

Using the OpenTAP SDK



Developer Training

OBJECTIVES

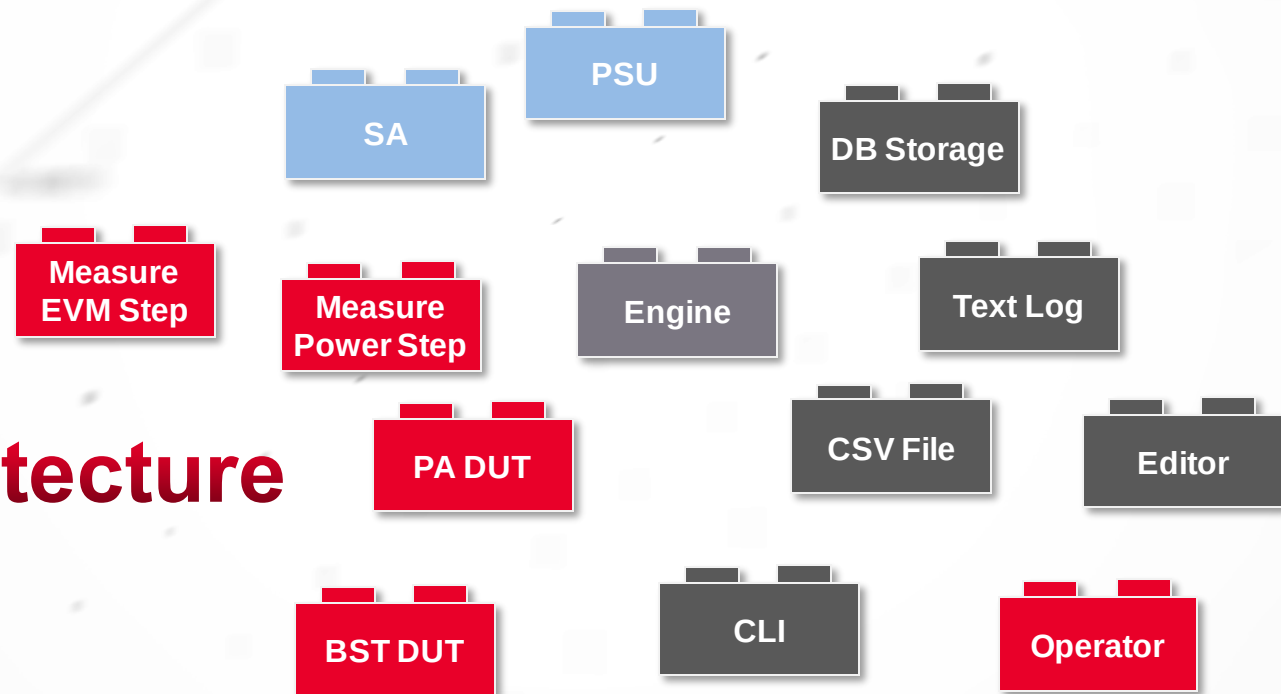
- Introduce you to the OpenTAP Architecture
- Show how the OpenTAP SDK integrates with Visual Studio
- Provide an overview of OpenTAP Plugin Development

Agenda

BASICS OF OPENTAP DEVELOPMENT

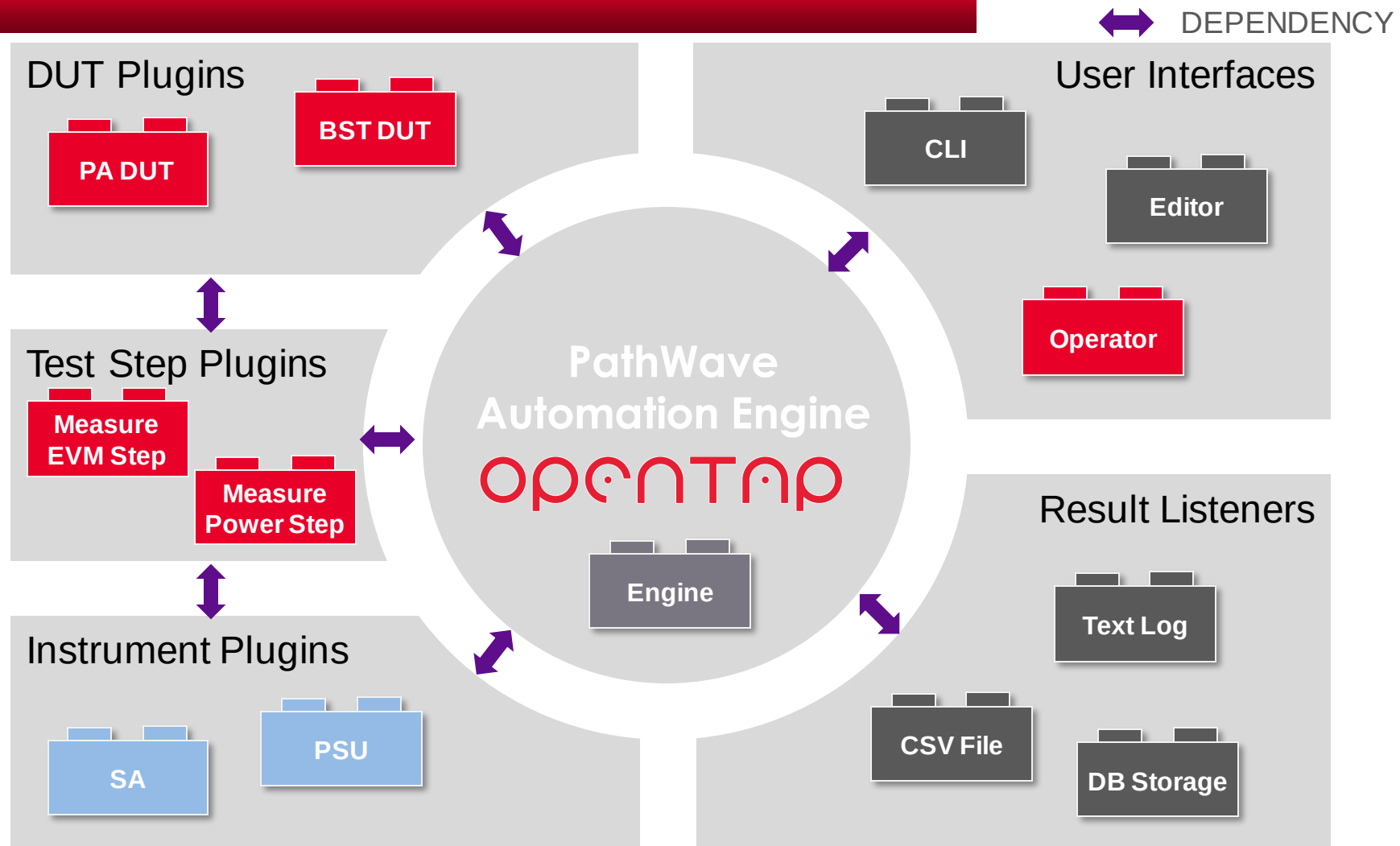
- TAP Architecture
- Getting Started in Visual Studio
- Test Steps
- Resources
- Instruments
- DUT
- Result Listeners
- Plugin Packaging
- Results System
- Viewing Results
- Extending TAP

