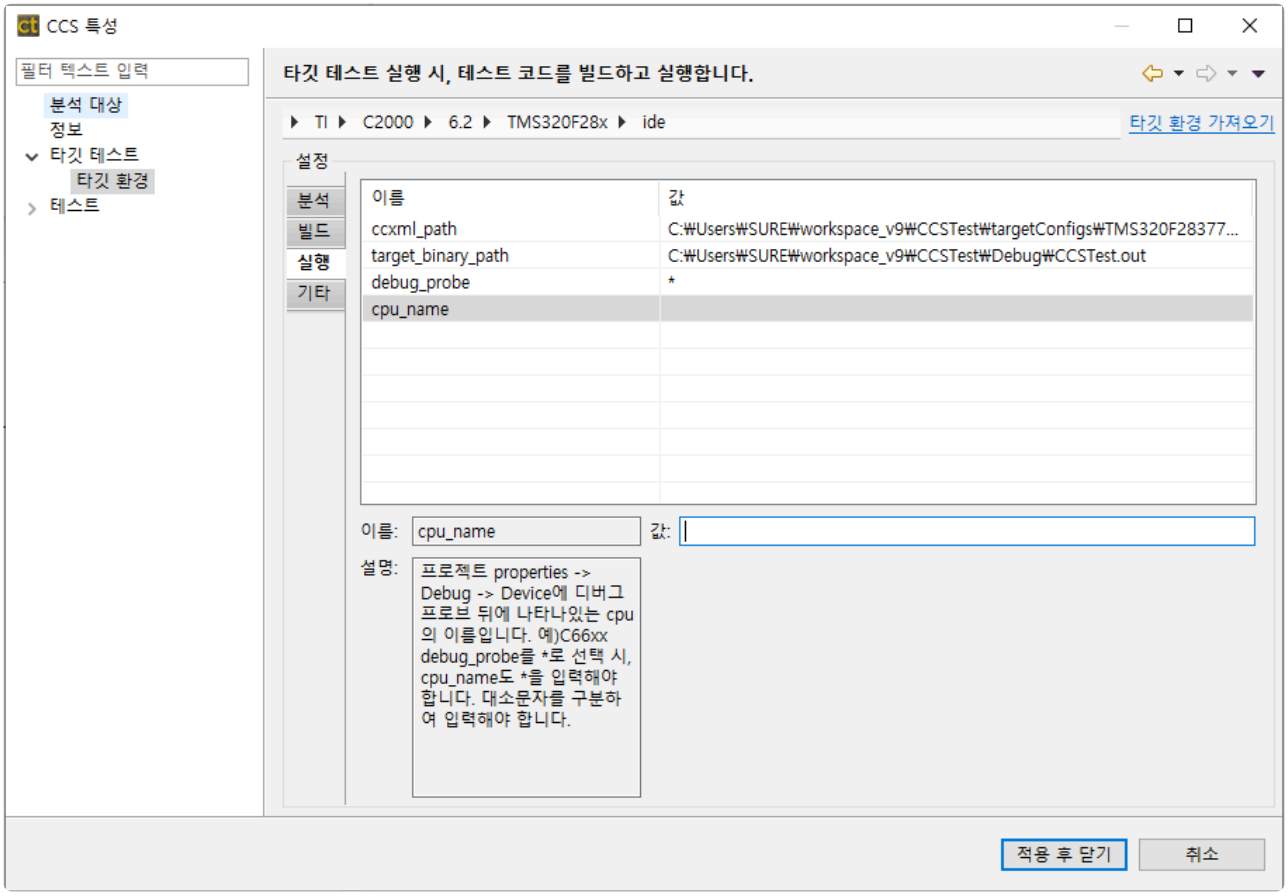
4. Code Composer Studio 빌드를 하기 위해서 타깃 환경 설정의 빌드 탭에 필요한 정보들을 입력합니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.



- 빌드 탭 항목

ide_directory_path	Code Composer Studio의 디렉토리 경로 *ex.C:\ti\ccs930*
workspace	Code Composer Studio의 워크스페이스 디렉토리 경로
project_name	CT 2023.12에서 분석할 Code Composer Studio 프로젝트 이름

5. 실행 탭



- 실행 탭 항목

ccxml_path	Code Composer Studio 타깃 구성 파일의 경로입니다. 프로젝트 경로와 타깃의 이름이 맞는지 확인합니다. 파일의 이름은 Code Composer Studio에서 설정한 타깃의 이름입니다. 예: {project_path}\targetConfig\{target_name}.ccxml
target_binary_path	Code Composer Studio 빌드 시 생성되는 바이너리 파일의 경로를 입력합니다. 예: {project_path}\Debug\{project_name}.out
debug_probe	Code Composer Studio properties에서 Device의 앞 부분을 참고하여 해당 타겟 디버거의 이름을 입력합니다. (예시의 사진에서는 Spectrum Digital XDS560V2 STM USB Emulator)
cpu_name	Code Composer Studio properties에서 Device의 뒷 부분을 참고하여 해당 타겟 CPU의 이름을 입력합니다. (예시의 사진에서는 C28xx)

- Code Composer Studio properties (Code Composer Studio 프로젝트 우클릭 후 [Properties] > [Debug] > [Device])

타깃에 디버거가 하나만 연결되어있을 시, debug_probe는 기본값(*)으로 두어도 됩니다.

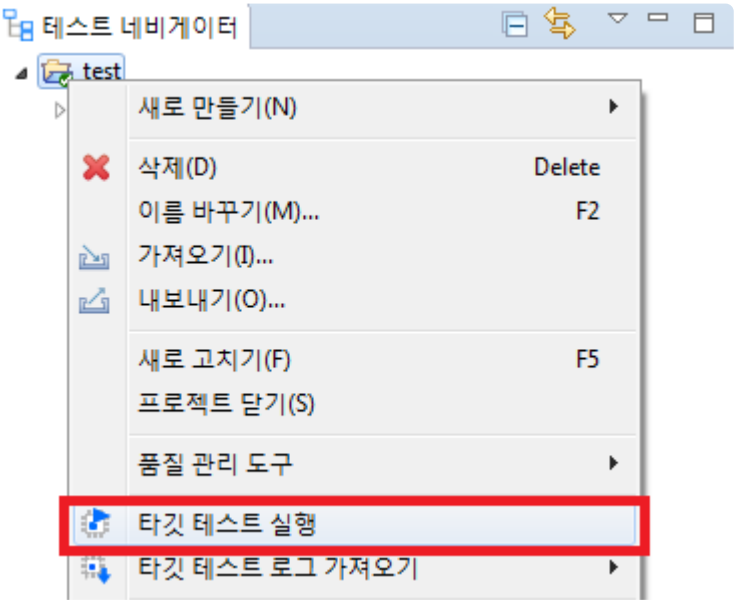
> single core cpu일 경우에는 cpu_name을 설정하지 않아도 됩니다.

6. 타깃 환경 설정이 끝나면 [OK] 또는 [Finish]버튼을 클릭합니다. 타깃 테스트를 수행할 준비가 끝났습니다.

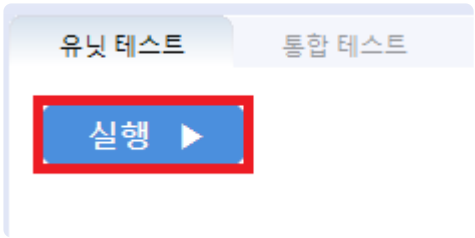
6.2.5.3. Step3: 타깃 테스트 실행

타깃 테스트를 실행하기에 앞서 빌드하고자 하는 프로젝트가 위치한 워크스페이스의 사용을 종료해야 합니다. IDE에서 워크스페이스를 사용 중이라면 타깃 테스트가 정상적으로 수행되지 않습니다.
테스트 네비게이터 뷰의 프로젝트 컨텍스트 메뉴에서 [타깃 테스트 실행]을 선택하거나 테스트 뷰의 [실행] 버튼을 클릭하여, 타깃 테스트를 실행할 수 있습니다.

- [타깃 테스트 실행]



- [실행]



실행 과정에서 Code Composer Studio가 실행되어 있으면 컴파일 에러가 발생합니다.

