Caraya is an assertion and unit test framework for LabVIEW that is simple and fast. It takes a whole new approach to unit testing; your VI is your test. Caraya allows you to convert your manual test VIs you use to debug your code into unit test cases with nearly no effort. This significantly lowers the barrier to systematicaly write unit tests for your project leading into improved overall code quality for real-world projects where developers don't always have the luxury to write unit test cases first.

Installation

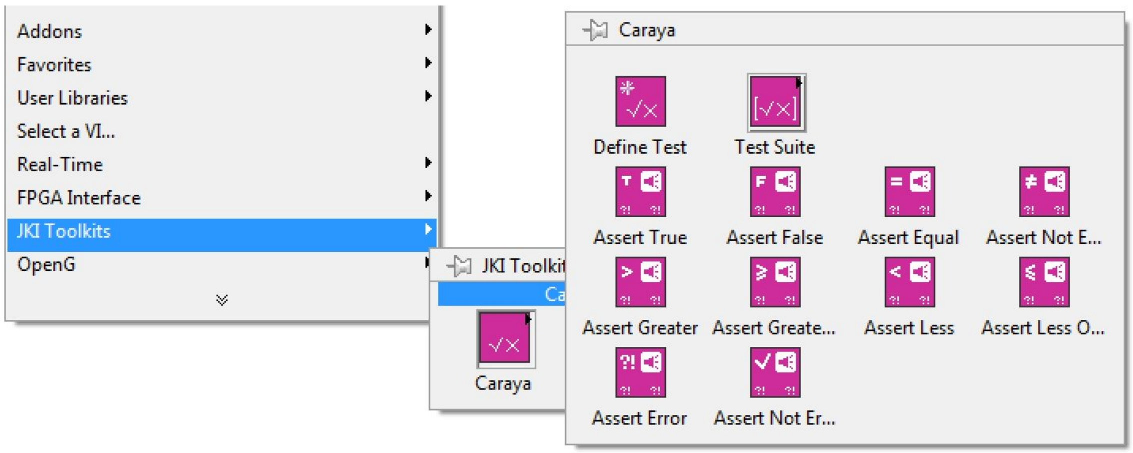
You can download and install Caraya with VI Package Manager.

Get Caraya

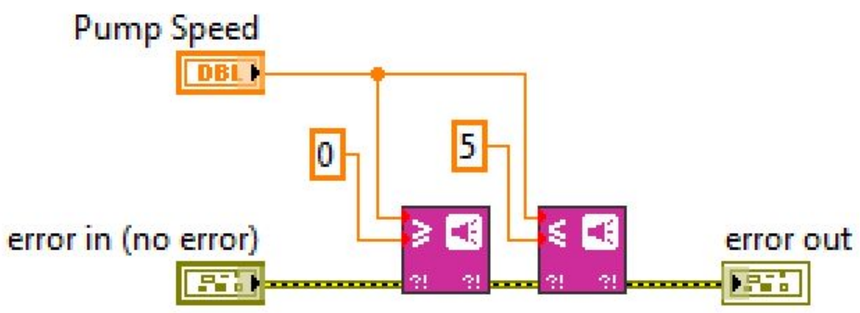
Usage

Caraya is a LabVIEW toolkit providing a library of assertion VIs to validate run-time constraints of any LabVIEW application and providing support VIs to construct unit test cases from any VI using assertions.

###Palette To create unit tests or to validate runtime conditions in your application, you need to drop the corresponding Caraya toolkit VIs to the block diagrams. The Caraya toolkit VIs are located under the JKI Toolkits functions palette menu.



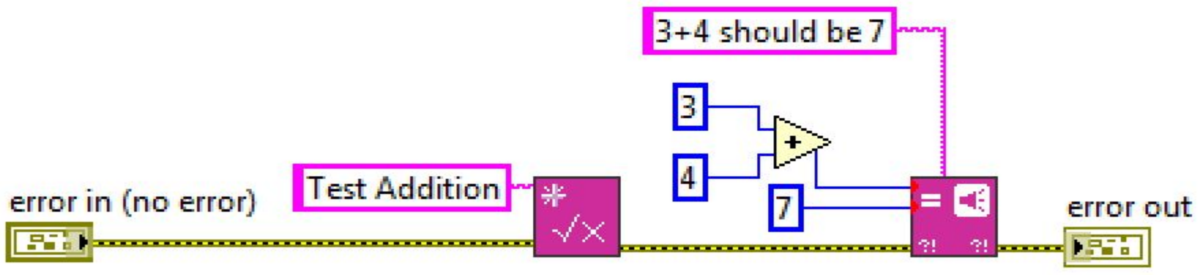
###Assertions Caraya provides a library of assertion VIs that return an error if the constraint requirements for the assertion VI are not met. You can use these anywhere in your application to verify that values are within meaningful range and generate and error if values don't make sense. For example if you are setting the speed for a pump, you may want to make sure that the speed is never negative and is below the maximum allowed pump speed. Assertation functions execute fast in your executable and provide only small performance penalty and as such can be used nearly everywhere in your code.



