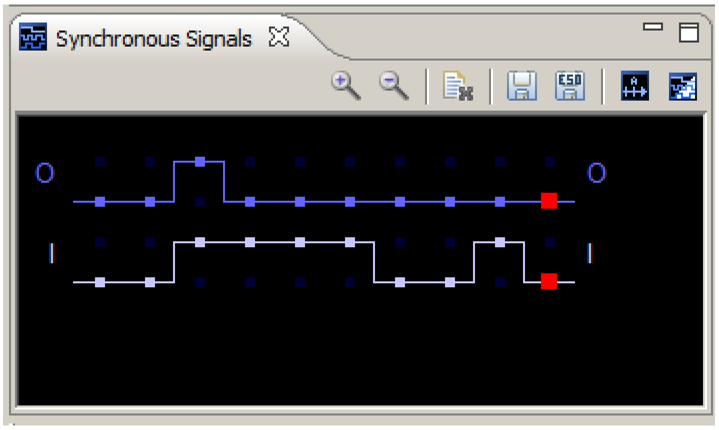
Regression Testing (JUnit & KART)

KIEM can be used for regression tests. Typically for a regression test one wants to compare a simulator's reaction to an expected reaction.

Record ESO Trace Files

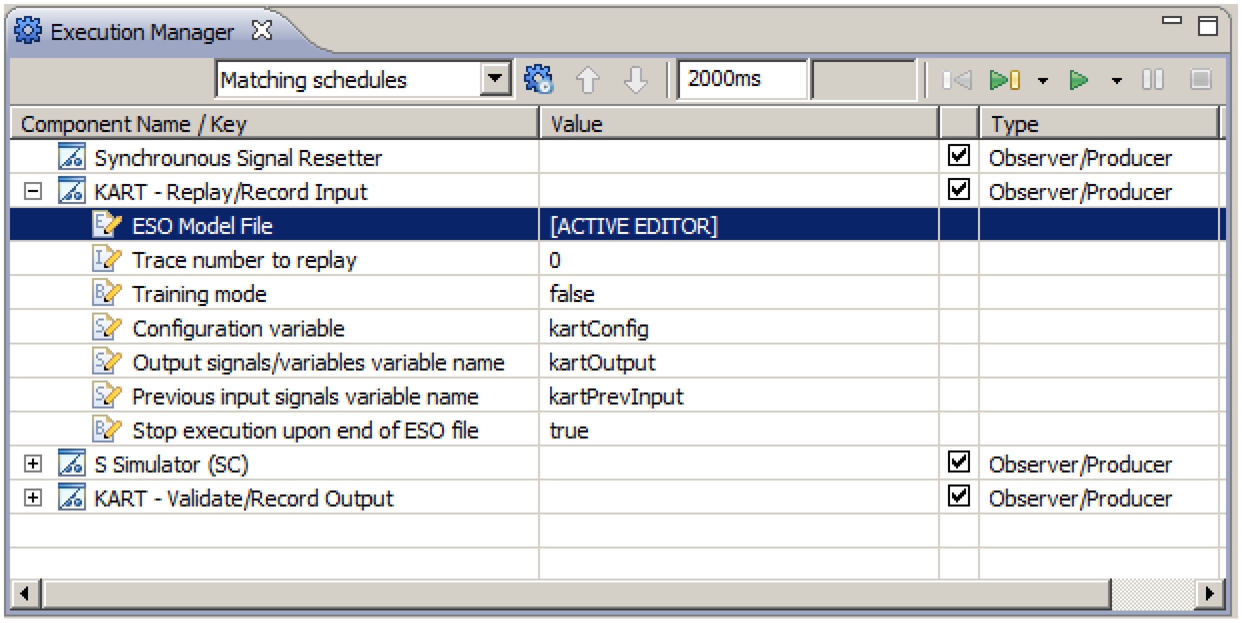


When simulating using KIEM and the Synchronous Signals View there is a button ("disk with ESO") in this View that allows saving signal data traces as ESO files. Overwriting an ESO file results in appending a new signal data trace to it. If you want a clean ESO file, first delete the existing one and then save the current trace.

An alternative is to use the KART DataComponents in training mode. The training mode can be configured in the KIEM property options. In training mode the components will not validate the output of a simulator but record its output.

Validate ESO Trace Files

For validation the KART DataComponents must be scheduled in the correct order. A typical order can be seen here:



Component Name / Key	Value		Type
Synchrounous Signal Resetter		✓	Observer/Producer
KART - Replay/Record Input		✓	Observer/Producer
ESO Model File	[ACTIVE EDITOR]		
Trace number to replay	0		
Training mode	false		
Configuration variable	kartConfig		
Output signals/variables variable name	kartOutput		
Previous input signals variable name	kartPrevInput		
Stop execution upon end of ESO file	true		
S Simulator (SC)		✓	Observer/Producer
KART - Validate/Record Output		✓	Observer/Producer

The KART Replay DataComponent should be placed before the Simulator DataComponent (here: S Simulator) and the KART Validation DataComponent should be placed behind. When \[ACTIVE EDITOR\] is selected the current ESO file to be used is calculated from the name of the model file currently open in the editor (by replacing the model file suffix with eso). In case of an automated (and even headless) JUnit regression test (see below) KIEM will provide this model. Other Workspace related ESO files can be specified here.