CCS의 디버그 스크립팅에 대한 자세한 정보는 Texas Instruments 홈페이지 를 참고하십시오.

6.2.5.4. 디버거를 통해 타깃 테스트 디버깅

1. ‘타깃 테스트 코드 내보내기’ 수행
2. CT_워크스페이스\.metadata\.plugins\com.codescroll.ut.embedded\프로젝트명\TestFixture\cs 경로로 이동
3. notepad로 해당 경로 아래에 있는 모든 파일들의 내용에서 #line 문자를 //#line으로 변경(#line -> //#line)
4. Code Composer Studio에서 내보내기 된 코드 빌드 수행
5. 디버깅 모드로 실행하면 cs_tfx.c 파일의 main 함수에 break point가 걸리는 것을 확인
6. Code Composer Studio 에서 좌측 상단에 ‘File’ > ‘Open File’ 클릭
7. 2번의 경로에서 테스트함수_test숫자.c 파일 열기(ex. zlibVersion_test0.c 파일)
8. ‘Open File’로 테스트 대상 함수의 정의가 있는 소스파일_숫자.c 파일도 열기(ex. inffast_1.c 파일)
9. 7번에 있는 함수에 break point를 지정, 8번에 있는 테스트 대상 함수의 시작부에 break point 지정
10. 7번과 8번에서 열었는 파일에서 break point가 걸리는 것을 확인할 수 있음(8번의 경우 함수 정의 시작부분에 break point를 지정)
11. 디버깅 수행

> ! Code Composer Studio에서는 탐침이 들어간 상태의 코드로만 디버깅을 할 수 있습니다. 또한 Code Composer Studio에서 디버깅이 가능한 환경이 구성되어 있지 않으면 디버깅을 할 수 없습니다.

6.2.6. Microchip MPLAB IDE

이 문서는 Microchip MPLAB IDE를 이용한 타깃 테스트 방법을 다음과 같이 세 단계로 나누어 설명합니다.

6.2.6.1. Step1: 디버거 스크립트 설정

CT 2023.12에서 타깃 테스트를 수행하기 위해 디버거에서 출력하는 로그를 파일 형태로 저장할 수 있도록 MPLAB에 포함되어 있는 mdb.bat 파일을 수정해주어야 합니다.
mdb.bat 파일 경로는 아래와 같습니다.

windows 32 bit인 경우

- `C:\Program Files\Microchip\MPLABX\vn.nn\mplab_ide\bin\mdb.bat`

Windows 64 bit인 경우

- `C:\Program Files (x86)\Microchip\MPLABX\vn.nn\mplab_ide\bin\mdb.bat`

mdb.bat 파일의 마지막 라인에 있는 코드를 다음과 같이 수정해주시면 됩니다.

수정 전

```
"%jdkhome:exe =exe%" -Dfile.encoding=UTF-8 -jar "%mdb_jar%" %1
```

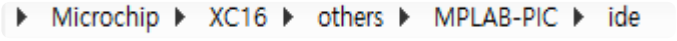
수정 후

```
call "%jdkhome:exe =exe%" -Dfile.encoding=UTF-8 -jar "%mdb_jar%" %1 >> %CT_TAR
GET_PATH%\mdb_log.txt
```

> ! Microchip MPLAB은 한글 인코딩 이슈가 있어 CT 2023.12 워크스페이스 또는 프로젝트명에 한글을 포함시키면 안됩니다.

6.2.6.2. Step2: CT에서 타깃 환경 설정

CT 2023.12의 타깃 테스트 프로젝트 생성 마법사 또는 프로젝트 특성의 타깃 환경 설정에서 디버거를 선택합니다. 프로젝트 생성 시 선택한 툴체인에 따라 선택 가능한 디버거 목록이 달라집니다.
디버거를 ide로 설정합니다.



선택한 정보에 따라 설정 항목들이 표시됩니다.
필수로 입력해야 하는 항목은 CT 2023.12에서 붉은 색으로 표시됩니다.

ide_directory_path	Mplab ide가 설치된 디렉터리의 경로입니다. 필수 항목입니다.
project_directory_path	프로젝트의 디렉터리 경로입니다. 필수 항목입니다.
make_path	make.exe 파일 경로입니다. mplab 프로젝트에서 빌드시 사용하는 make.exe 경로를 입력해주시면 됩니다. 필수 항목입니다.

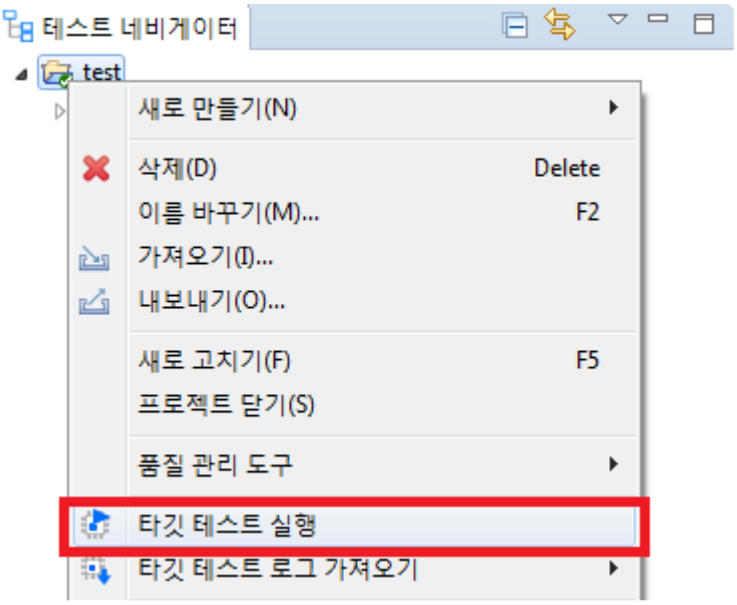
target_binary_path	타깃에 업로드할 바이너리 파일입니다.(빌드 시 생성되는 바이너리 위치) 필수 항목입니다.
debugger_tool	디버거 툴 정보를 선택하시면 됩니다.(ICD3, RealICE, PICkit3, SIM, PM3, LicensedDebugger, LicensedProgrammer, SK 중 선택) 필수 항목입니다.
chip	테스트 대상 칩의 제품 이름입니다(ex..dsPIC33EP512MU814). 필수 항목입니다.

CT 2023.12에서 타깃 테스트를 수행하기 위해서는 Step1 의 내용처럼 mdb.bat 파일이 수정이 되어있어야 합니다.
타깃 환경 설정이 끝나면 [OK] 또는 [Finish]버튼을 클릭합니다. 타깃 테스트를 수행할 준비가 끝났습니다.

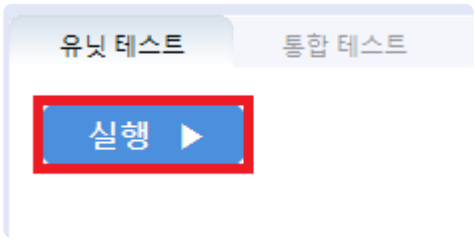
6.2.6.3. Step3: 타깃 테스트 실행

테스트 네비게이터 뷰의 프로젝트 컨텍스트 메뉴에서 [타깃 테스트 실행]을 선택하거나 테스트 뷰의 [실행] 버튼을 클릭하여, 타깃 테스트를 실행할 수 있습니다.

- [타깃 테스트 실행]



- [실행]



6.3. 타깃 빌드 가이드

CT 2023.12 타깃 프로젝트 정보를 사용하여 타깃 테스트 코드를 빌드하는 방법을 안내합니다.