OpenTAP Architecture



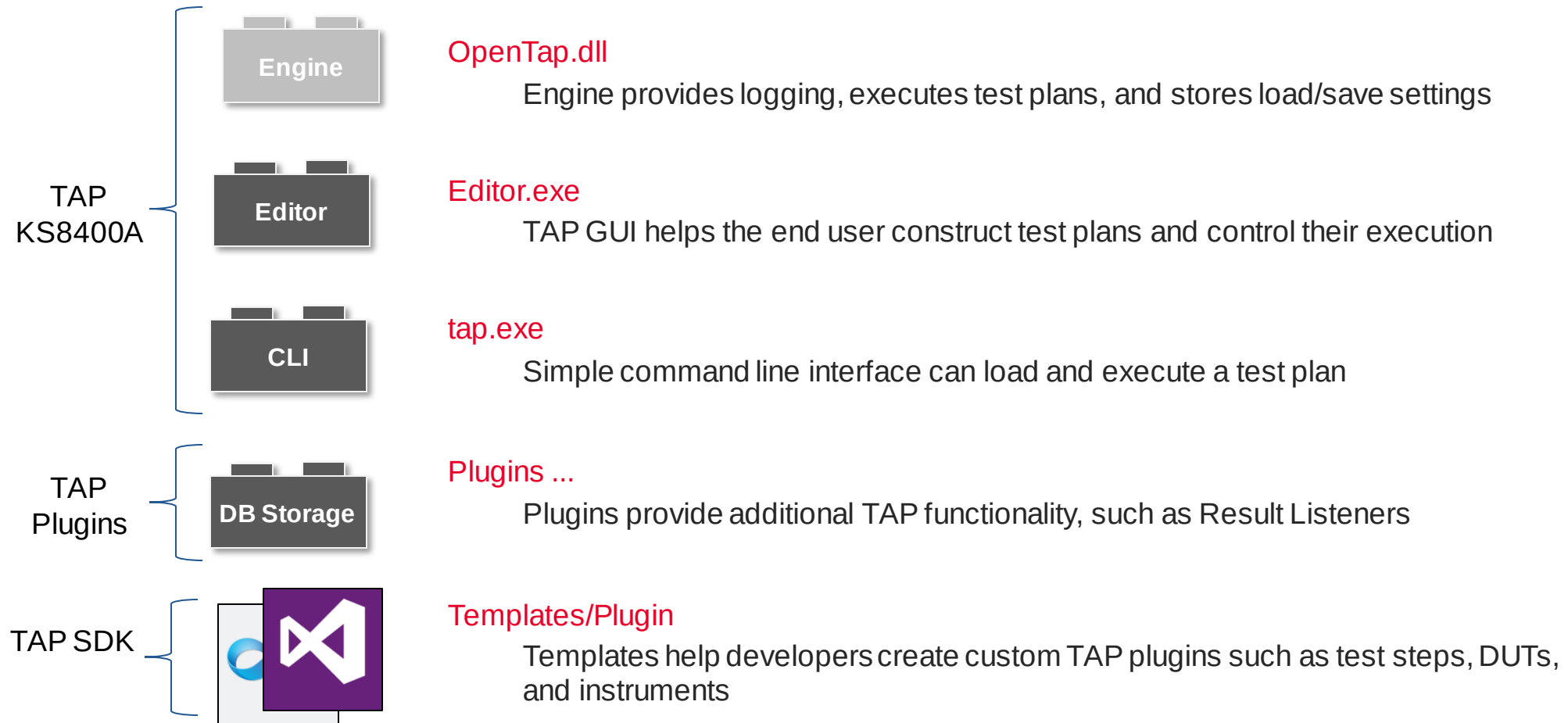
OpenTAP Architecture

OVERVIEW

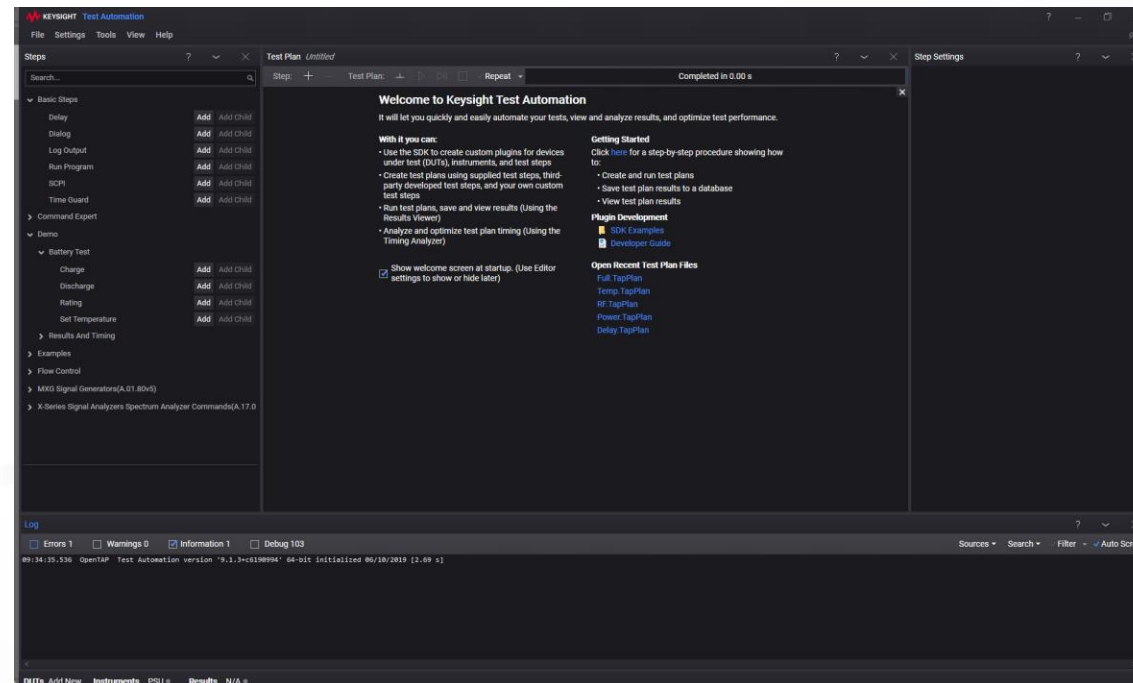


OpenTAP Architecture

WHAT'S PROVIDED



Getting Started



Getting Started

INSTALLING PATHWAVE TEST AUTOMATION

1. Go to <http://keysight.com/find/tap>

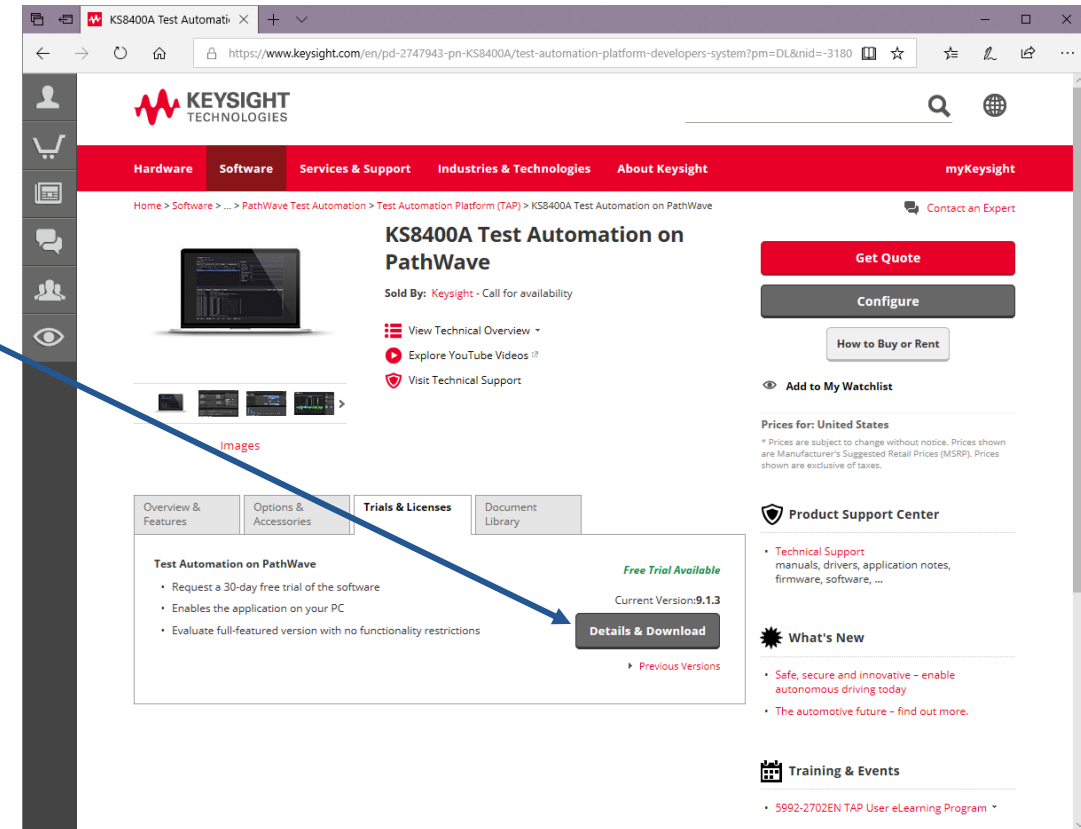
Notice the resources in the Document Library

2. Download the **Test Automation** installer

3. Run Installer

- (30 days free trial license is available)

Now you are ready to get started!



