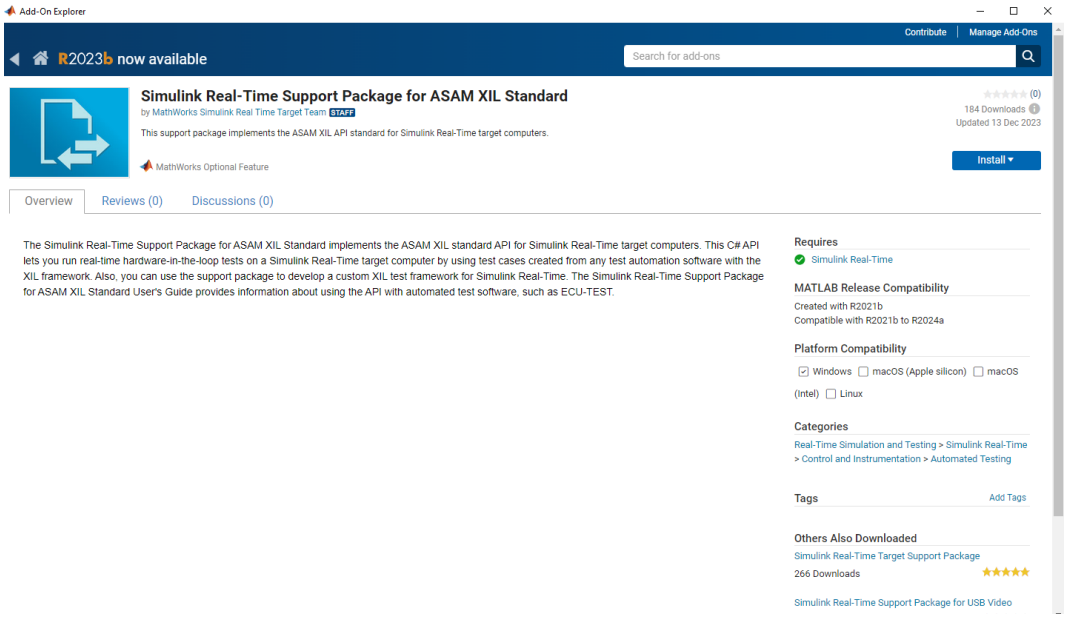
Test Automation with ecu.test using Simulink® and Speedgoat Test Systems

CONTENTS

1 Prerequisites for Using ecu.test

1.1 In MATLAB®, select Home > Add-Ons > Get Add-Ons and install the Simulink® Real-Time™ XIL API support package.



1.2 After support package installation, verify that the manifest file `MathWorksXILServer.imf` is located under `C:\ProgramData\ASAM\XIL\Implementation` and provides the correct Assembly path. For R2024a, that would be:
`C:\ProgramData\MATLAB\SupportPackages\R2024a\toolbox\slrealtime\xil\src\bin\win64`

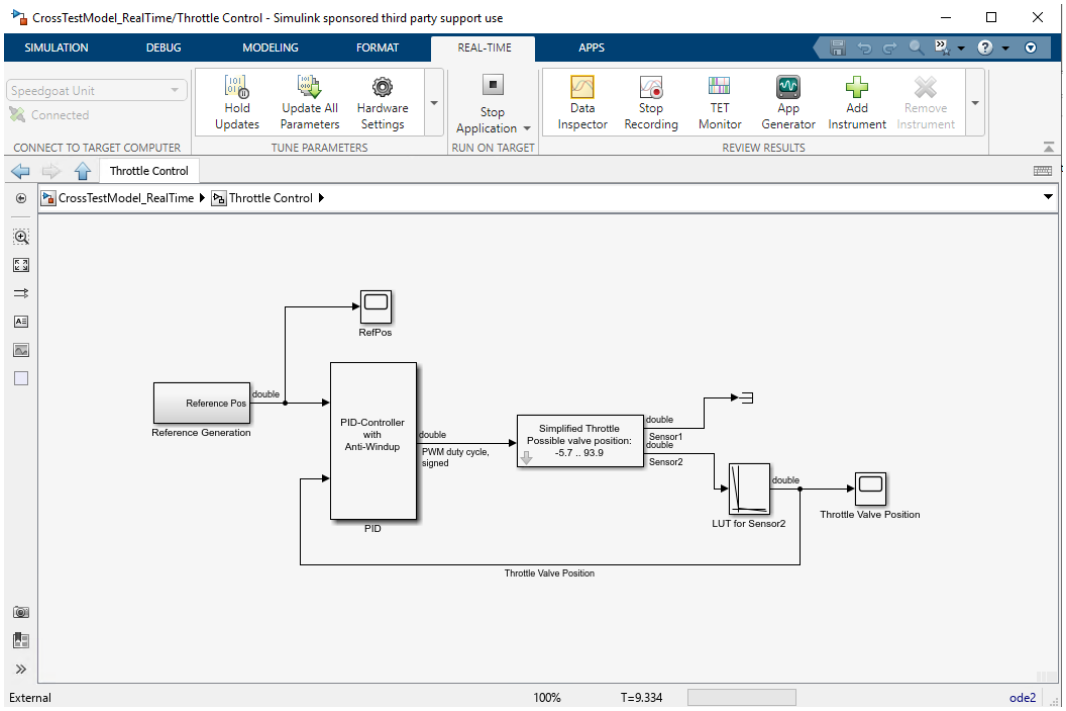
1.3 Register MATLAB as the automation server and share the MATLAB session by typing in the MATLAB Command Window:

```
comserver('register','User','current');
enableservice('AutomationServer', true);
```