- IAR Embedded Workbench IDE
- Texas Instruments Code Composer Studio
- CodeWarrior IDE
- Hightec Development Platform IDE
- Tasking VX IDE
- Renesas CS+ IDE
- MPLAB X IDE
- Microsoft Visual Studio
- GNU Compiler

6.3.1. IAR Embedded Workbench IDE

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

IAR Embedded Workbench 빌드를 하기 위해서는 타깃 환경 설정의 분석 및 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

- 분석 탭

cpu	Processor variant의 Core에서 선택할 수 있는 타겟의 cpu

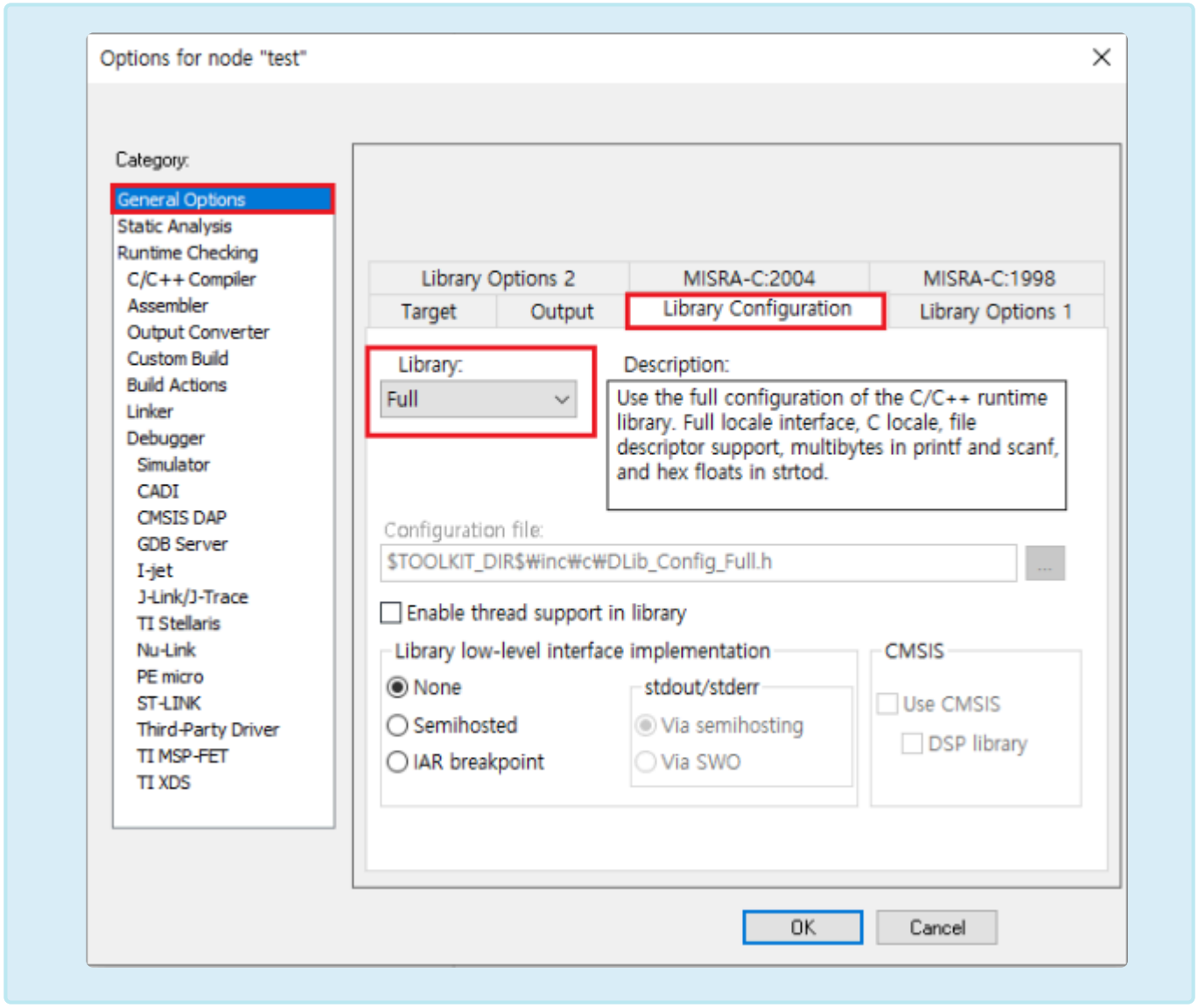
- 빌드 탭

ide_directory_path	IAR Embedded Workbench IDE 설치 경로 *ex. C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.4*
project_file_path	IAR Embedded Workbench 프로젝트 파일(.ewp) 경로
build_configuration	IAR Embedded Workbench 프로젝트 build configuration (Project -> Edit Configurations…)

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

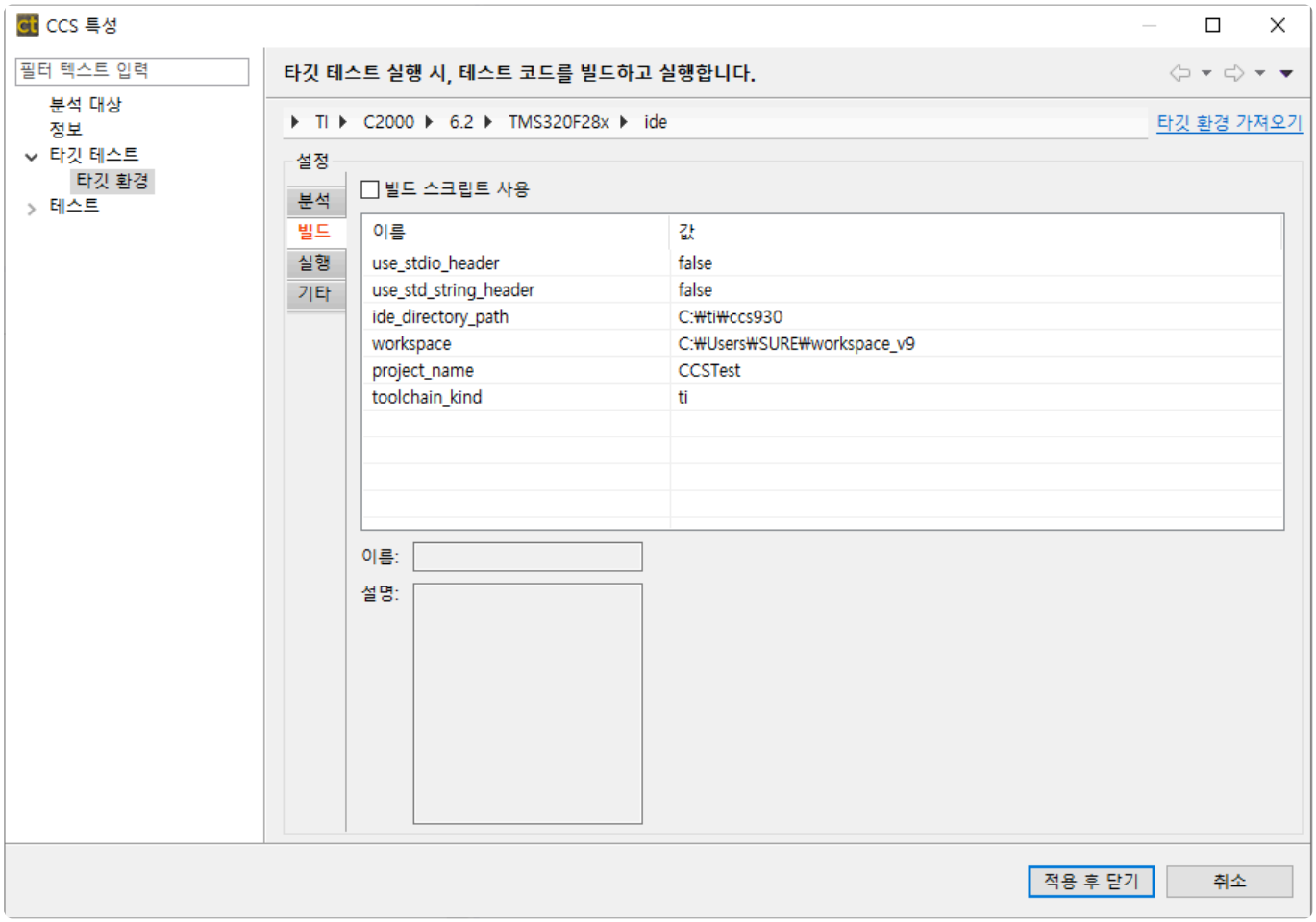
stdio.h 의 IO 함수 사용 시 라이브러리 설정 변경이 필요합니다.
워크스페이스의 프로젝트 우클릭 -> Options -> General Options -> Library Configuration -> Library tab 을 Full로 변경합니다.