Getting Started

RUNNING YOUR STEP

Test Step Panel

View the sequence of test steps that comprise the test plan

Test Plan Control

Add, remove, run and stop the test plan

Step Settings Panel

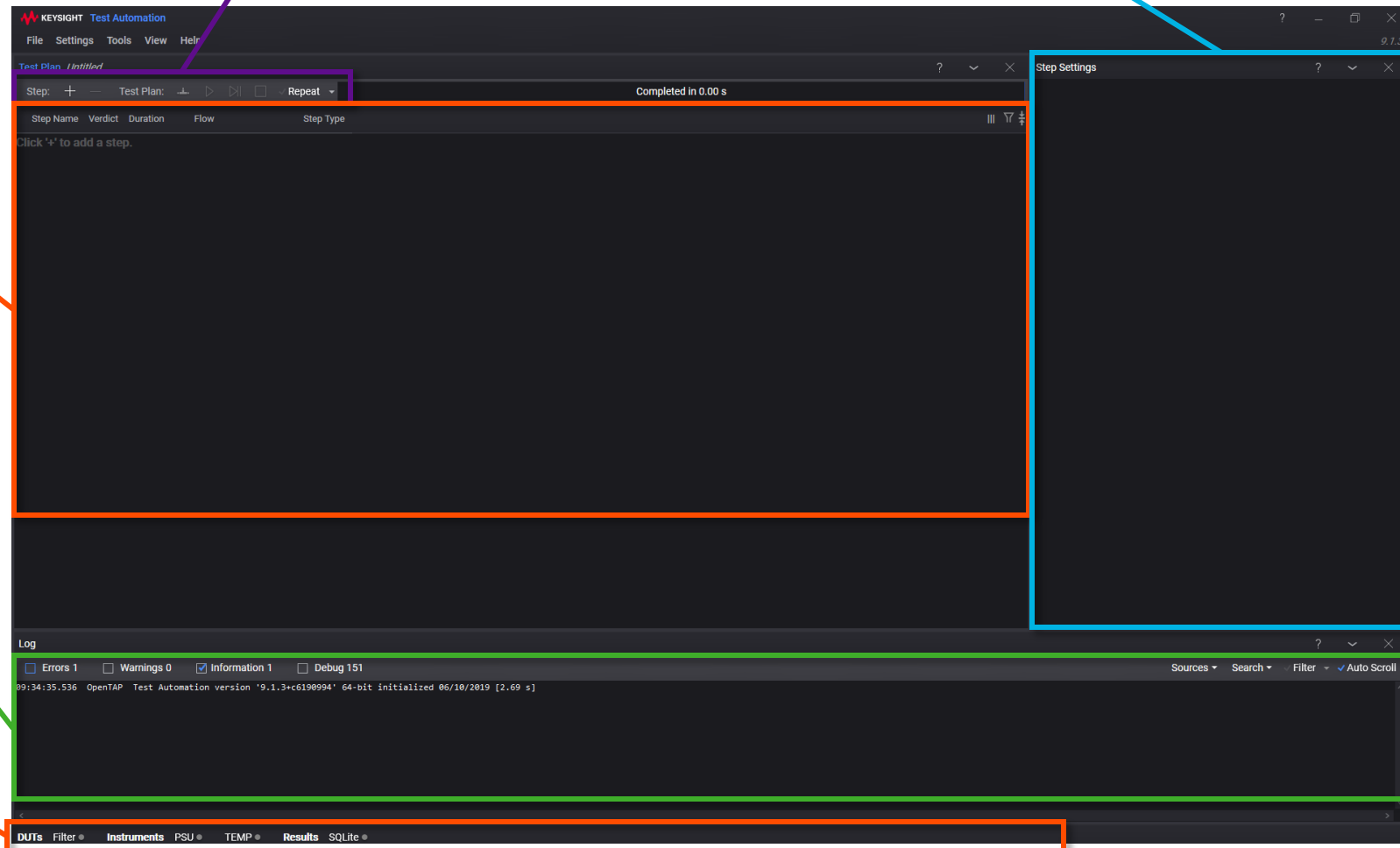
Configure the selected step

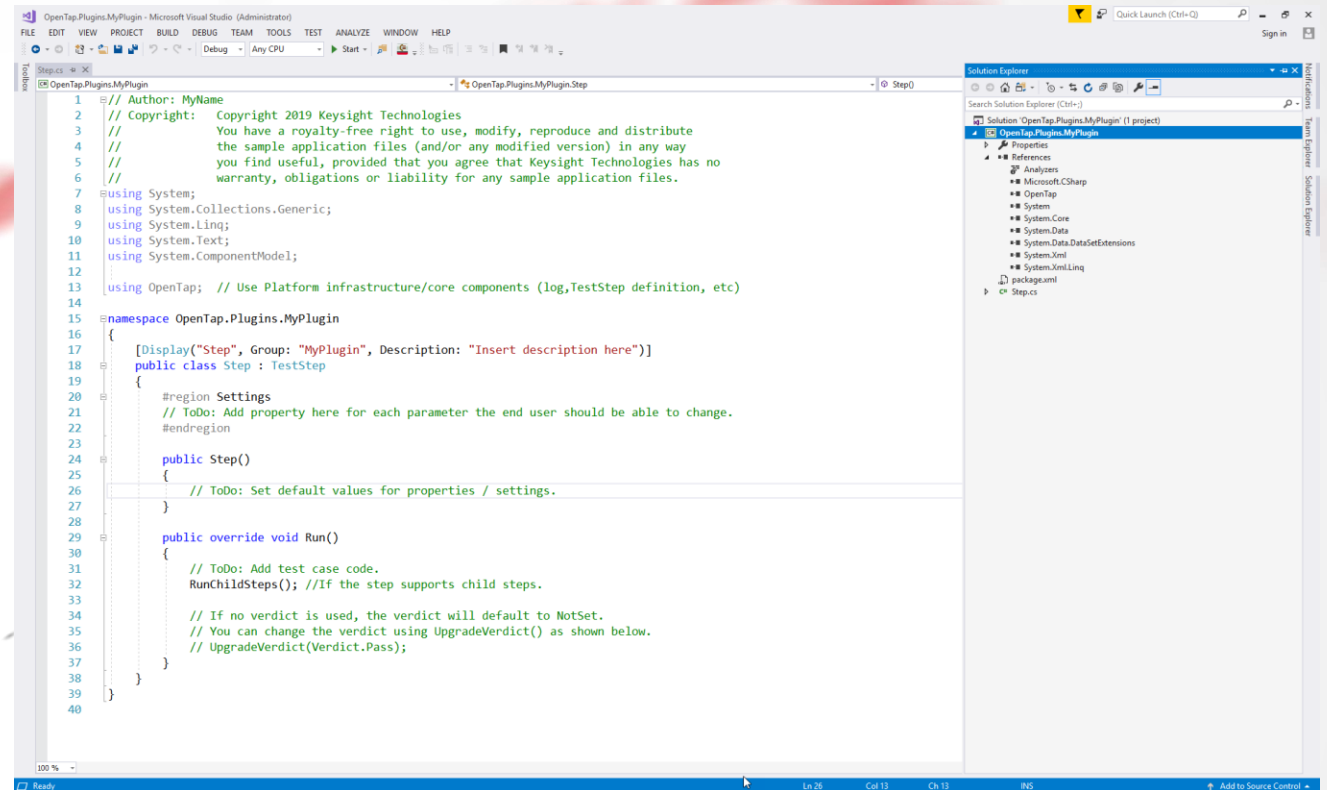
Log Panel

View messages related to test plan execution

Resource Bar

Configure Instruments, DUTs and Result Listeners

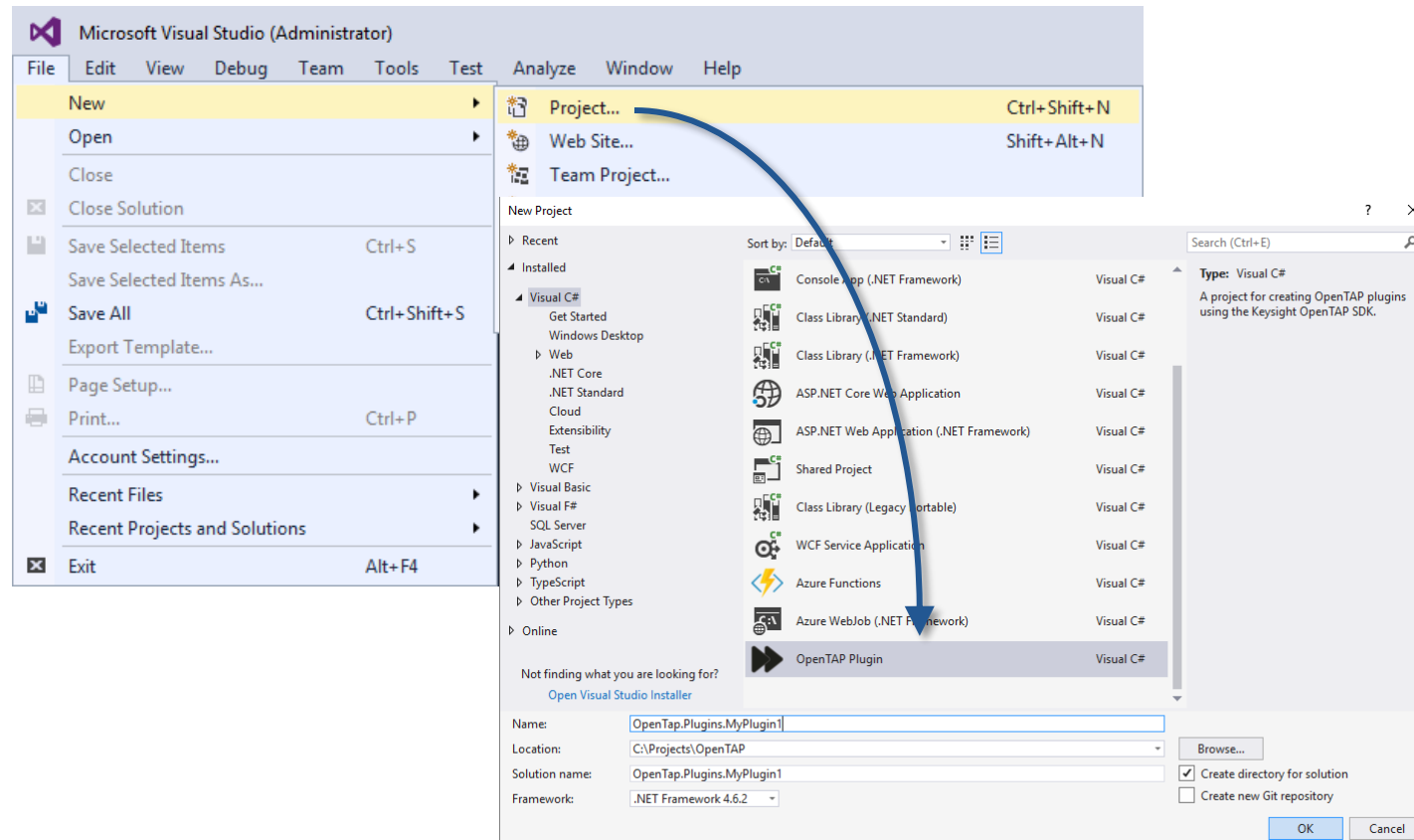




Getting Started in Visual Studio

Getting Started

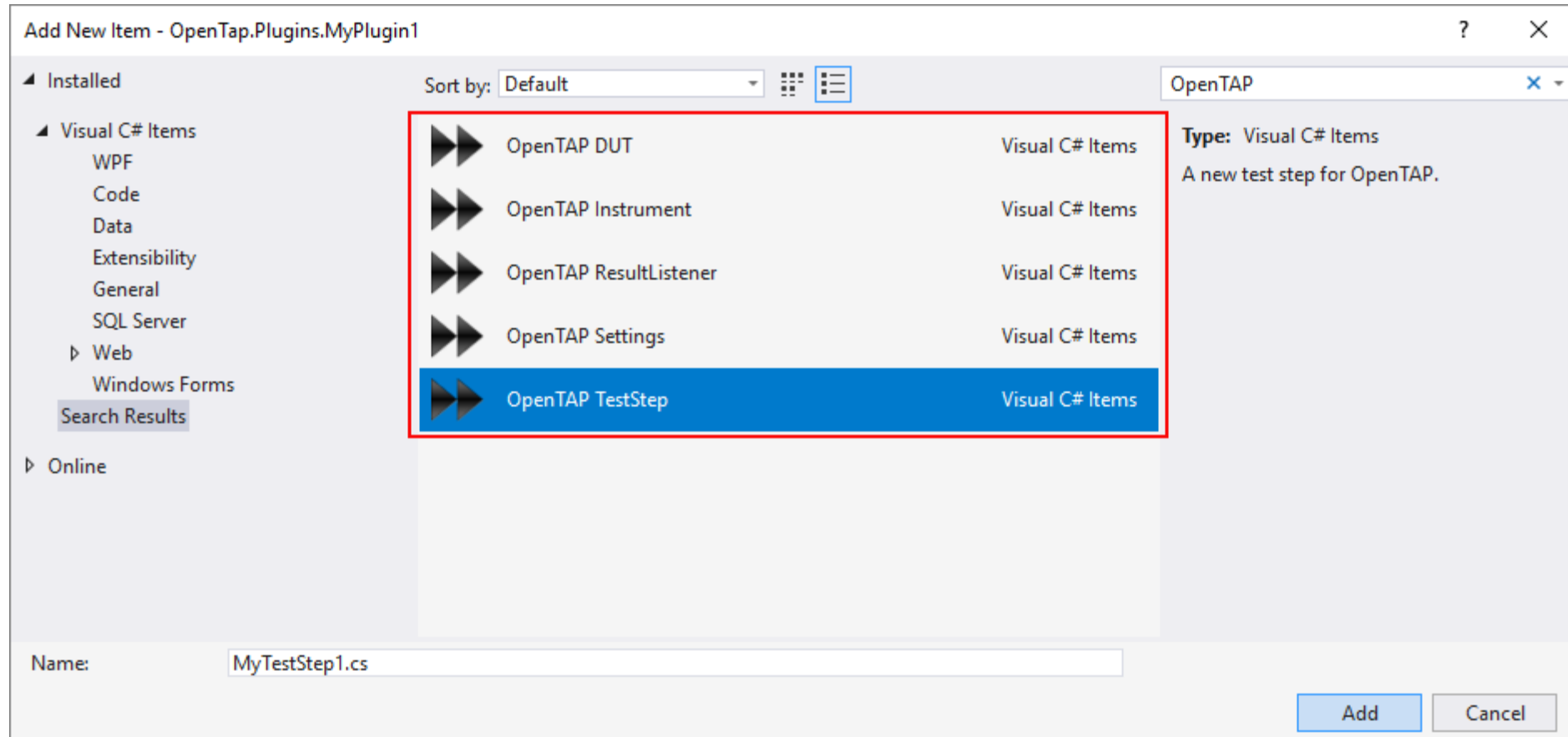