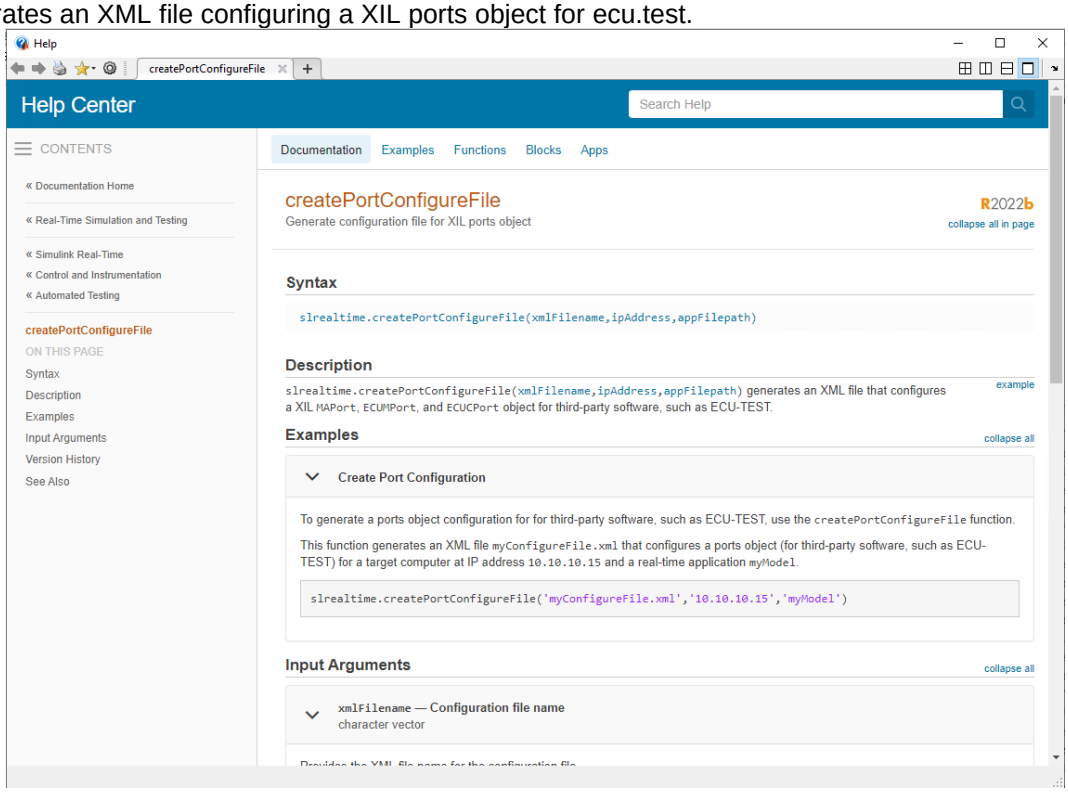
If you do not, ecu.test opens a new MATLAB session when configuring the test bench and test.

1.4 Connect your Speedgoat target machine to your development computer.

1.5 Build the model and click on “Run on Target”. Make sure this runs without any errors. The real-time application MLDATX file is required to set up the test bench and test in ecu.test.