Options for node "test"
Category:
General Options
Static Analysis
Runtime Checking
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
CADI
CMSIS DAP
GDB Server
I-jet
J-Link/J-Trace
TI Stellaris
Nu-Link
PE micro
ST-LINK
Third-Party Driver
TI MSP-FET
TI XDS
Library Options 2
MISRA-C:2004
MISRA-C:1998
Target
Output
Library Configuration
Library Options 1
Library:
Full
Description:
Use the full configuration of the C/C++ runtime library. Full locale interface, C locale, file descriptor support, multibytes in printf and scanf, and hex floats in strtod.
Configuration file:
$TOOLKIT_DIR$¥inc¥c¥DLib_Config_Full.h
Enable thread support in library
Library low-level interface implementation
None
Semihosted
IAR breakpoint
stdout/stderr
Via semihosting
Via SWO
CMSIS
Use CMSIS
DSP library
OK
Cancel

6.3.2. Texas Instruments Code Composer Studio

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

Code Composer Studio 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.



- 빌드 탭

ide_directory_path	Code Composer Studio의 디렉토리 경로 *ex.C:\ti\ccs930*
workspace	Code Composer Studio의 워크스페이스 디렉토리 경로
project_name	Controller Tester에서 분석할 Code Composer Studio 프로젝트 이름

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

! 실행 과정에서 Code Composer Studio가 실행되어 있으면 컴파일 에러가 발생합니다.

6.3.3. CodeWarrior IDE

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

CodeWarrior 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

- 빌드 탭

ide_directory_path	CodeWarrior IDE 설치 경로 *ex. C:\Program Files (x86)\Freescale\CW for MPC55xx and MPC56xx 2.10, C:\Freescale\CW MCU*
ide_version	IDE 버전, Classic 또는 Eclipse
project_file_path	Classic의 경우 프로젝트 생성 시 명명한 .mcp 파일, Eclipse의 경우 프로젝트 생성 시 만들어진 .project 파일

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

6.3.4. Hightec Development Platform IDE

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

Hightec IDE 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

- 빌드 탭

ide_directory_path	Hightec IDE 설치 경로 *ex. C:\HIGHTEC\toolchains\arm\v4.6.5.0*
project_directory_path	HighTec IDE에서 생성한 프로젝트 디렉터리의 경로

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

6.3.5. Tasking VX IDE

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

Tasking VX IDE 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

- 빌드 탭

ide_version	설치된 Tasking VX IDE의 버전
makefile_path	Tasking 프로젝트에 생성된 makefile의 경로
ide_directory_path	Tasking VX IDE가 설치된 디렉토리 경로 *ex. C:\Program Files (x86)\TASKING\C166-VX v3.1r2*

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

6.3.6. Renesas CS+ IDE

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

Renesas CS+ IDE 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

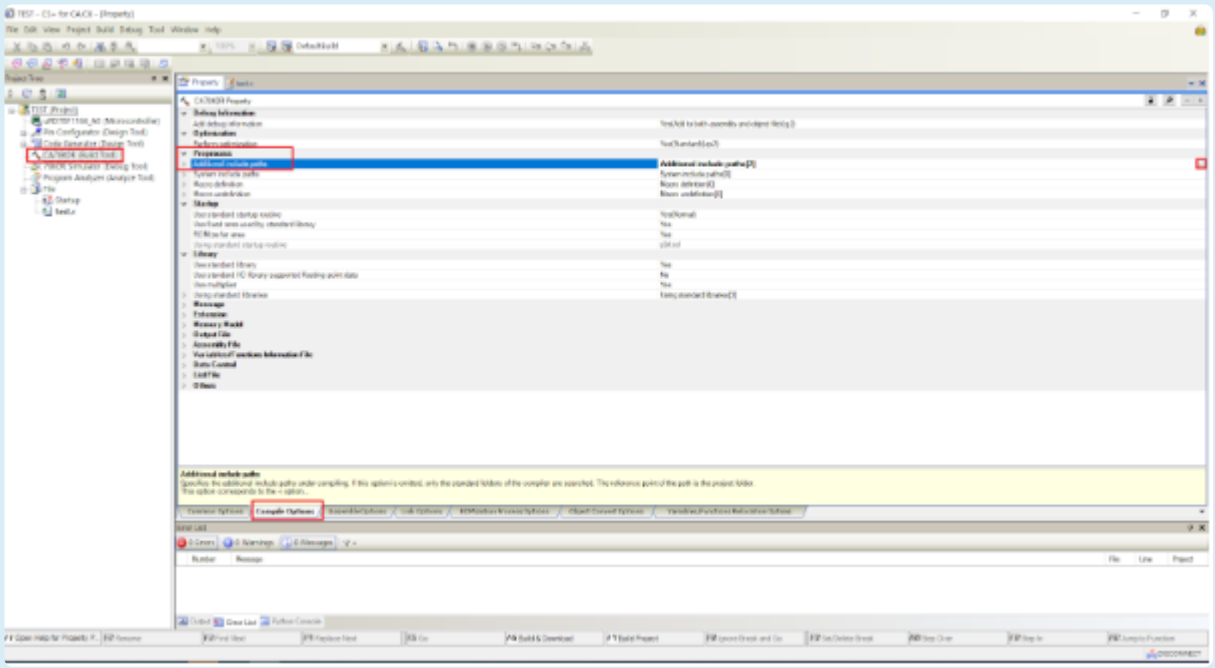
- 빌드 탭

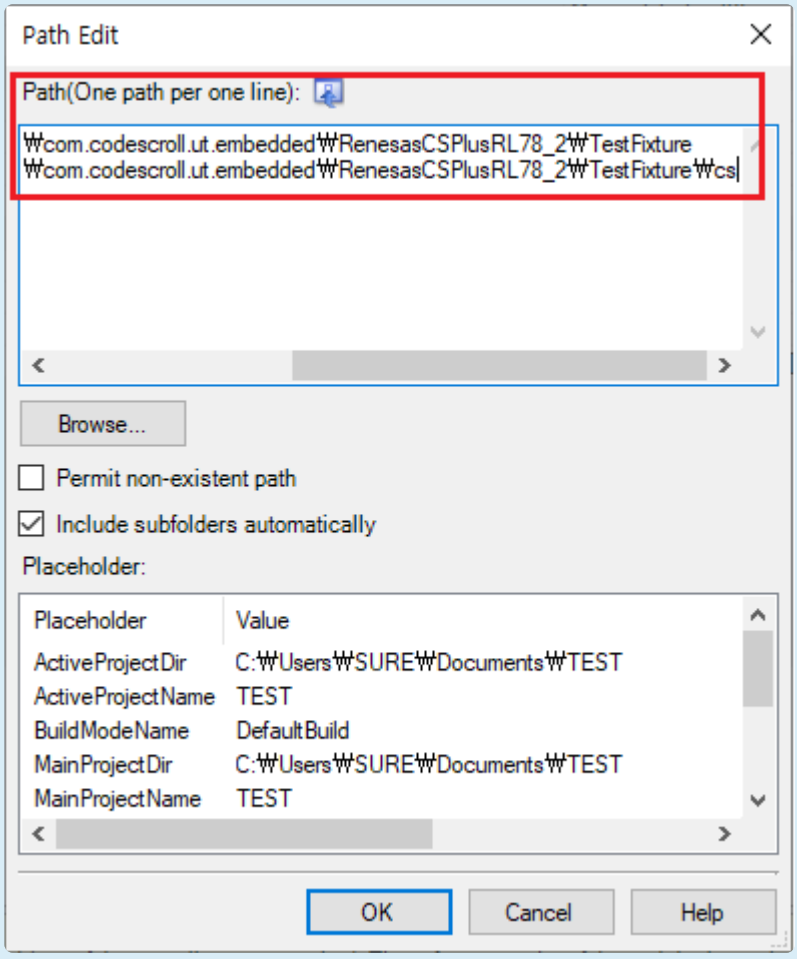
ide_directory_path	Renesas CS+ 설치 디렉토리 경로 *ex. C:\Program Files (x86)\Renesas Electronics*
ide_kind	IDE 종류(CS+)
workspace_path	Renesas HEW IDE의 경우만 필요하므로,CS+에서는 임의의 경로 지정
project_file_path	Renesas CS+에서 생성한 프로젝트 파일 경로(.mtpj)