If the constraint requirement is not met, the assertion returns an error. If the input to the assertion VI already has an error, the preceding error will be passed forward.

###Unit Tests ####Simple Test Cases Caraya allows you to turn any VI into a unit test case simply by dropping a Define Test VI to the block diagram. The unit test is executed simply by clicking the run arrow of the VI. To define pass and fail criteria, simply drop one or more assertion VIs to your unit test VI and wire your test conditions to these assertion VIs.

For example to define a test case for an addition operation, drop the Define Test VI and the Assert Equal VI to your block diagram. Wire your test case condition to the Assert Equal node as shown in the picture below. Now you're complete with your first test case, nothing else is required. Simply click the run button of your VI and the Caraya user interface pops up with the results of your test.

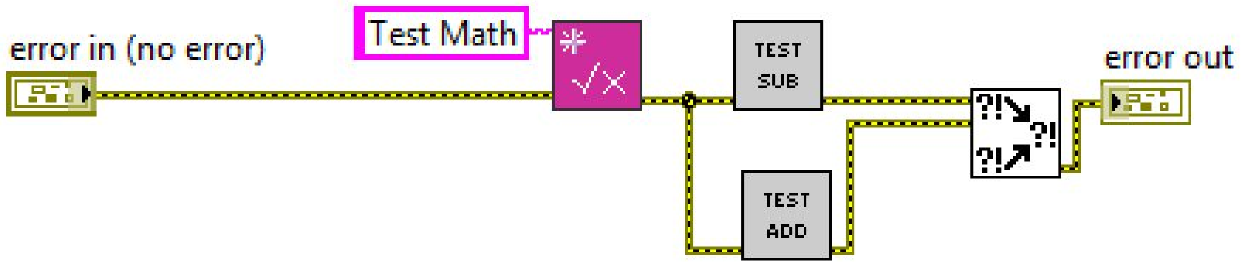


Note that the Define Test VI needs to execute before any of the assertion VI for the assertion VIs to register to be part of the unit test.

####Hierarchical Tests It makes sense to divide your tests into multiple entities each testing a single logical aspect of your application. You can compose test hierarchies consisting of multiple logical test cases simply by adding your test cases as SubVIs of a "parent" test case. When running the top level test VI, all the subVIs containing test cases will also be executed. You can have as many levels of of VI hierarchy as you see necessary. Caraya doesn't limit the number of levels in your hierarchy. A meaningful test hierarchy would look something like the following:

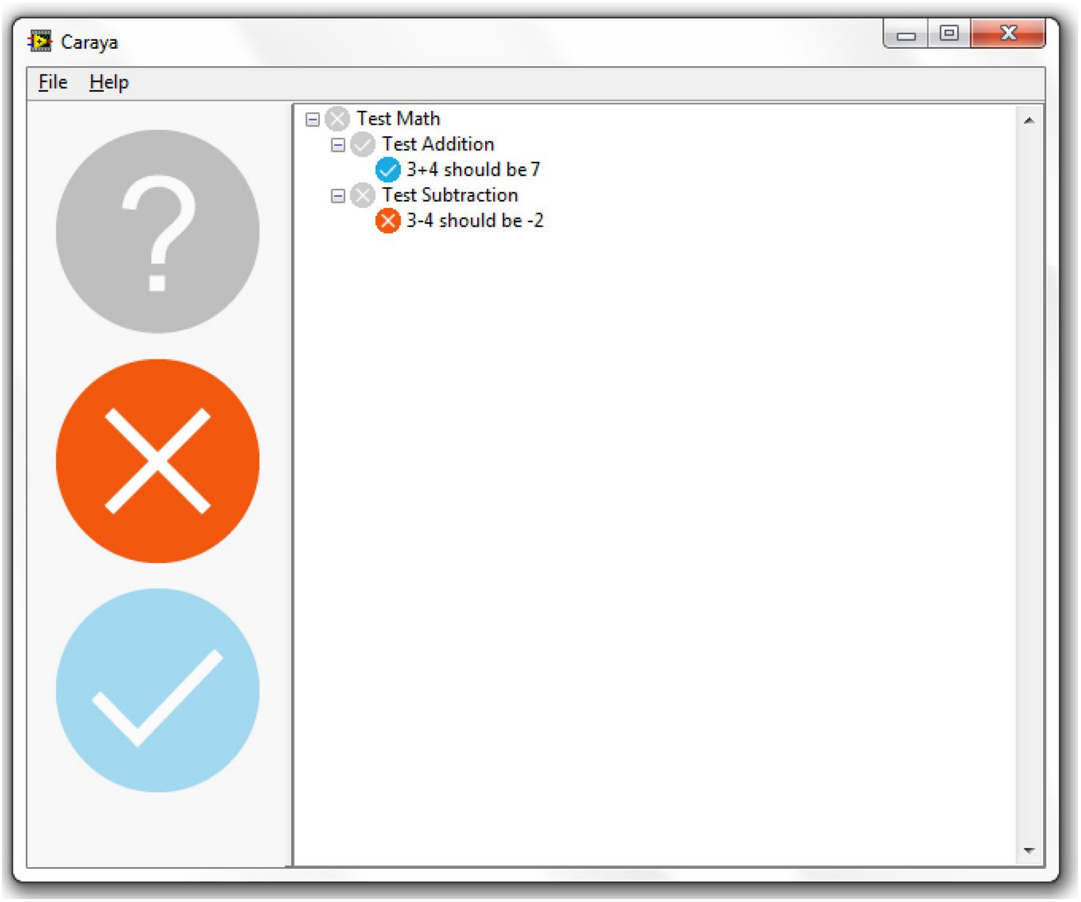
```
- Test Application
  - Test Pump Operations
    - Test Pump On/Off
     - Test Set Speed
     - Test Read Speed
  - Test Valve Operations
    - Test Valve Open/Close
     - Test Read Valve State
  - Test Container Liquid Handling
    - Test Fill Container
     - Test Maintain Container Level
     - Test Empty Container Level
     - Test Read Container Level
```