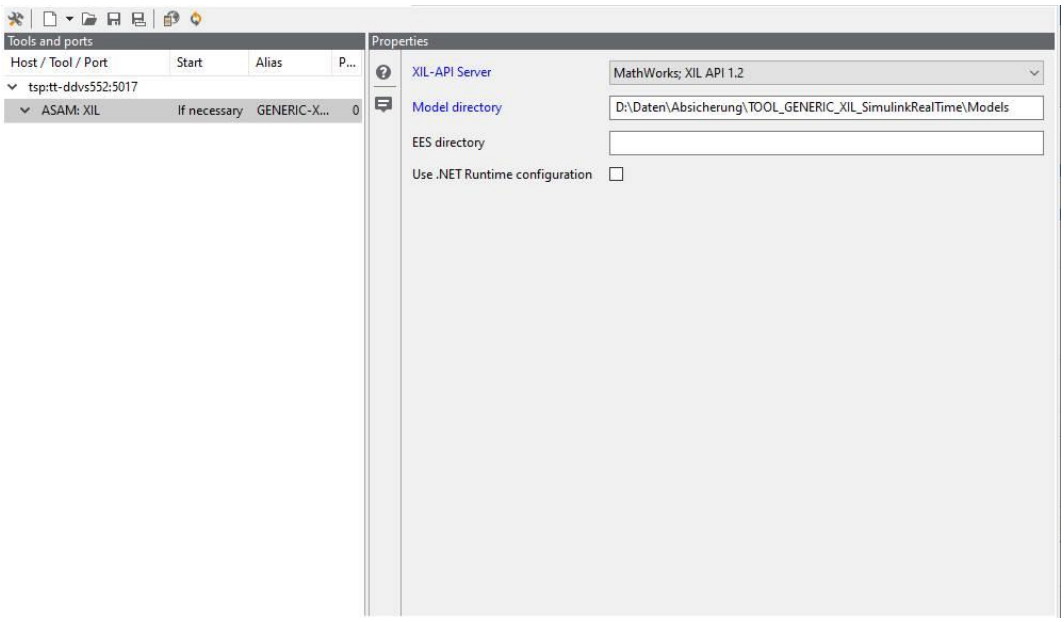


1.6 Generate XIL configuration with MATLAB command createPortConfigureFile.
The command:
`slrealtime.createPortConfigureFile(xmlFilename,ipAddress,appFilepath)`
generates an XML file configuring a XIL ports object for ecu.test.

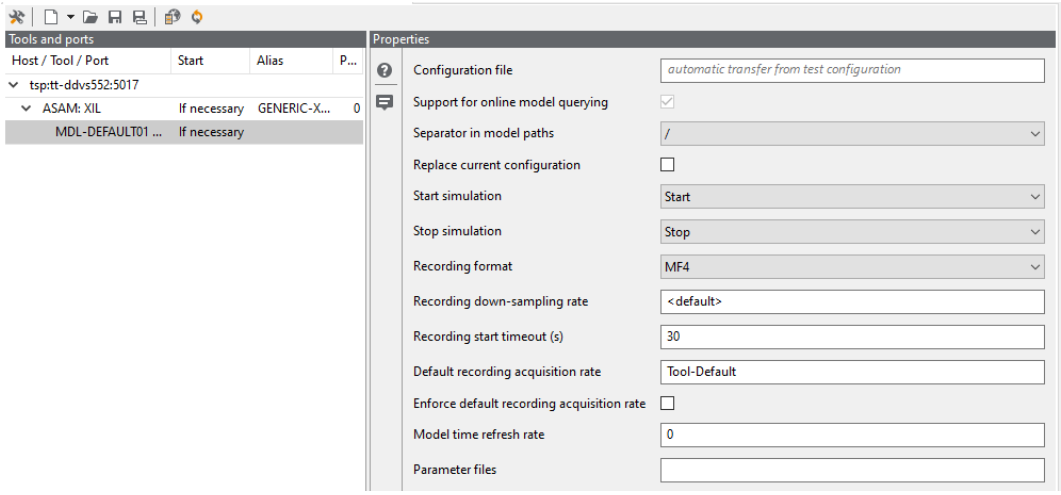


2 Create a New Test Bench

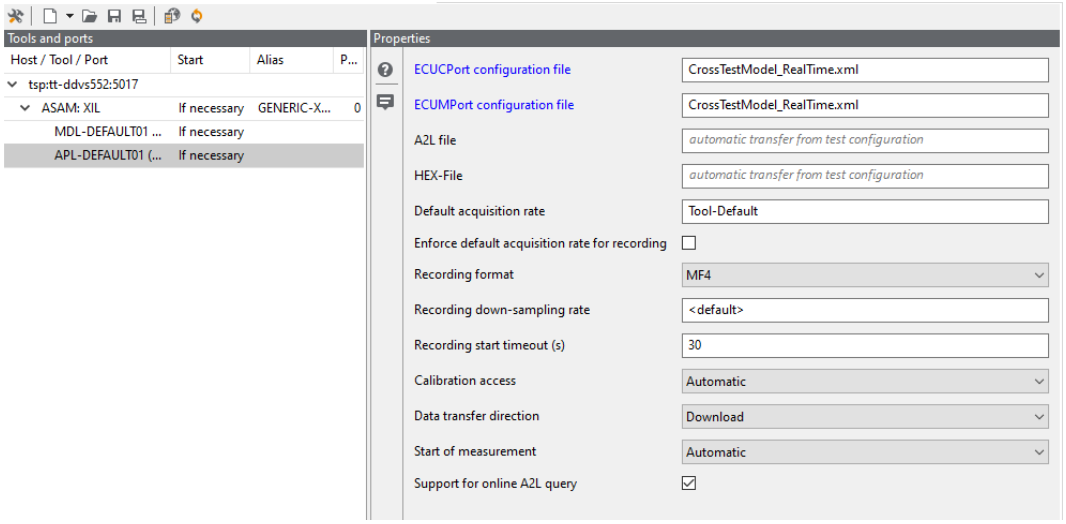
2.1 Open ecu.test. Select File > New > Test Bench Configuration. Add the tool “ASAM: XIL” and select the installed XIL-API Server from MathWorks® as shown below:



2.2 Create a Model Access Port for the test bench. Right-click ASAM: XIL and New Port -> Create Port -> Model Access Port. Edit the ‘Properties’ as shown below:



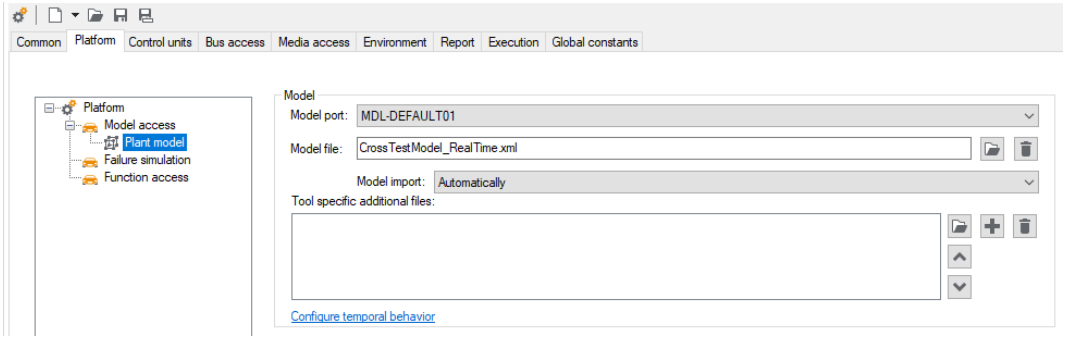
2.3 Create an Application Port for the test bench. Right-click ASAM: XIL and New Port -> Create Port -> Application Port. Edit the ‘Properties’ as shown below:



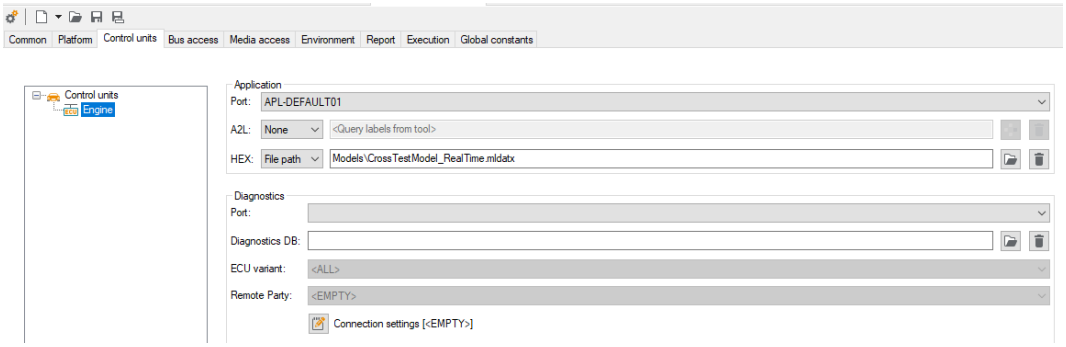
Note: The configuration files must point to the created config file from createPortConfigureFile.

3 Create a Test Configuration

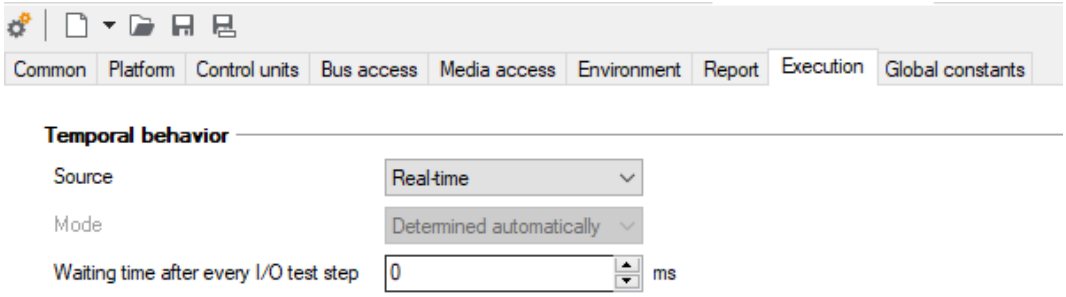
3.1 Under the ‘Platform’ tab, select ‘Model access’ and add a new model named ‘Plant model’. The ‘Model file’ for the ‘Model port’ must be set to the created .xml file from createPortConfigureFile.