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

> Controller Tester에서 테스트 코드들을 내보낼 때 일부 코드는 상대 경로를 참조합니다.
> 타깃 테스트 코드를 빌드하기 위해서는 Renesas CS+ 프로젝트에 해당 경로의 절대 경로를 참조하도록 설정해야 합니다.
> - Build Tool의 Property -> Compile Options -> Preprocess -> Additional include paths 에서 아래의 경로 추가
> *(CTWORKSPACE)* \.metadata\.plugins\com.codescroll.ut.embedded\ *CT project name* \TestFixture
> *(CTWORKSPACE)* \.metadata\.plugins\com.codescroll.ut.embedded\ *CT project name* \TestFixture\cs




Path Edit
Path(One path per one line):
₩com.codescroll.ut.embedded₩RenesasCSPlusRL78_2₩TestFixture
₩com.codescroll.ut.embedded₩RenesasCSPlusRL78_2₩TestFixture₩cs
Browse...
Permit non-existent path
Include subfolders automatically
Placeholder:
Placeholder
Value
ActiveProjectDir
C:₩Users₩SURE₩Documents₩TEST
ActiveProjectName
TEST
BuildModeName
DefaultBuild
MainProjectDir
C:₩Users₩SURE₩Documents₩TEST
MainProjectName
TEST
OK
Cancel
Help

6.3.7. MPLAB X IDE

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

MPLAB X IDE 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

- 빌드 탭

ide_directory_path	MPLAB X IDE 설치 경로 *ex. C:\Program Files (x86)\Microchip\MPLABX\v5.35*
project_directory_path	MPLAB X IDE에서 생성한 프로젝트 디렉터리 경로

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

6.3.8. Microsoft Visual Studio

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

Microsoft Visual Studio 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

- 빌드 탭

ide_directory_path	Microsoft Visual Studio 설치 경로 *ex. C:\Program Files (x86)\Microsoft Visual Studio 10.0*
build_configuration	대상 솔루션의 테스트할 구성 및 플랫폼
sin_path	대상 솔루션의 파일 경로 (.sin 파일)

위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

6.3.9. GNU Compiler

타깃 환경 설정 페이지는 사용자가 선택한 툴체인에 따라 자동으로 정보가 채워집니다. 선택 가능한 디버거의 종류는 툴체인 분석 설정에 따라 달라집니다.

GNU Compiler 빌드를 하기 위해서는 타깃 환경 설정의 빌드 탭에서 필요한 정보들을 입력하고 완료를 누릅니다. 작성해야 하는 항목은 아래 표와 같으며, 이는 필수 항목입니다.

- 빌드 탭

makefile_path	사용자가 작성한 makefile의 경로

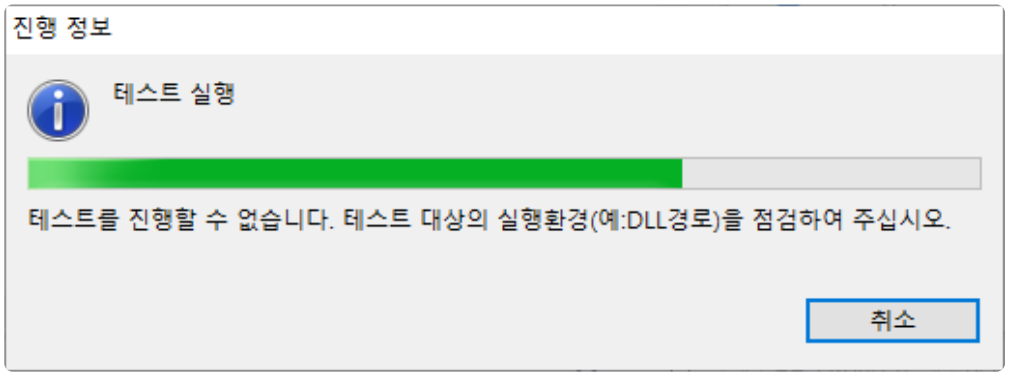
위 항목을 작성하지 못하고 타깃 환경 설정 페이지의 완료 버튼을 누른 경우나 경로가 변경된 경우에는 테스트 네비게이터의 프로젝트 마우스 우클릭 -> 특성 -> 타깃 테스트 -> 타깃 환경에서 다시 설정할 수 있습니다.

타겟 환경 설정을 마치고 유닛 테스트 뷰의 실행 버튼을 누르면 Controller Tester에서 타겟 테스트 코드를 빌드합니다.

7. 테스트 오류 발생 시 원인 파악하기

CodeScroll Controller Tester에서 테스트를 수행할 때 오류가 발생하는 경우가 있습니다. 이 때 사용자는 Controller Tester의 디버그 정보 확인 기능으로 테스트 오류 발생 원인을 찾아볼 수 있습니다.