Automated JUnit Regression Tests

It is relatively simple to run KIEM with KART in an automated setting on a bunch of model and ESO files. A test-plugin must be created for this purpose for each simulator (or other types of similar DataComponents) to be tested. An execution file (=persistent KIEM schedule like for example the one above) must exist To create such a schedule click into the KIEM view an after modification select Save as.

In the defined bundle create a folder that contains all model files and all ESO files. The model files are later fed into all DataComponents using the Simulation DataComponent interface (except for the KART component where the corresponding ESO files will be used). A necessary convention is that the model files have the same name as the corresponding ESO files!

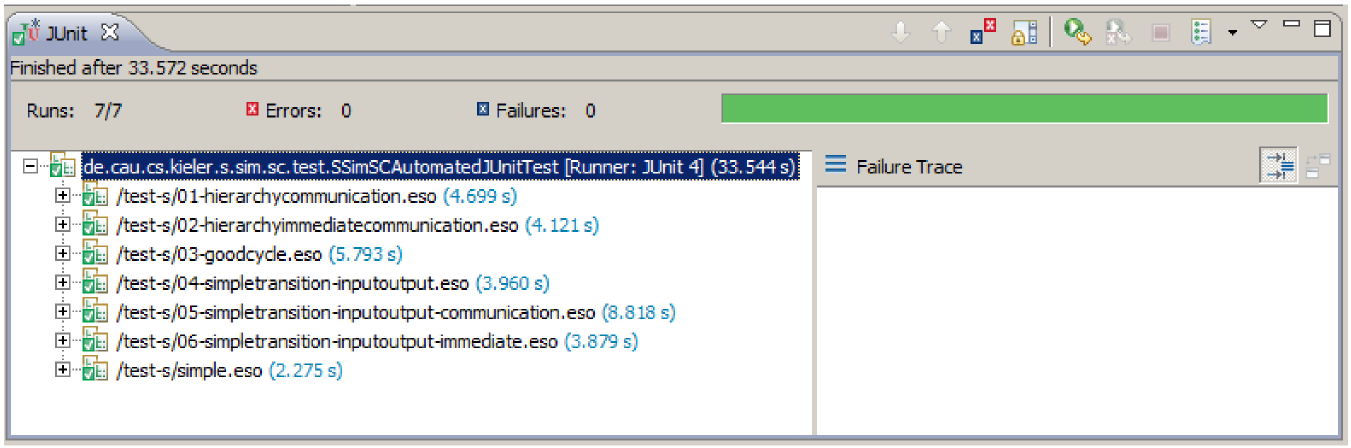
Then create a new Class that extends KiemAutomatedJUnitTest and that contains "Test" in its name. See the following example:

MyAutomatedJUnitTest

```
public class SSimSCAutomatedJUnitTest extends KiemAutomatedJUnitTest {
    protected String getPluginId() {
        return SSimSCTestPlugin.PLUGIN_ID;
    }
    protected IPath getBundleTestPath() {
        return new Path("testdata");
    }
    protected String getModelFileExtension() {
        return "s";
    }
    protected String getTemporaryWorkspaceFolderName() {
        return "test-s";
    }
    protected String getExecutionFileName() {
        return "automated.execution";
    }
}
```

A test plugin must supply its PluginID. It further must specify the bundle relative path to the execution file (the KIEM schedule including KART components like the one above). It must supply the bundle relative IPath to the model and eso files (here: /testdata/). It must supply the file endings of the models, corresponding eso file names will be constructed, other files will be ignored. A temporary workspace folder name must be added because links in the test Workspace will be created that point to the supplied bundle files.

If all this is done, the automated execution can be run via a JUnit Plugin test run configuration:



Finished after 33.572 seconds

Runs: 7/7   Errors: 0   Failures: 0

```
de.cau.cs.kieler.s.sim.sc.test.SSimSCAutomatedJUnitTest [Runner: JUnit 4] (33.544 s)
    /test-s/01-hierarchycommunication.eso (4.699 s)
    /test-s/02-hierarchyimmediatecommunication.eso (4.121 s)
    /test-s/03-goodcycle.eso (5.793 s)
    /test-s/04-simpletransition-inputoutput.eso (3.960 s)
    /test-s/05-simpletransition-inputoutput-communication.eso (8.818 s)
    /test-s/06-simpletransition-inputoutput-immediate.eso (3.879 s)
    /test-s/simple.eso (2.275 s)
```

Upon failure the execution file, the model file the trace file number, the tick in which the error occurred will be displayed. Additionally the expected signal and the observed signal is presented in the error message of the first failing tick. This can later be inspected using the same execution schedule, model and ESO file in a typical Eclipse application run configuration setting with visual KART feedback to debug the problem.