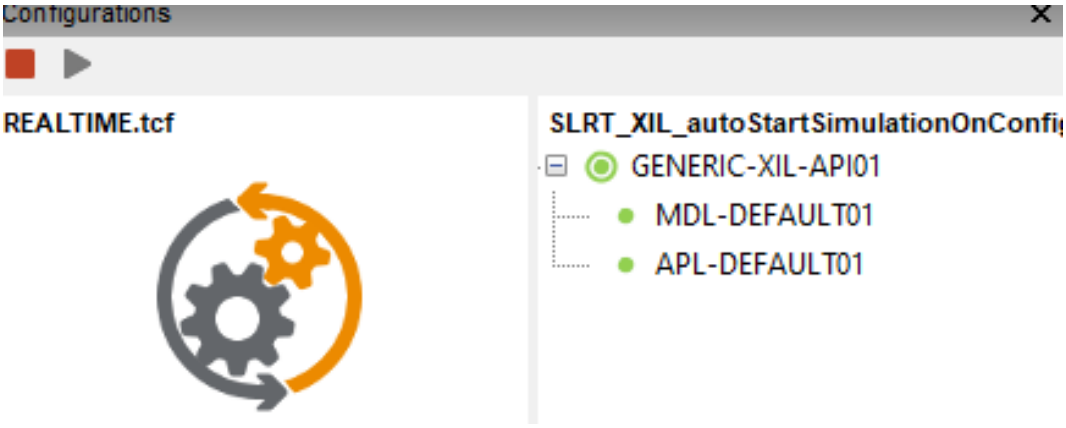
3.2 Under the ‘Control units’ tab, add a new control unit named ‘Engine’. The HEX file must be set to the MLDATX file from the real-time application.



3.3 Under the ‘Execution’ tab, the temporal behavior must be set to ‘Real-time’.



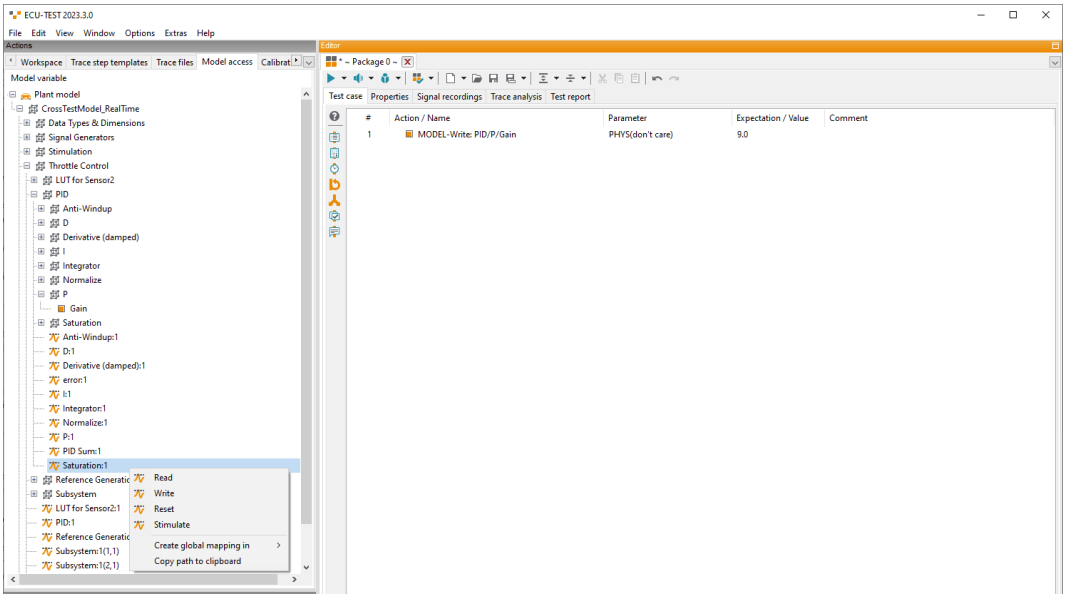
3.4 Select and load the configurations.