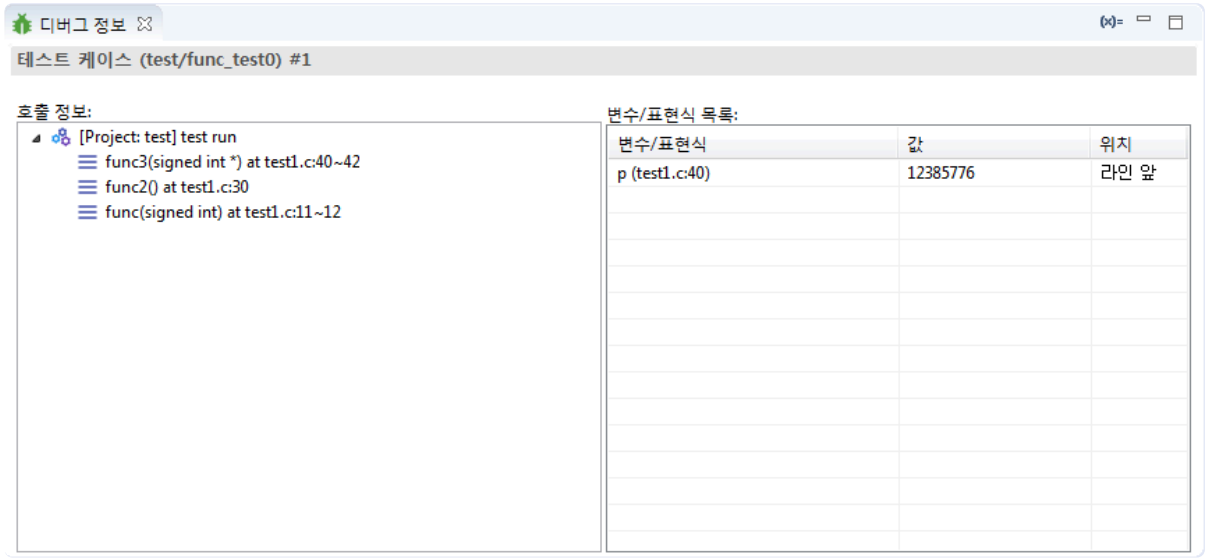
테스트 실행이 실패하는 경우



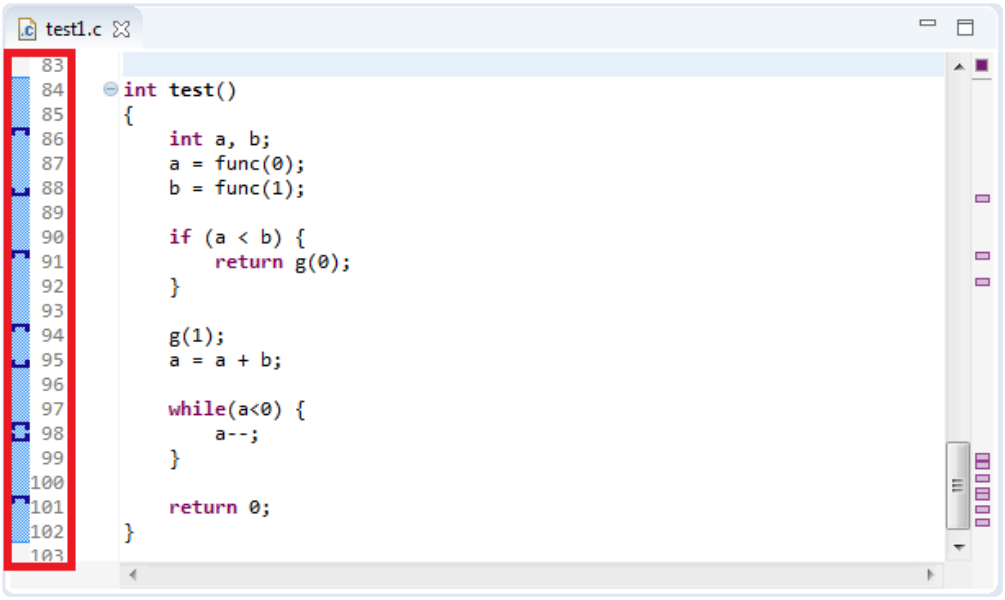
테스트가 실패하는 경우에도 디버그 정보 확인을 수행할 수 있습니다. 생성된 테스트 케이스의 [디버그 정보 확인]을 수행하면 [디버그 정보 뷰]에 호출 정보가 표시되며, 테스트가 실패한 위치를 알 수 있습니다.

테스트 실행 후 결과가 오류를 포함하는 경우

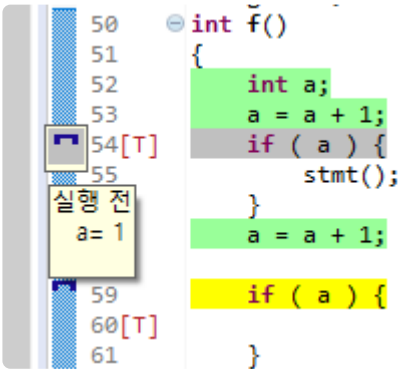
Controller Tester에서 테스트를 수행한 후 Signaled, Abnormal Exit과 같은 오류 결과를 표시하는 경우가 있습니다. 오류가 발생한 테스트 케이스의 [디버그 정보 확인]을 수행하면 [디버그 정보 뷰]에 함수 호출 정보가 표시됩니다. 만일 디버그할 변수/표현식을 추가했다면, 실행된 변수/표현식 목록이 함께 표시됩니다.



호출 정보는 함수 호출 순서를 나타냅니다. 함수가 호출된 위치가 기록되며, 호출 정보 최상단에 마지막 실행 위치가 기록됩니다.
변수/표현식 목록은 테스트 케이스와 함께 실행된 변수/표현식 값을 나타냅니다. 소스 코드 전체에 추가된 변수/표현식 목록은 [디버그 정보 뷰]의 툴바 메뉴의 [변수/표현식 목록]에서 확인할 수 있습니다.



소스 코드 편집기의 마커에서도 디버그할 변수/표현식 정보를 확인할 수 있습니다. 디버그할 변수/표현식을 추가하게 되면 소스 코드 편집기에 추가 위치가 마커로 표현되며, 각 마커 위로 마우스를 올리면 해당 위치에 추가된 변수/표현식 목록을 확인할 수 있습니다.
디버그 정보를 포함하는 테스트 케이스를 선택한 경우, 각 마커에 테스트 케이스가 실행한 변수/표현식 결과가 함께 표시됩니다.



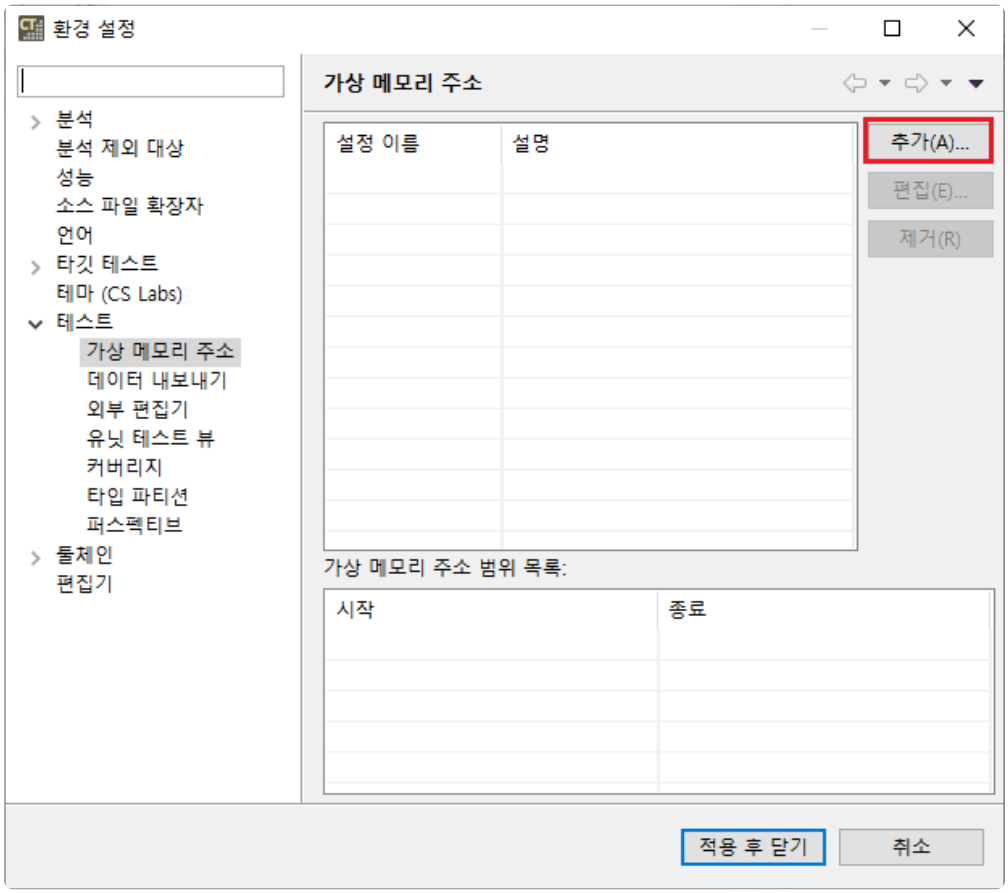
함수 호출 정보와 실행된 변수/표현식 값을 통해 [디버그 정보 확인]을 수행한 테스트 케이스의 오류 발생 원인을 찾아볼 수 있습니다.

> ✻ 디버그할 변수/표현식을 추가하는 방법은 User Manual 문서의 [디버그 정보 확인] 항목을 참조하십시오.