OPENTAP PROJECT TEMPLATE FOR VISUAL STUDIO



Getting Started

OPENTAP CLASS TEMPLATES FOR VISUAL STUDIO

- The OpenTAP SDK class templates provide skeleton implementation.



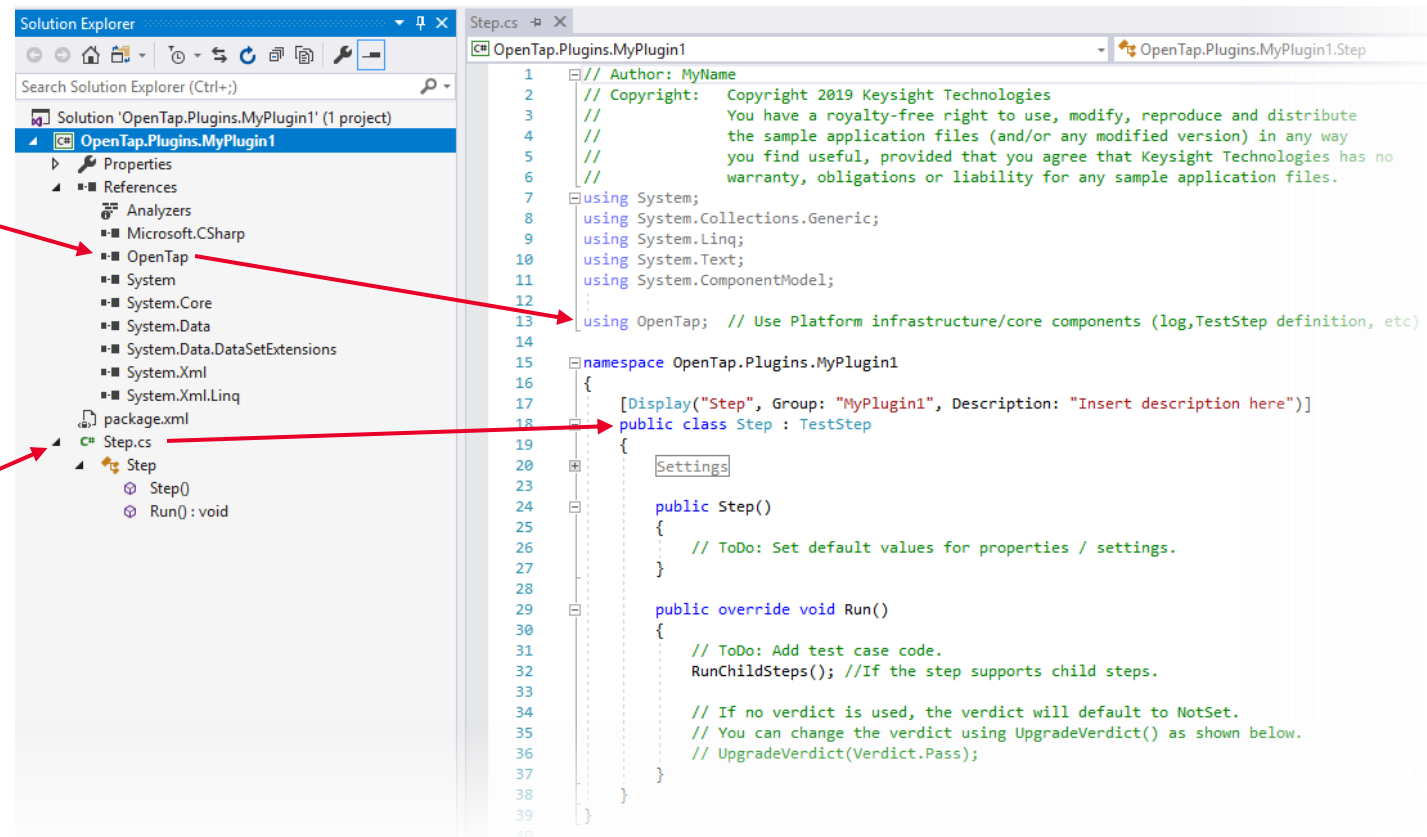
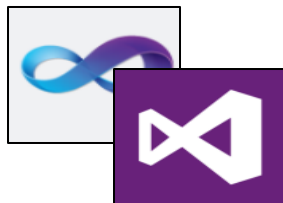
Getting Started

FIRST TEST STEPS

References to DLLs
OpenTAP.dll



Template TestStep
Skeleton TestStep is included in new project

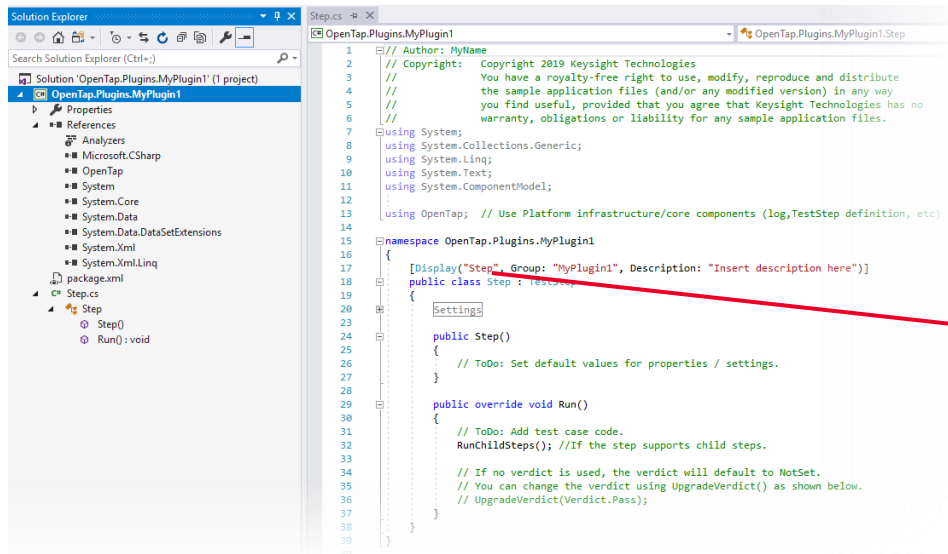


Getting Started

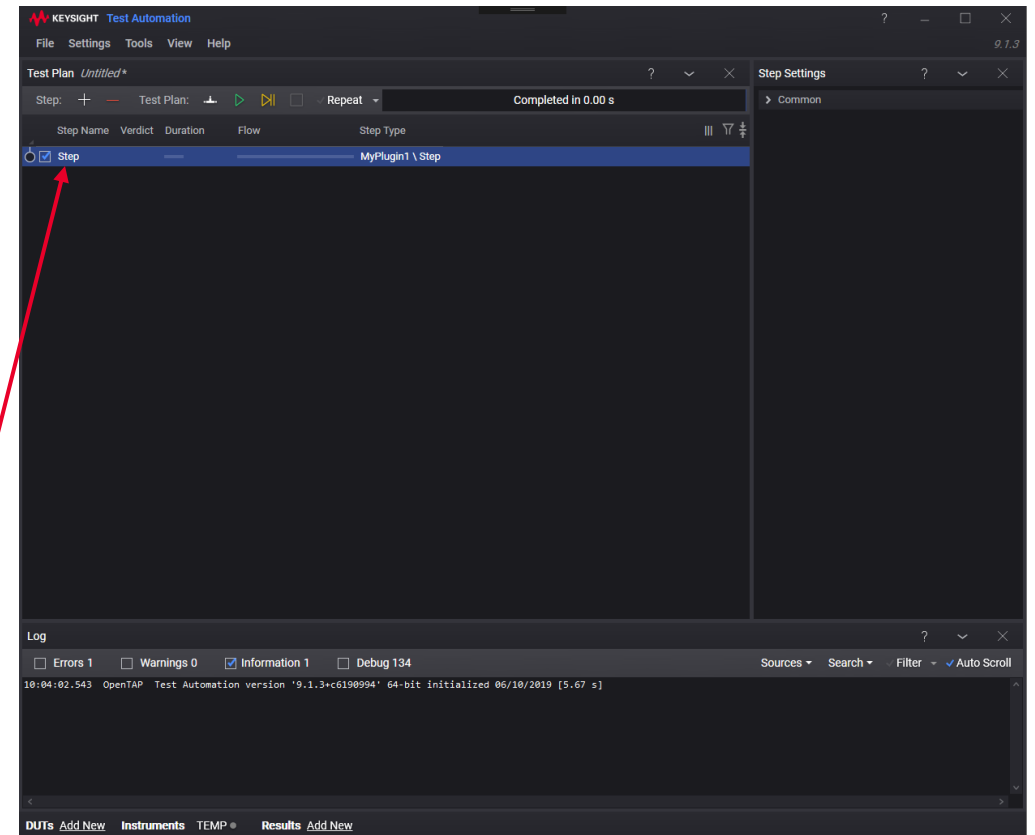
LOADING THE EDITOR FROM VISUAL STUDIO

Press **F5** (or click **Start**) in Visual Studio to:

1. Compile the code into a plugin DLL containing your TestStep
2. Move the DLL to the %TAP_PATH% location (defined by installer)
3. Start the Editor, where you can use your new plugin



Plugin DLL



Getting Started

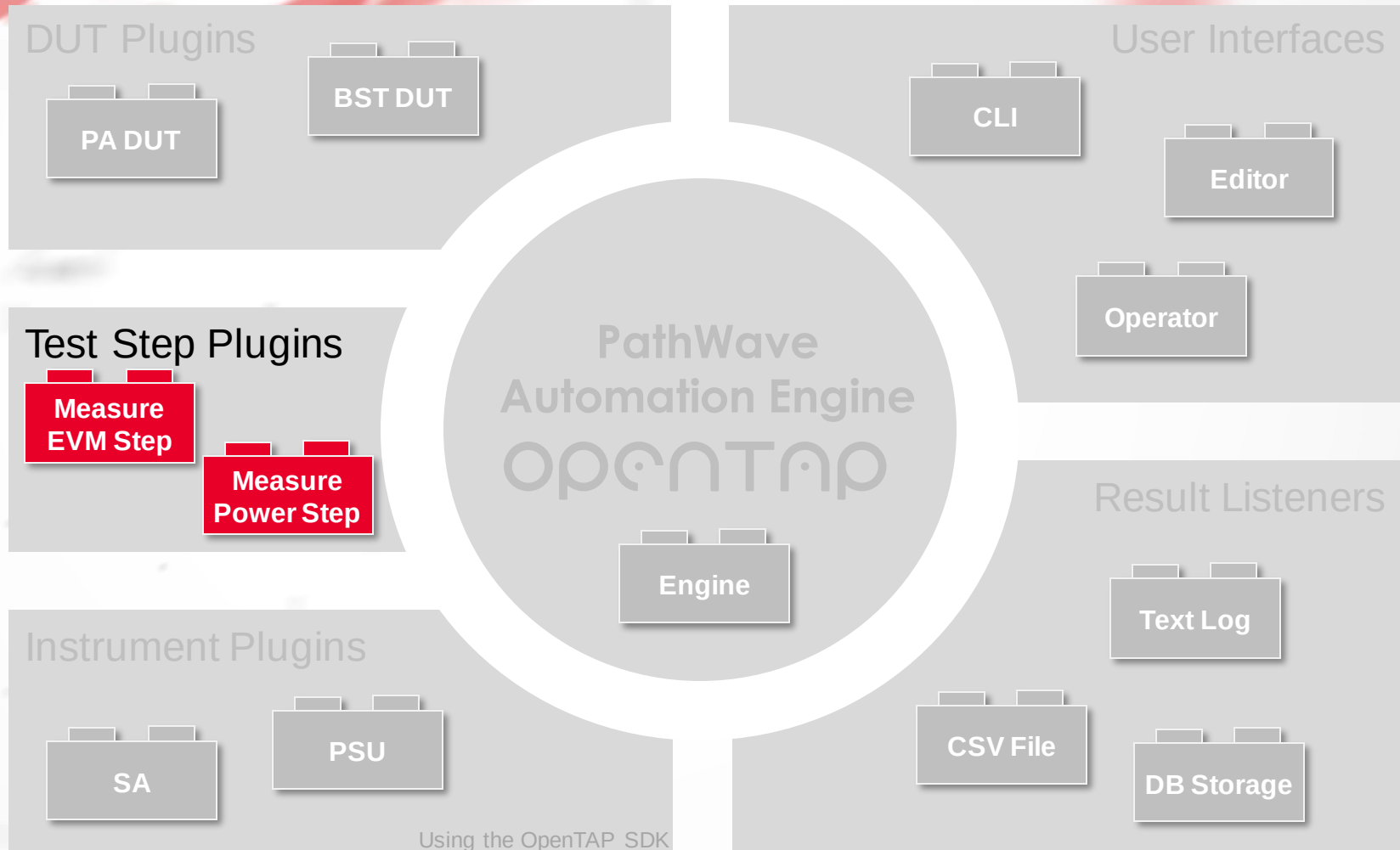
VISUAL STUDIO INTERACTION

Set the Editor to activate links in the Log that take you directly to the issue in the code.

The screenshot shows the Visual Studio interface with several components highlighted by red boxes and arrows:

- Settings Dialog:** The **Editor** tab is selected. Under the **Debug** section, the following options are checked:
 - Auto Attach Debugger
 - Show Source Code Links
- Log Panel:** The **Debug 16** window is active, showing a log of test execution. A red box highlights a file path in the log: `C:\Users\billbush\Source\Repos\Core30Nov\Sdk\Examples\...`.
- Code Editor:** The file `ThrowsAbortException.cs` is open. A red box highlights a line of code: `throw new TestPlan.AbortException("I forced abort", true);`. An arrow points from the file path in the log to this line of code.

Test Steps



Test Steps

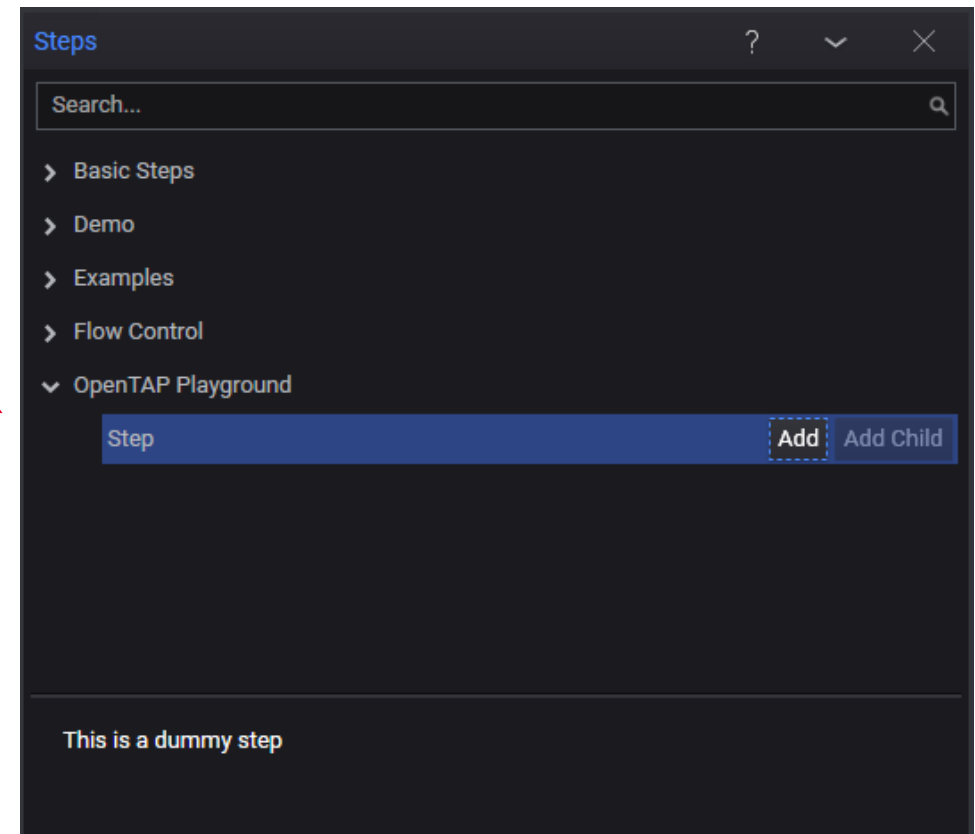
SDK SKELETON

- A TestStep is created by inheriting from **OpenTap.TestStep** or implementing **OpenTap.ITestStep**.
- TestStep skeleton:

```
[Display("Step", Group: "OpenTap Playground", Description: "This is a dummy step.")]
public class Step : TestStep
{
    #region Settings
    // TODO: Add property here for each parameter the end user can change
    #endregion

    public Step()
    {
        // TODO: Set default values here
    }

    public override void Run()
    {
        // TODO: Add test case code here
    }
}
```



Test Steps

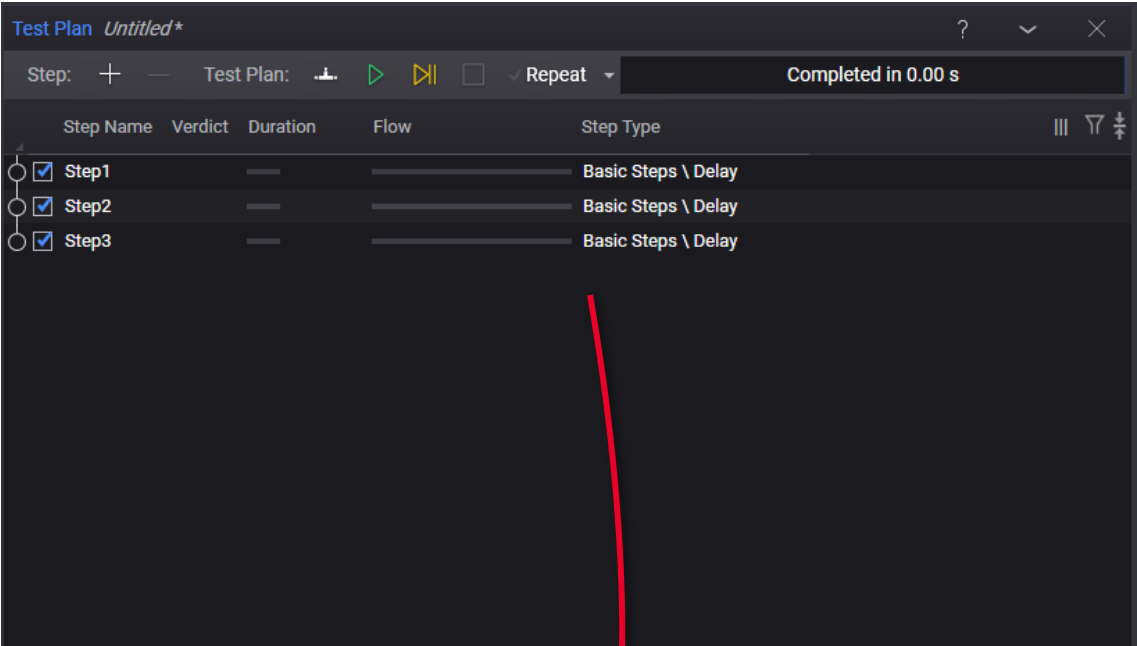
METHODS

- **PrePlanRun:** Optionally add any setup code. This method executes before the TestPlan starts.
- **Run:** Add test case code here. Typical actions:
 - Set up DUT
 - Set up Instrument
 - Perform Measurements
 - Publish Results
 - Determine and Set Verdict
- **PostPlanRun:** Optionally add any cleanup code. This method executes after the TestPlan has finished.