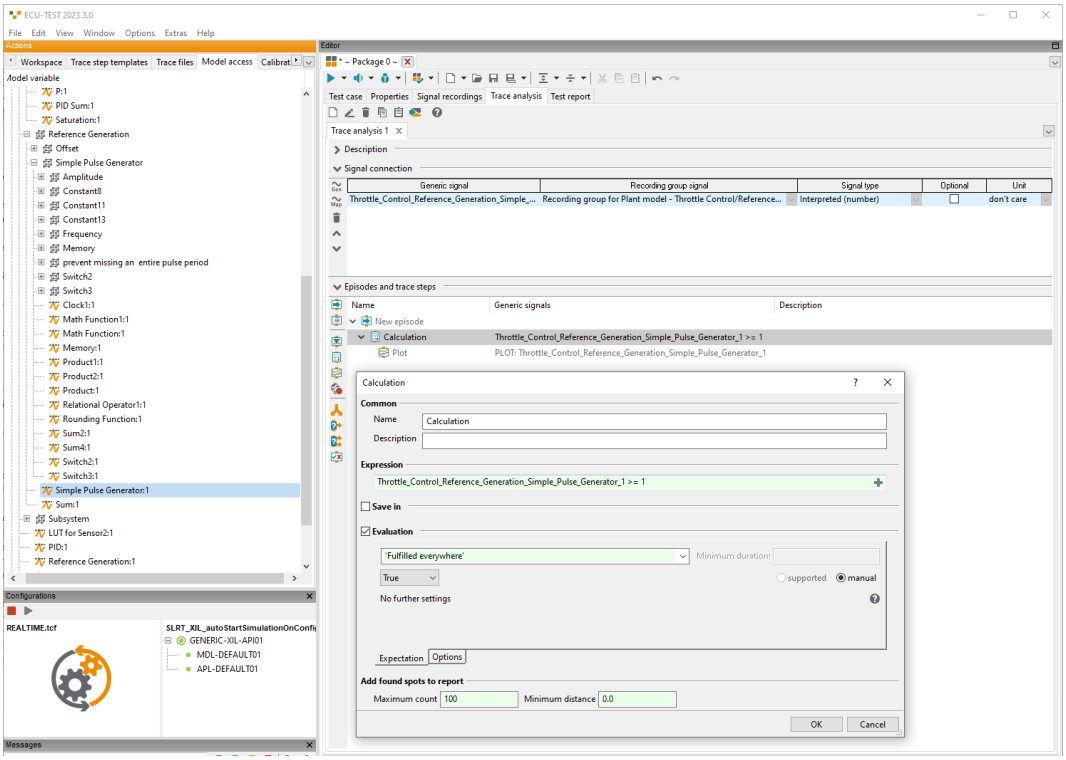
4 Create and Run Packages

In ecu.test, packages refer to the collection of test cases, test configurations, and related resources organized for efficient management and execution of test activities.

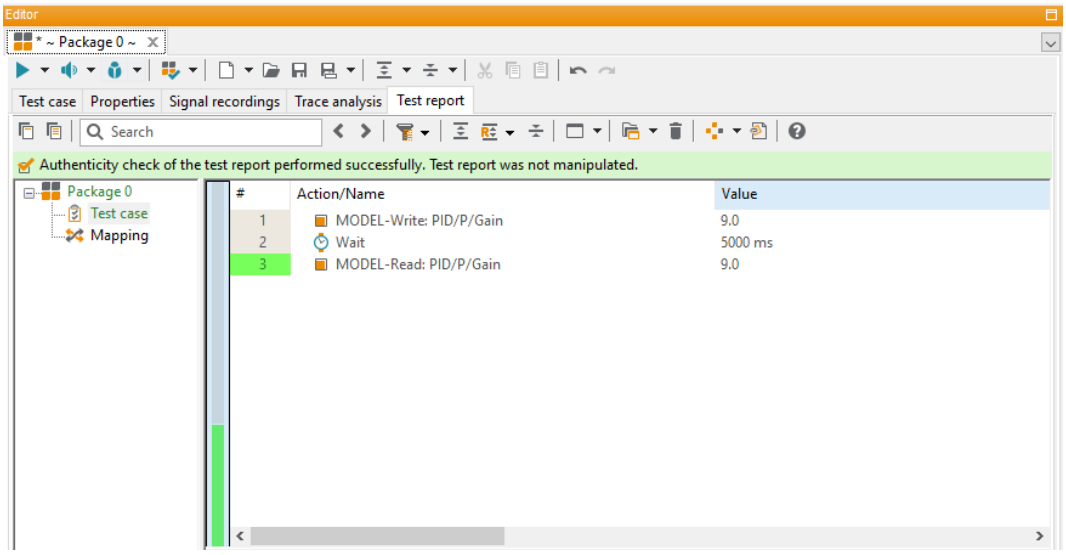
4.1 Create a new package and add the ‘Read’, ‘Write’, and ‘Stimulate’ steps from the model access tab to interact with the model.



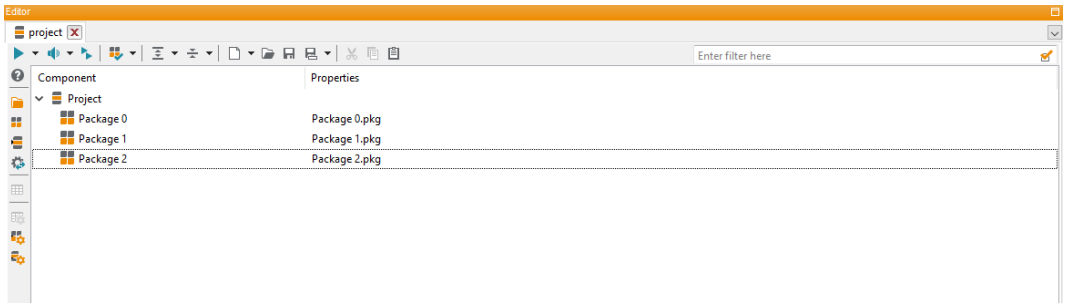
4.2 Add trace analyses and plots to check the behavior of signals over a specific period or the whole time.