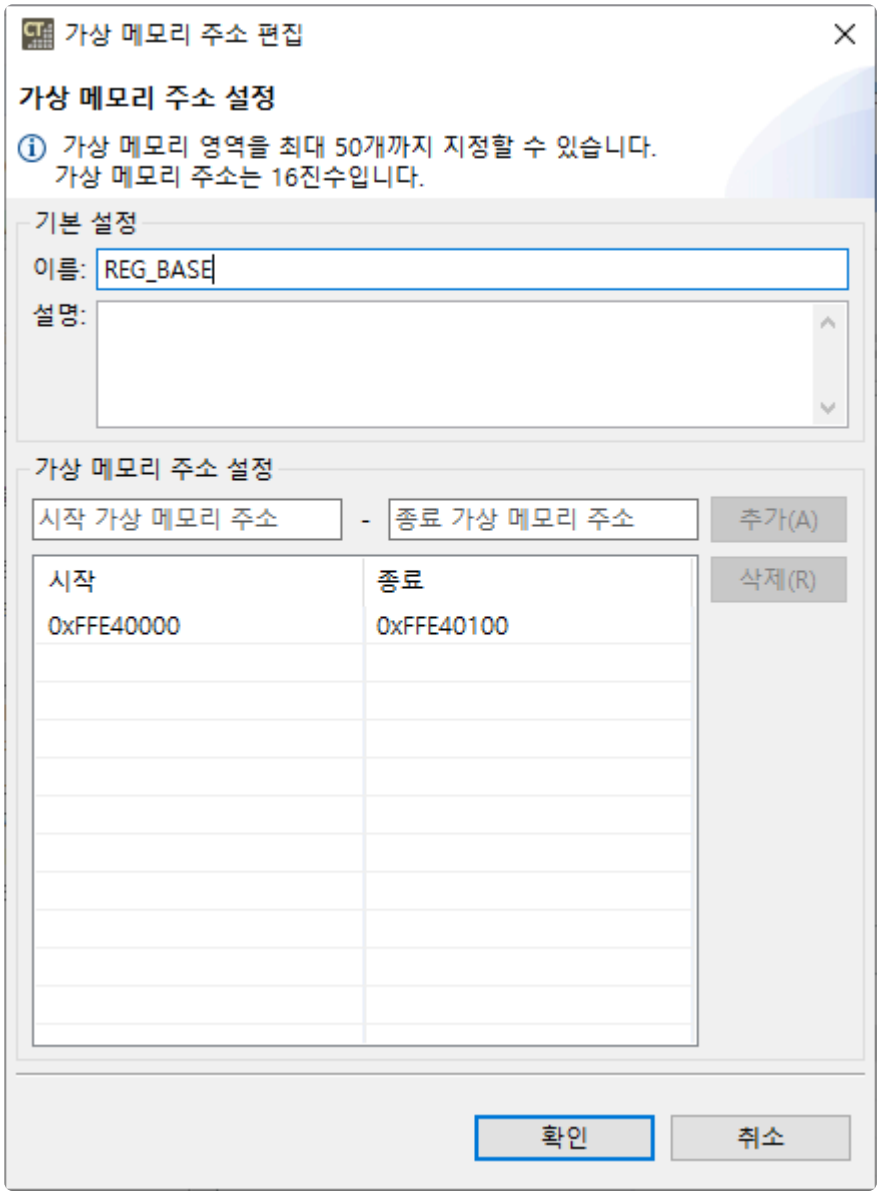
8. 가상 주소 사용 가이드

가상 메모리 주소를 설정하여 임베디드 환경 테스트를 위한 메모리를 설정할 수 있습니다.

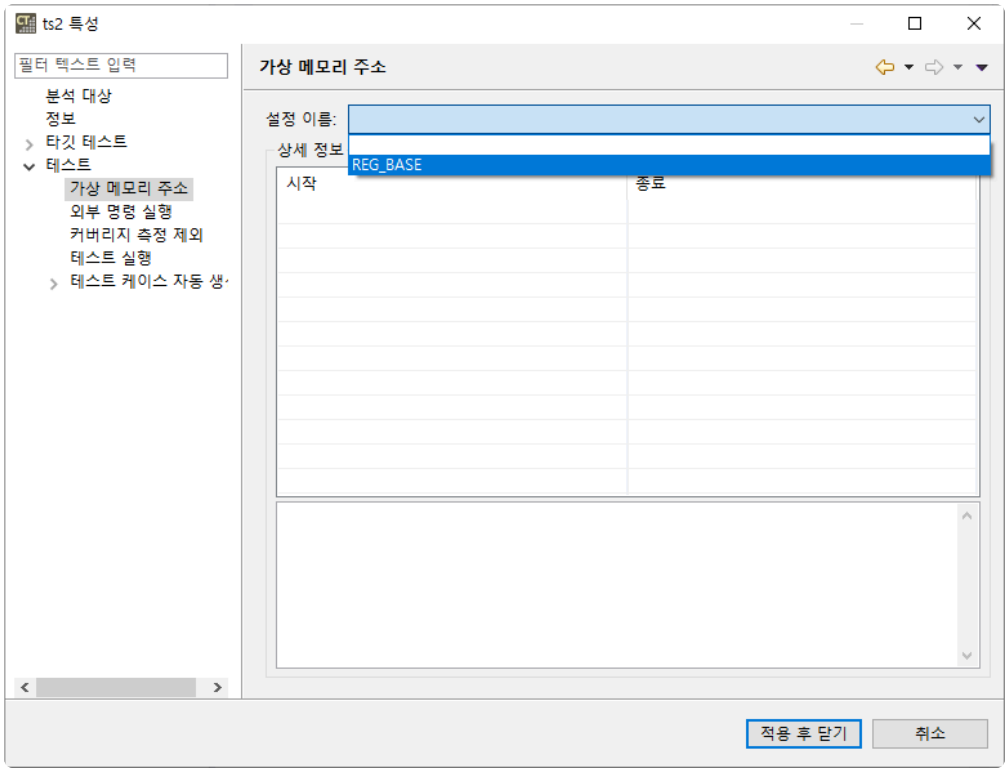
1. 상단 메뉴 [창] – [환경 설정] – [테스트] – [가상 메모리 주소] – [추가(A)...] 선택