Test Steps

EXECUTION

Example
Testplan



Note reverse order:

TestPlan
Execution
Order

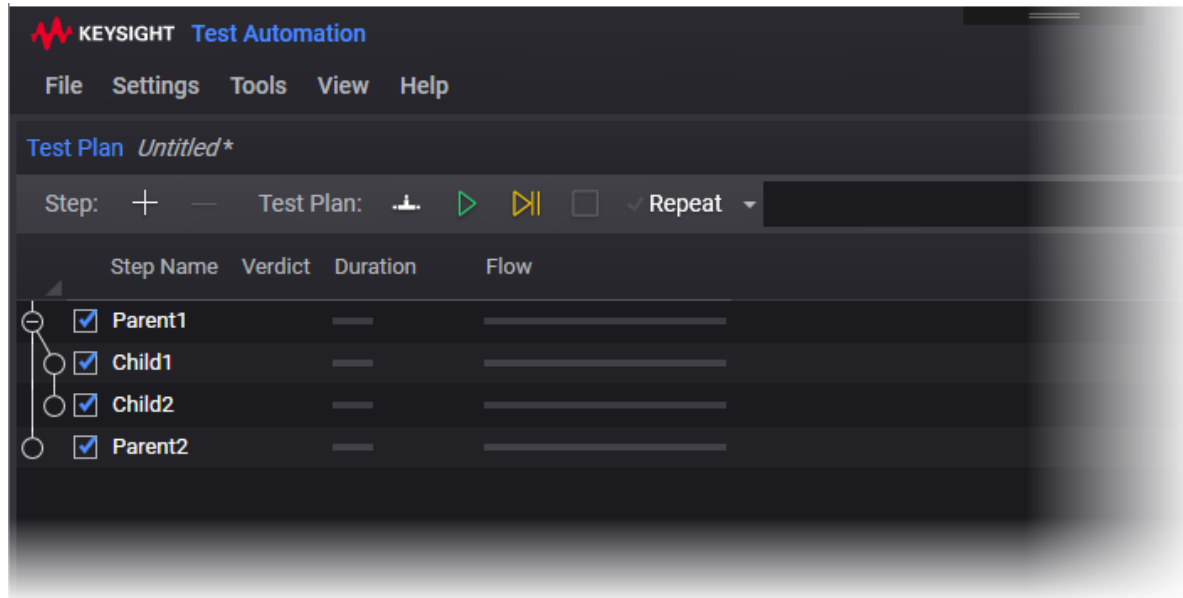
Execution Order	Purpose
Call PrePlanRun for every step (1,2,3)	Setup/Initialization
Call Run for every step (1,2,3)	Run (logic)
Call PostPlanRun for every step (3,2,1)	Cleanup

Test Steps

EXECUTION: HIERARCHIES AND CHILD STEPS

- A Test plan contains a list of test steps. Some test steps can have children.

Test Steps with Child Steps



Test Plan Execution Order

Order	Step	Method
1	Parent1	PrePlanRun()
2	Child1	PrePlanRun()
3	Child2	PrePlanRun()
4	Parent2	PrePlanRun()
5	Parent1	Run()
6	Child1	Run()
7	Child2	Run()
8	Parent2	Run()
9	Parent2	PostPlanRun()
10	Child2	PostPlanRun()
11	Child1	PostPlanRun()
12	Parent1	PostPlanRun()

Test Steps

HIERARCHIES AND CHILD STEPS

- A child TestStep is an ordinary step that is allowed to be inserted as a child of another step. Three attributes enable this:
 - `[AllowAsChildIn()]`
Specifies the type of TestStep that can be a parent to this step.
Add multiples of this attribute to allow the step to be a child of multiple types of parent TestSteps.
 - `[AllowAnyChild]`
Specifies that this step can be a parent to any ordinary TestStep.
 - `[AllowChildrenOfType()]`
Specifies that this step can be a parent only of TestSteps of a certain type or that implement a certain interface.
- Child TestSteps can access their parent TestStep by using the `TestStep.Parent` property or the `TestStep.GetParent<>()` method.

```
[Display("MyChildTestStep1", Group: "Demo")]  
[AllowAsChildIn(typeof(Step))]  
public class MyChildTestStep1 : TestStep  
{  
    public override void Run()  
    {  
        Step = parent = (Step)this.Parent;  
        int channel = parent.TxChannel;  
        // ...  
    }  
}
```

Test Steps

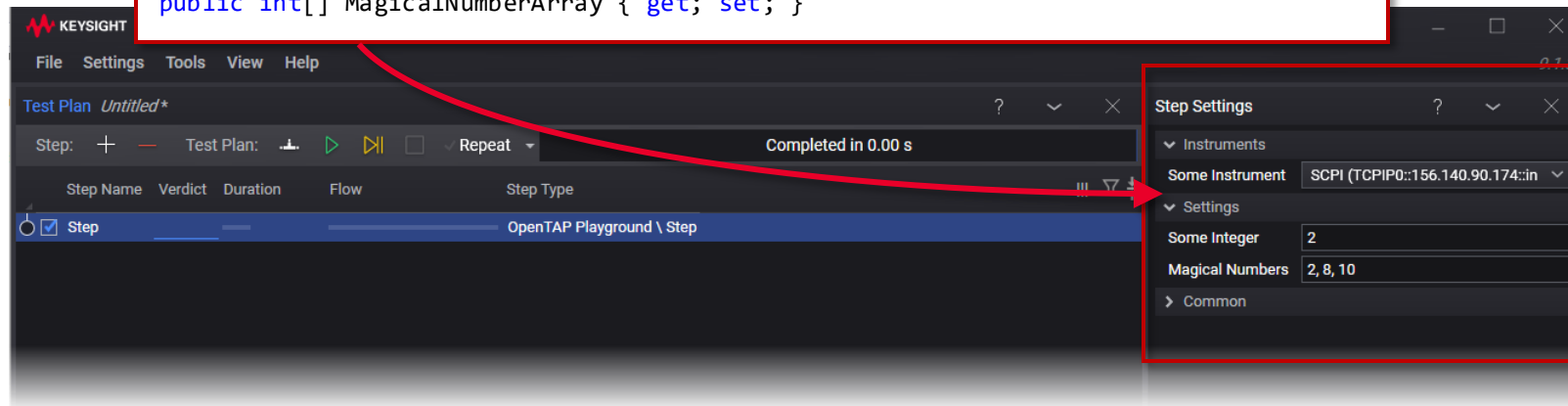
SETTINGS

- Public properties are automatically reflected in Editor
- Settings are automatically loaded and saved when the TestPlan is loaded/saved
- Settings can be grouped and ordered

```
[Display("Step", Group: "OpenTAP Playground", Description: "This is a dummy step.")]
public class Step : TestStep
{
    [Display("Some Instrument", Group: "Instruments", Description: "Dummy instrument.")]
    public Instrument SomeInstrument { get; set; }

    [Display("Some Integer", Group: "Settings", Order: 2)]
    public int SomeInteger { get; set; }

    [Display("Magical Numbers", Group: "Settings", Order: 3)]
    public int[] MagicalNumberArray { get; set; }
}
```



Test Steps

VERDICT

Values

- **NotSet** (0): No verdict was set
- **Pass** (10): Test passed
- **Inconclusive** (20): Test had an inconclusive result.
- **Fail** (30): Test failed
- **Aborted** (40): Test was aborted
- **Error** (50): Test failed due to an exception or other procedural error (for example, no instrument/DUT connection).

Logic

- Each *TestStep* has a verdict
- *UpgradeVerdict* takes the max of new value and current value
- *TestPlan* verdict is the max of all *TestStep* verdicts

Example of Setting Verdict

```
if(value < minLimit || value > maxLimit)
    UpgradeVerdict(Verdict.Fail);
else
    UpgradeVerdict(Verdict.Pass);
```