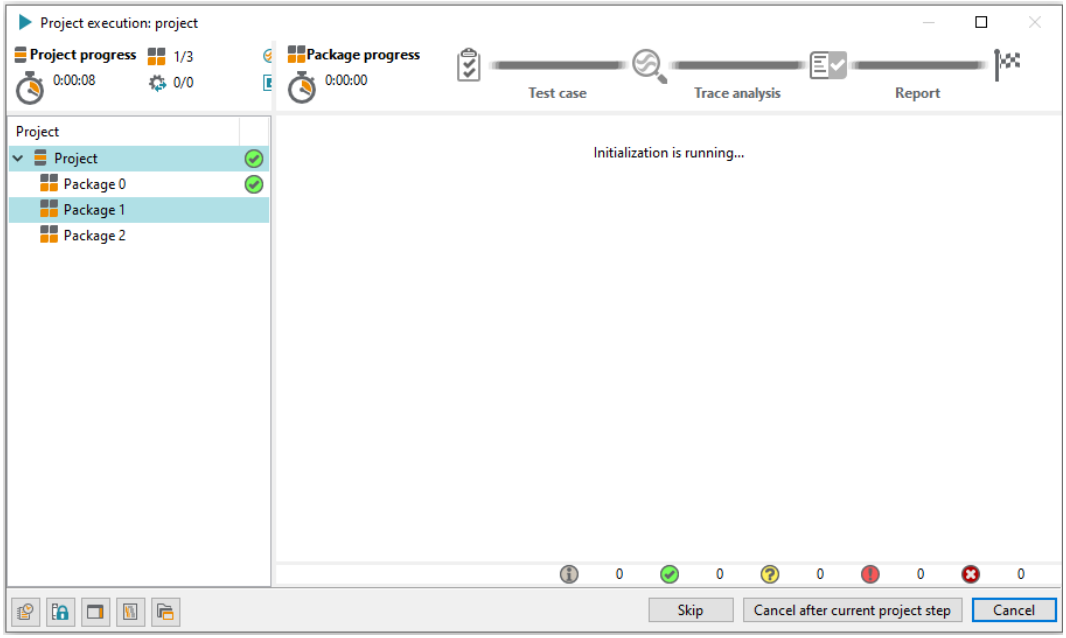
4.3 Run the package and check the report.



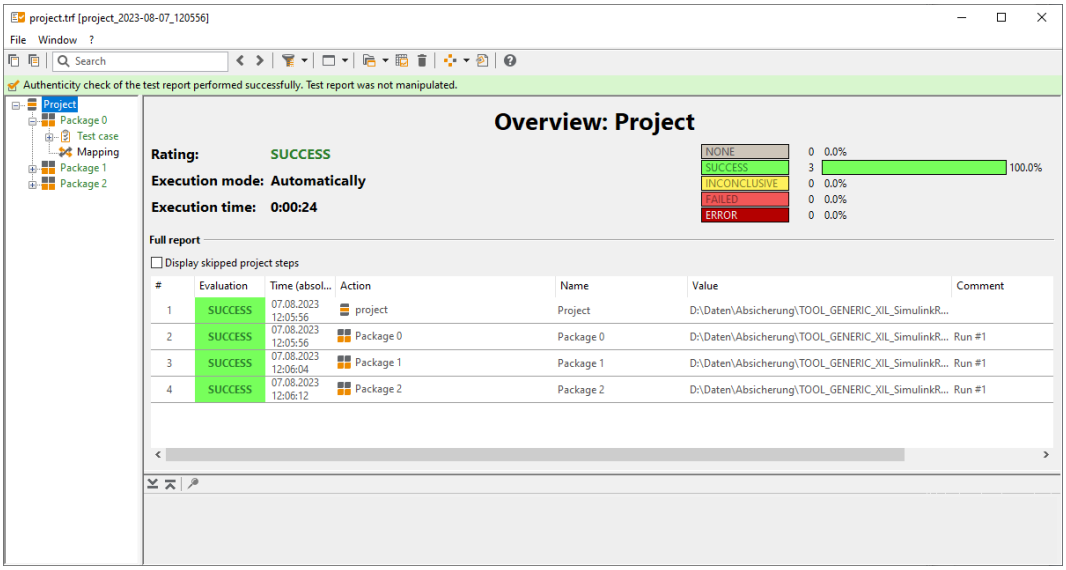
4.4 To execute several packages, create a new project and drag and drop the packages.