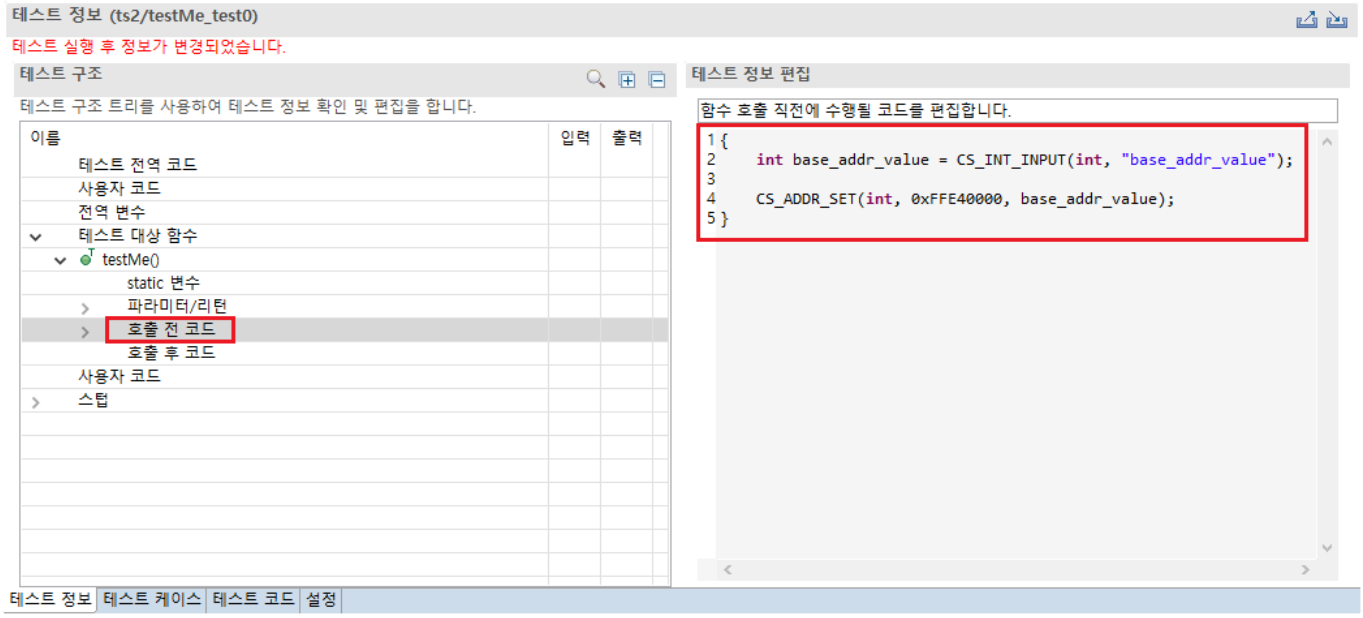
2. 가상 주소의 이름과 범위 입력 후 [추가(A)] 버튼 클릭



3. 프로젝트 우클릭 – [특성] – [테스트] -[가상 메모리 주소] 선택 후 [설정 이름] 콤보 박스에서 등록했던 가상 주소 범위 선택

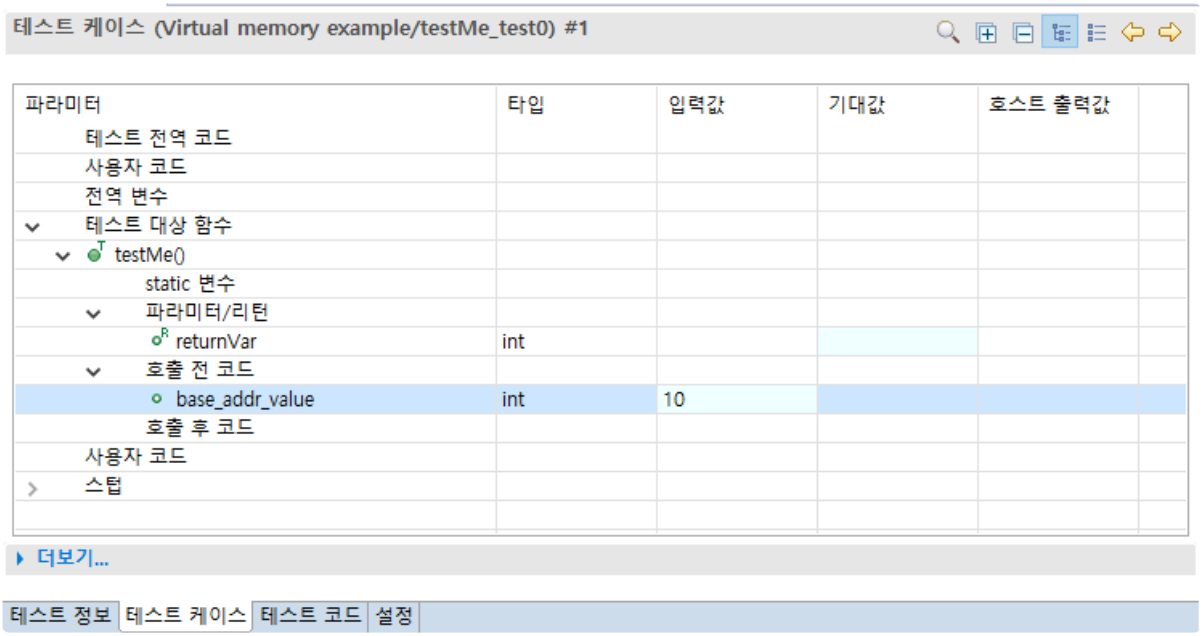


4. 테스트 구조 편집기의 [호출 전 코드]에 가상 주소에 값을 세팅하는 매크로 사용



매크로에 대한 자세한 설명은 매뉴얼의 테스트 매크로 페이지를 확인하시기 바랍니다.

5. 테스트 케이스 값 편집


테스트 케이스 (Virtual memory example/testMe_test0) #1
파라미터
타입
입력값
기대값
호스트 출력값
테스트 전역 코드
사용자 코드
전역 변수
테스트 대상 함수
testMe()
static 변수
파라미터/리턴
returnVar
int
호출 전 코드
base_addr_value
int
10
호출 후 코드
사용자 코드
스텁
더보기...
테스트 정보
테스트 케이스
테스트 코드
설정

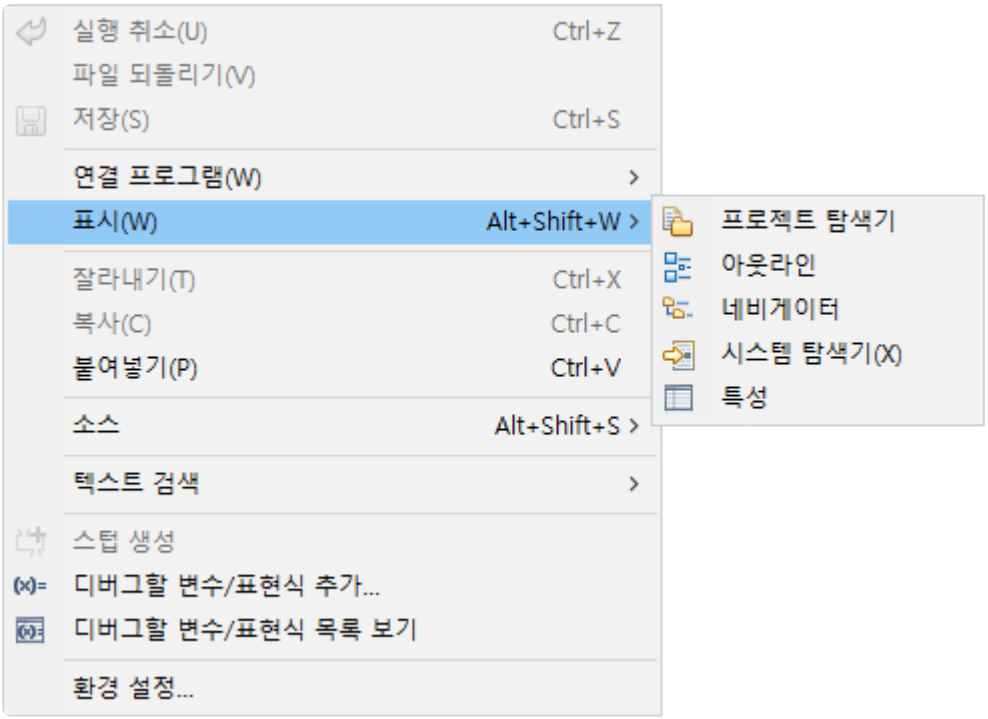
9. 소스 코드 탐색

CT 2023.12는 사용자의 편의를 위해 소스 코드 편집기 영역에서 단축키와 컨텍스트 메뉴를 제공합니다.

단축키

항목	단축키	설명
Include Browser 열기	`Ctrl` + `Alt` + `I`	[Include Browser 뷰]에서 선택된 파일의 포함 관계 표시
아웃라인 보기	`Ctrl` + `O`	아웃라인 팝업에서 선택된 파일의 아웃라인 표시
소스/헤더 파일 전환	`Ctrl` + `Tab`	소스 파일과 헤더 파일을 전환
계층 구조에서 유형 열기	`Ctrl` + `Alt` + `H`	[호출 계층 구조 뷰]에서 선택한 항목의 계층 구조 표현(함수/전역변수)
발생 표시 토글	`Alt` + `Shift` + `O`	블록을 지정하거나 커서가 위치한 항목에 대해 발생 표시를 켜거나 끔
선언 열기	`F3`, `Ctrl` + `Click`	선택된 항목의 선언으로 이동 또는 include 파일인 경우 파일 열기
자원 열기	`Ctrl` + `Shift` + `R`	이름으로 검색하여 파일 열기
참조 찾기	`Ctrl` + `Shift` + `G`	선택된 항목에 대한 참조를 [검색 뷰]에서 표시
히스토리 뒤로/앞으로	`Alt` + `Left` / `Right`	편집기 히스토리를 앞/뒤로 이동
텍스트를 파일에서 찾기	`Ctrl` + `K` / `Ctrl` + `Shift` + `K`	선택된 텍스트를 현재 파일에서 앞/뒤로 검색
펼치기 토글	`Ctrl` + `Numpad_Divide`	펼치기 아이콘을 보기/숨기기 토글
축소/확대	`Ctrl` + `-` / `Ctrl` + `Shift` + `=`	소스 코드 편집기를 축소/확대
펼치기/접기	`Ctrl` + `Numpad_Add` / `Numpad_Subtract`	커서가 있는 항목을 펼치기/접기
행 이동	`Alt` + ↓ / ↑	행을 아래/위로 이동
행 복사/중복	`Ctrl` + `Alt` + ↓ / ↑	행을 아래/위로 복사

컨텍스트 메뉴



항목	설명
아웃라인	[아웃라인 뷰]에서 현재 파일의 아웃라인을 표시
시스템 탐색기	현재 파일 위치를 윈도우 탐색기에서 열기

항목	설명
텍스트 검색	선택된 문자열을 대상 (워크스페이스/프로젝트/파일)에서 찾아서 [검색 뷰]에 표시