In the previous section we created a test case for the addition operation. Now if you create a test case for subtraction the same way, you can create higher level test VI consisting both addition and subtraction tests simply by dropping each of the two individual test cases on the block diagram of a the parent test case as shown below.



Normal execution rules apply. If you add your tests in parallel as above, both tests run in parallel and independent of each other. If you place your tests sequentially, normal error wire behavior applies and the second test won't run if the first test fails.

####Test Result Dialog When you run a VI with unit tests, a dialog will pop up reporting you the results of the unit tests. The results are being reported in the same hierarchical fashion as your constructed your test cases.



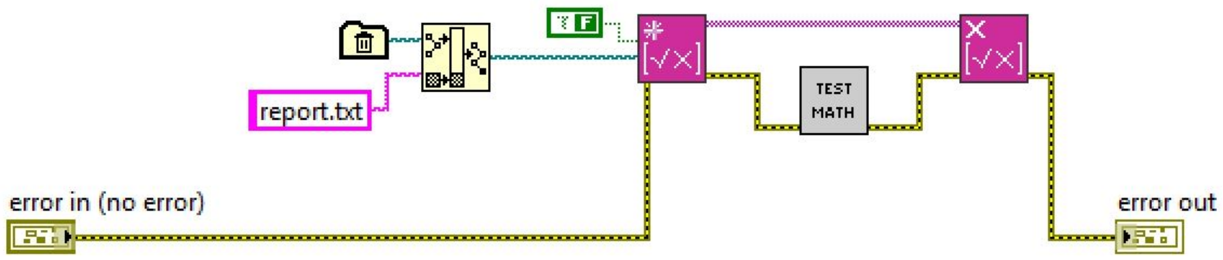
Test Math
Test Addition
3+4 should be 7
Test Subtraction
3-4 should be -2

If your all your tests pass, you can simply close the dialog and continue your development. If your tests fail, you can debug and edit your test cases and application source code without closing the dialog. Or feel free to close the dialog, you can simply open it by rerunning any of your tests.

####Automated Tests Sometimes you want to run your tests as part of an automated build process. When automating the execution of the unit tests, you typically want your test results to be written into a report file. You also probably don't want to see the interactive window but rather run your tests completely headless. Most automated build systems also need some sort of indication when the tests are completed and if they passed or failed. Caraya supports these scenarios with a concept of test suites. A test suite is a unit of automation for Caraya that provides you more fine grained control of how the test cases are being executed.

A test suite is nothing more than a typical test case with few differences. As test suites are the unit of automation and you should probably only define one test suite for your whole unit test hierarchy that simply calls your top level test case.

To define a test suite you can provide a path for the report file as well as define if you want the test cases to be executed in a headless mode. For example to run the test cases for the mathematical functions that we defined above, you simply drop the top level test case to your block diagram together with Define Test Suite VI and Destroy Test Suite VI.



The Define Test Suite VI tells Caraya to generate a report file and to run the tests in a headless mode. The Destroy Test Suite tells Caraya that the test suite has completed executing. Caraya passes the outcome of the whole test suite in the error wire of the Destroy Test Suite i.e. if any of the tests failed the error output would contain an error. You can use this error to tell your automated build if the unit tests succeeded or failed.

Examples

You can find examples on how to use Caraya under the LabVIEW examples directory

```
<LabVIEW>\examples\JKI Toolkits\Caraya
```

Contributing

1. Fork it!
2. Create your feature branch: `git checkout -b my-new-feature`
3. Commit your changes: `git commit -am 'Add some feature'`
4. Push to the branch: `git push origin my-new-feature`
5. Submit a pull request

To contribute to Caraya, you will need 32-bit LabVIEW 2009 SP1 professional development environment.

Credits

Caraya is an open source project maintained by JKI.

License

Caraya is distributed under the open source three clause BSD license providing everyone right to use and distribute both souce code and compiled versions of Caraya. See LICENSE.md file for details.