4.5 Execute the project.

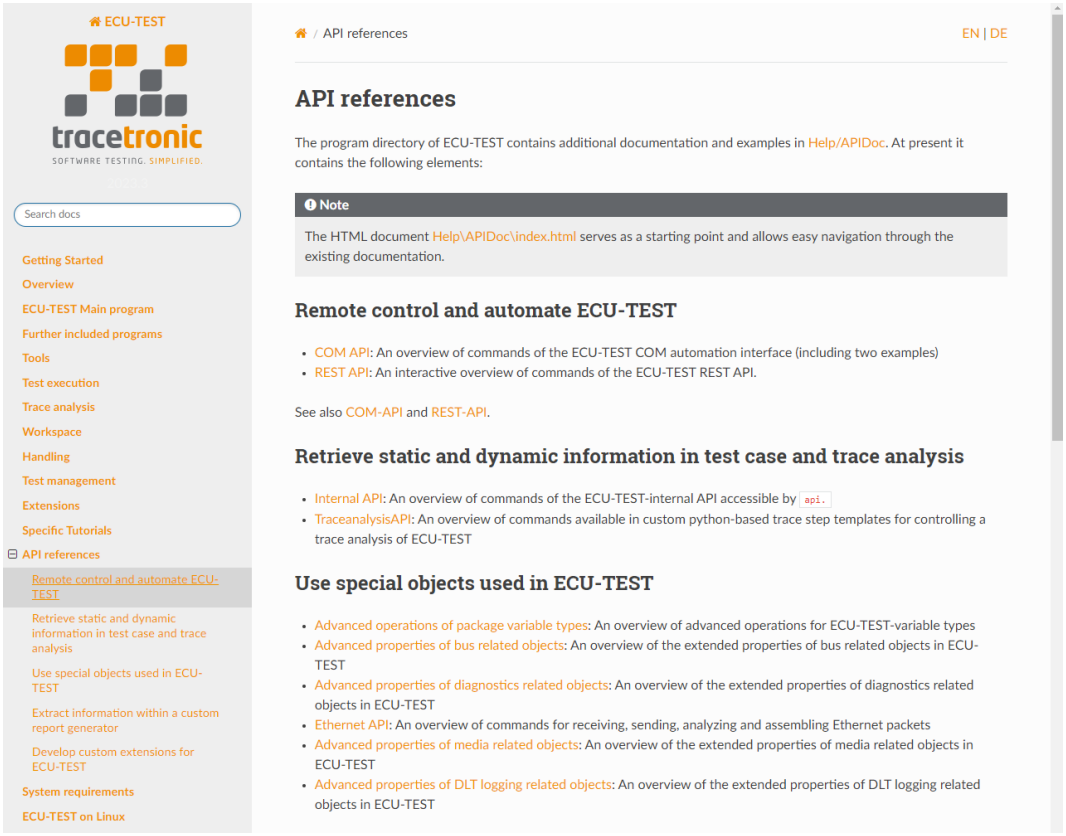


4.6 The report contains the runs of all three packages.



5 Test Automation APIs

5.1 Check out API references for the APIs required for test automation, such as REST and COM. All information can be found in the help.



5.2 Check out this example of using the REST API.

API documentation

Search docs

- Internal APIs
- Advanced operations of package variable types
- Advanced properties of bus related objects
- Bus related objects of trace analysis
- Advanced properties of diagnostics related objects
- Advanced properties of media related objects
- Advanced properties of DLT logging related objects
- COM-API
- REST-API
- Report API
- Object API
- TraceAnalysisAPI
- Generator APIs
- UserUtility API
- Utility Examples
- Tooladapter API

/ REST-API

REST-API

To control the test execution remote of ECU-TEST, an OpenAPI based REST-API exists as an alternative to the COM-API. It is available on both Windows and Linux.

When ECU-TEST is started, the API base path is locally available at http://127.0.0.1:5050/api/v1. An interactive OpenAPI documentation is accessible at 127.0.0.1:5050/api/v1/ui. By default, the API is only available locally. For remote access, start ECU-TEST with the command line option `--restApiEnableRemoteAccess`. In order to change the default port use the command line option `--restApiPort PORT` and replace "PORT" with a suitable port number.

A sample workflow with Python can look like this:

```python
from time import sleep
import requests

def WaitForOperationEnd(infoEndpoint):
    while True:
        info = requests.get(infoEndpoint)
        currentStatus = info.json()['status']['key']
        if currentStatus not in ['WAITING', 'RUNNING']:
            print(f'Finished! The status is {currentStatus}.')
            return info
        sleep(1)

BASE_URL = 'http://127.0.0.1:5050/api/v1'
LIVE_ENDPOINT = f'{BASE_URL}/live'
EXECUTION_ENDPOINT = f'{BASE_URL}/execution'

# Check if API is reachable
try:
    requests.get(LIVE_ENDPOINT)
except requests.exceptions.ConnectionError:
    raise RuntimeError('Cannot connect to ECU-TEST')

# Trigger a test run with specified package, TBC and TCF
myExecutionOrder = {
    'testCasePath': 'MyPackage.pkg',
    'tbcPath': 'MyTestbenchConfiguration.tbc',
    'tcfPath': 'MyTestConfiguration.tcf'
}
response = requests.put(EXECUTION_ENDPOINT, json=myExecutionOrder)
response.raise_for_status()  # Check if something went wrong


# Check if test run has finished
executionInfo = WaitForOperationEnd(EXECUTION_ENDPOINT)

# Upload the report to a TEST-GUIDE instance
reportId = executionInfo.json()['result']['testReportId']
uploadEndpoint = f'{BASE_URL}/reports/{reportId}/upload'
myUploadOrder = {
    'testGuideUrl': 'https://myTestGuideHost:1234',
    'authKey': 'abc123',
    'projectId': 3,
}
response = requests.put(uploadEndpoint, json=myUploadOrder)
response.raise_for_status()  # Check if something went wrong

# Check if upload has finished
WaitForOperationEnd(uploadEndpoint)
```

5.3 Use the TEST-GUIDE to handle the rising number of reports. TEST-GUIDE also offers many automation features, such as executing projects with an intelligent distribution of available resources out of the box. See here: https://www.tracetronic.com/products/test-guide/