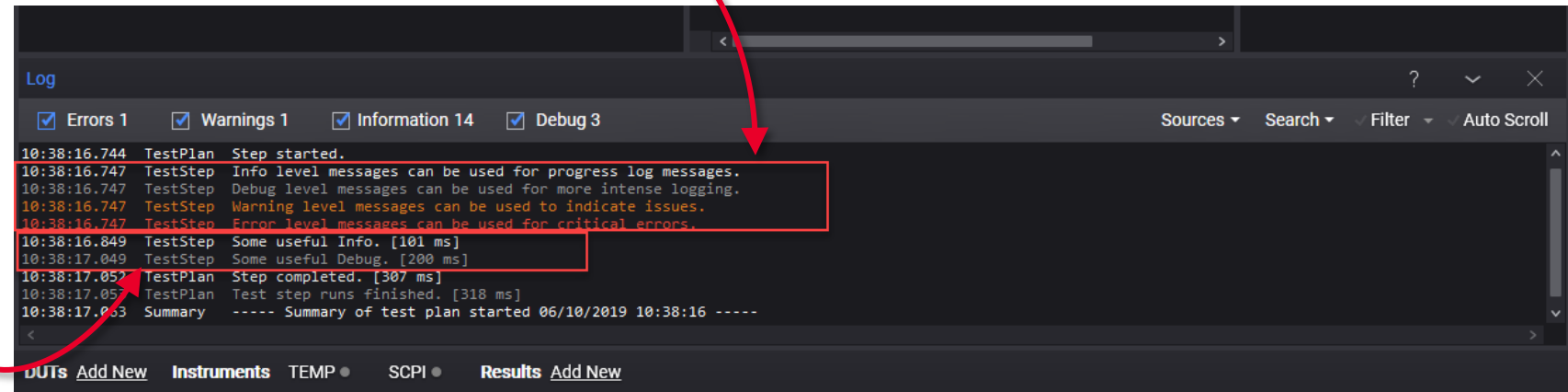
Test Steps

LOGGING

- OpenTAP provides logging for severity and timing

```
Log.Info("Info level messages can be used for progress log messages.");  
Log.Debug("Debug level messages can be used for more intense logging.");  
Log.Warning("Warning level messages can be used to indicate issues.");  
Log.Error("Error level messages can be used for critical errors.");
```

```
Stopwatch sw = Stopwatch.StartNew();  
TapThread.Sleep(100);  
Log.Info(sw, "Some useful Info.");  
  
sw = Stopwatch.StartNew();  
TapThread.Sleep(200);  
Log.Debug(sw, "Some useful Debug.");
```



Test Steps

TRACEBAR

- Example that shows a simple value versus limits, with limit checking

```
// Tracebar can be used to show results in the MyLog.
var traceBar = new TraceBar();
traceBar.LowerLimit = -3.0;
for (var i = -2; i < 11; i++)
{
    traceBar.UpperLimit = i < 5 ? 3 : 15;
    // GetBar returns a string with value, low limit, a dashed line
    // indicating magnitude, the upper limit, and (if failing), a fail indicator.
    string temp = traceBar.GetBar(i);
    Log.Info("MyResult: " + temp);
    TapThread.Sleep(200);
}
// Sample output shown below.
// MyResult: 2.00 - 3-----|----- 3
// MyResult: 3.00 - 3-----| 3
// MyResult: 4.00 - 3-----> 3 Fail
// MyResult: 5.00 - 3----- -| -----15
// MyResult: 6.00 - 3----- -| -----15

// TraceBar remembers if any results failed, so it can be used for the verdict.
UpgradeVerdict(traceBar.CombinedVerdict);
```

Log

☒ Errors 0 ☒ Warnings 0 ☒ Information 25 ☒ Debug 1

Time	Category	Message	Value	Limit	Verdict
10:47:28.450	TestPlan	Step started.			
10:47:28.456	TestStep	MyResult:	-2	-3	3
10:47:28.657	TestStep	MyResult:	-1	-3	3
10:47:28.857	TestStep	MyResult:	0	-3	3
10:47:29.057	TestStep	MyResult:	1	-3	3
10:47:29.258	TestStep	MyResult:	2	-3	3
10:47:29.457	TestStep	MyResult:	3	-3	3
10:47:29.657	TestStep	MyResult:	4	-3	3 Fail
10:47:29.858	TestStep	MyResult:	5	-3	15
10:47:30.058	TestStep	MyResult:	6	-3	15
10:47:30.258	TestStep	MyResult:	7	-3	15
10:47:30.458	TestStep	MyResult:	8	-3	15
10:47:30.658	TestStep	MyResult:	9	-3	15
10:47:30.858	TestStep	MyResult:	10	-3	15
10:47:31.061	TestPlan	Step completed with verdict 'Fail'. [2.61 s]			
10:47:31.062	TestPlan	Test step runs finished. [2.62 s]			
10:47:31.072	Summary	Summary of test plan started 06/10/2019 10:47:28			
10:47:31.074	Summary	Step	2.61 s		Fail
10:47:31.075	Summary	Test plan completed with verdict Fail in 2.86 s			
10:47:31.075	Summary				
10:47:31.207	TEMP	Device TEMP closed			
10:47:31.207	TEMP	Resource "TEMP" closed. [231 us]			

DUTs Add New Instruments TEMP SCPI Results Add New

Test Steps

SETTINGS VALIDATION

```
[Display("RuleValidation Example", Groups: new[] { "Examples", "Plugin Development", "Step Settings" },
    Description: "An example of how validation works.")]
// Validation works for ComponentSettings, Resources (DUTs, Instruments, and ResultListeners) and Test Steps,
// since they all extend ValidatingObject.
public class RuleValidation : TestStep
{
    [Display("Should Be True Property", Description: "This value should be true to pass validation.")]
    public bool ShouldBeTrueProp { get; set; }

    public int MyInt1 { get; set; }
    public int MyInt2 { get; set; }

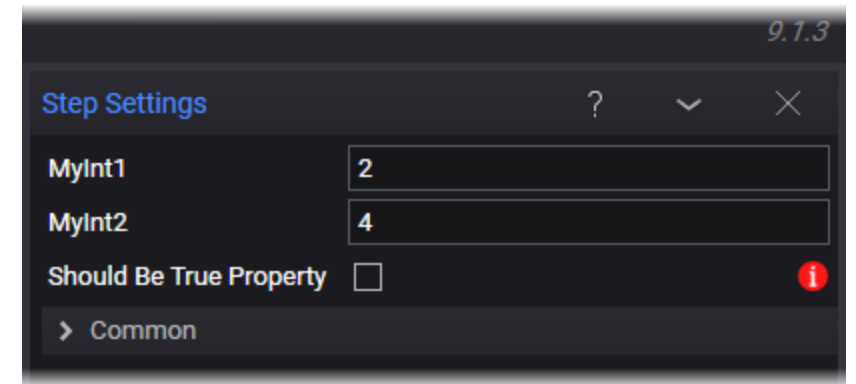
    public RuleValidation()
    {
        // Validation occurs during the constructor. When using the GUI, validation will occur upon
        // editing. When using the engine without the GUI, validation occurs upon loading the test plan.

        // Calls a function that returns a boolean.
        Rules.Add(CheckShouldBeTrueFunc, "Must be true to run", "ShouldBeTrueProp");

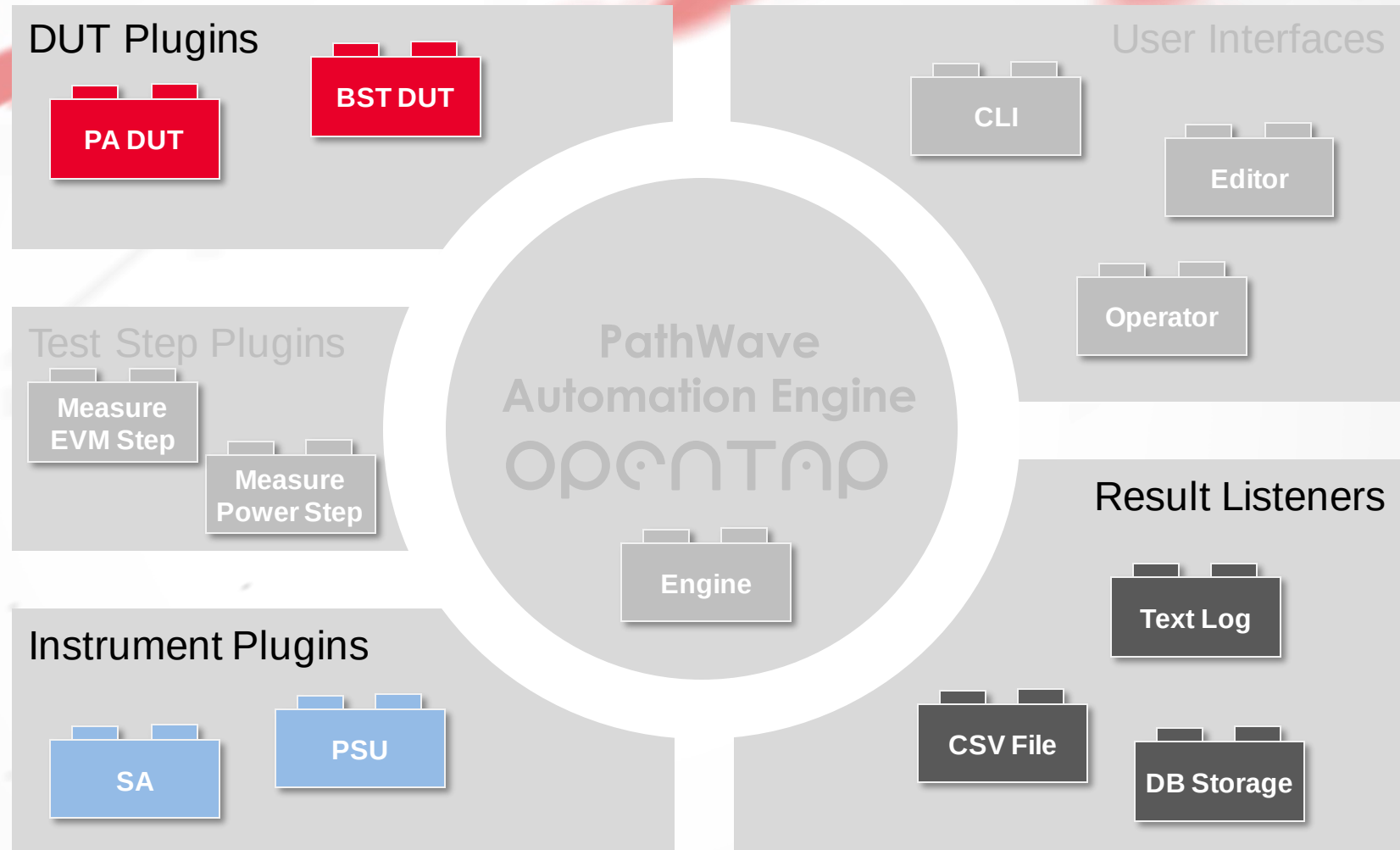
        // Calls an anonymous function that returns a boolean.
        Rules.Add(() => MyInt1 + MyInt2 == 6, "MyInt1 + MyInt2 must == 6", "MyInt1", "MyInt2");

        // Ensure all rules fail.
        ShouldBeTrueProp = false;
        MyInt1 = 2;
        MyInt2 = 2;
    }

    private bool CheckShouldBeTrueFunc()
    {
        return ShouldBeTrueProp;
    }
}
```



Resources

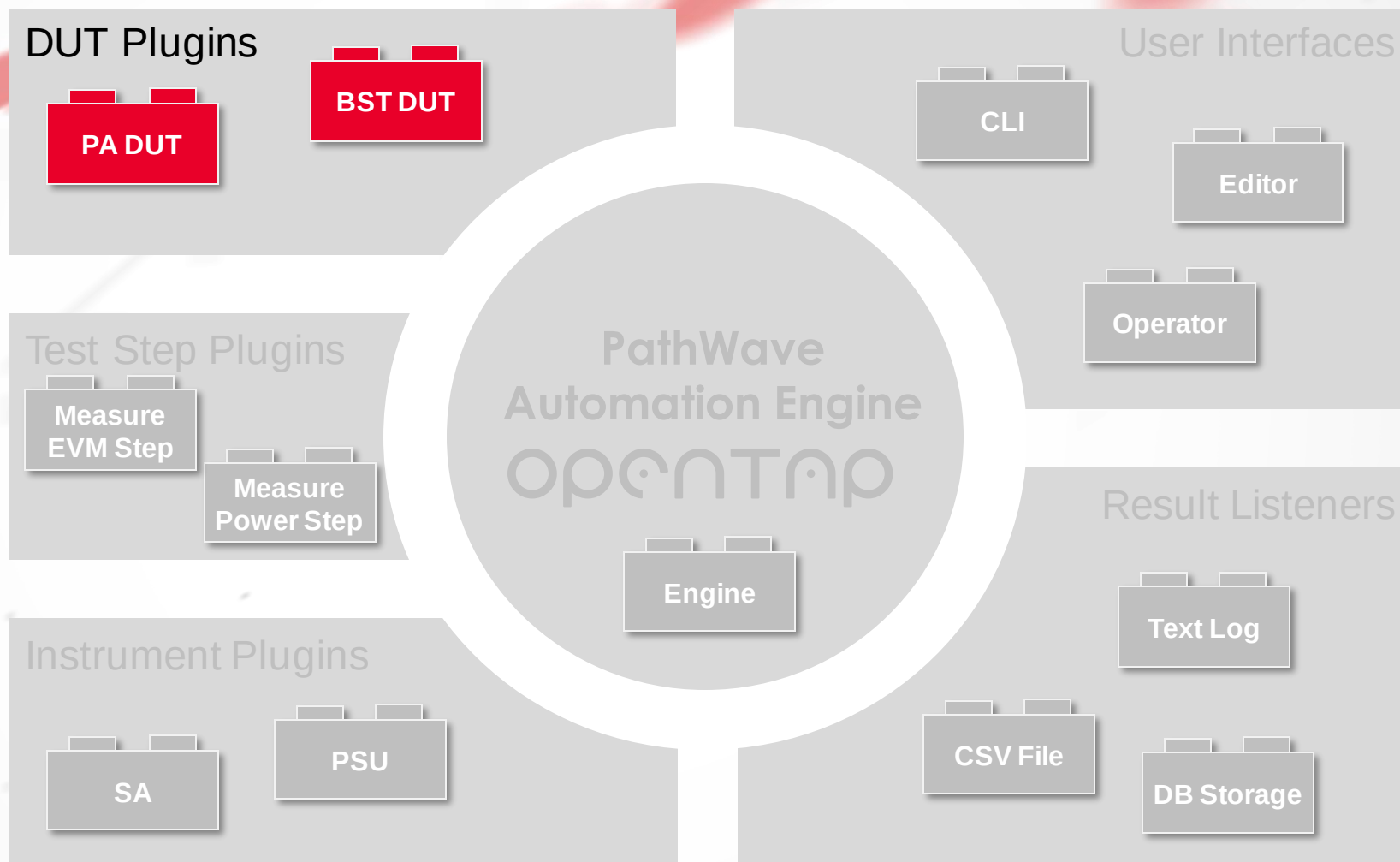


Resources

ELEMENTS COMMON TO RESOURCES

- **Open:** Called before the TestPlan execution starts
- **Closed:** Called when the TestPlan execution ends
- **IsConnected:** Indicates of connection status (Boolean)
- **OnActivity:** Raises activity event for use by GUI
- **Log:** Writes progress, debug, error and timing messages

DUTs



DUT

OVERVIEW

- A DUT is created by either:
 - Inheriting from `OpenTap.Dut` or
 - Implementing `OpenTap.IDut`

```
[Display("ExampleDut", Group: "MyPlugin1",  
    Description: "Example DUT.")]  
[ShortName("MyDUT")]  
public class ExampleDut : Dut  
{  
    // Initializes a new instance of this DUT class.  
    public ExampleDut()  
    {  
        // ToDo: Set default values for properties / settings.  
    }  
  
    // Opens a connection to the DUT  
    public override void Open()  
    {  
        base.Open(); // Sets IsConnected = true;  
    }  
  
    // Closes the connection to the DUT  
    public override void Close()  
    {  
        base.Close(); // Sets IsConnected = false;  
    }  
}
```

`Open()` is called before the
TestPlan execution starts

`Close()` is called when the
TestPlan execution ends

DUT

SETTINGS

- Properties on the DUT class becomes settings the end user can change.

```
[Display("FourPortDut",  
    Groups: new[] { "Examples", "Plugin Development"},  
    Description: "A four port DUT. ")]  
public class FourPortDut : Dut  
{  
    [Display("MyString", Group: "DutExtensions")]  
    public string MyString { get; set; }  
  
    [Display("MyDouble", Group: "DutExtensions")]  
    public double MyDouble { get; set; }  
}
```

Add New DUT

Search...

- Examples
 - Example Plugin
- Plugin Development
 - Connections
 - Four Port DUT** Add

Close

Bench Settings

Profile: Default

Connections | **DUTs** | Example Component Settings | Instruments

FourPortDut

Four Port DUT

- ▼ DutExtensions
 - MyDouble: 0
 - MyString: A String
- ▼ Common
 - ID: ID123456
 - Comment: Some Comment

+
-

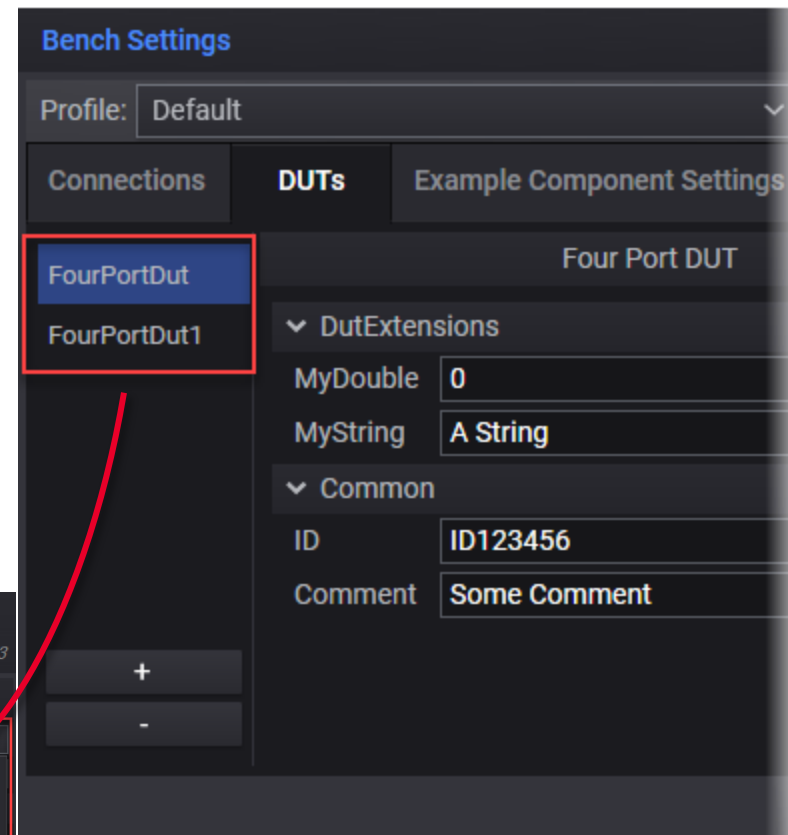
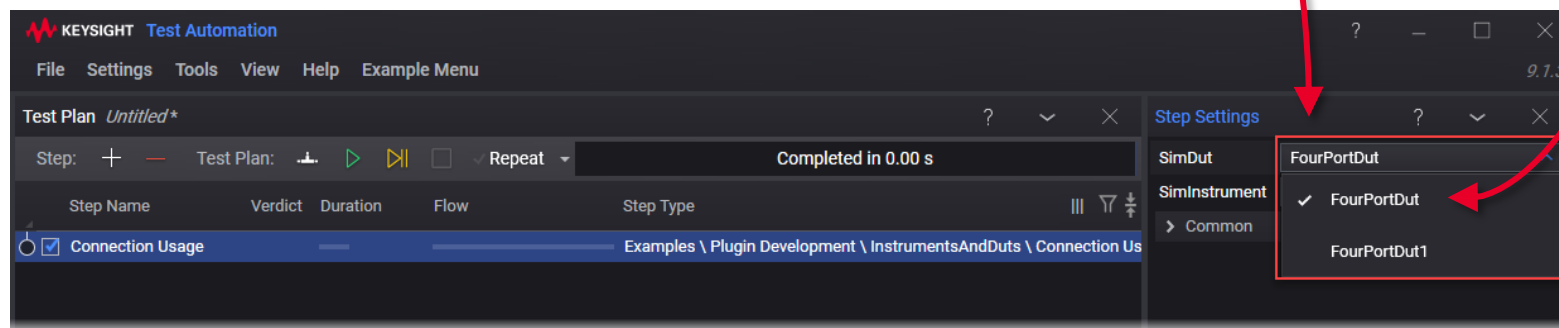
OK Cancel

DUT

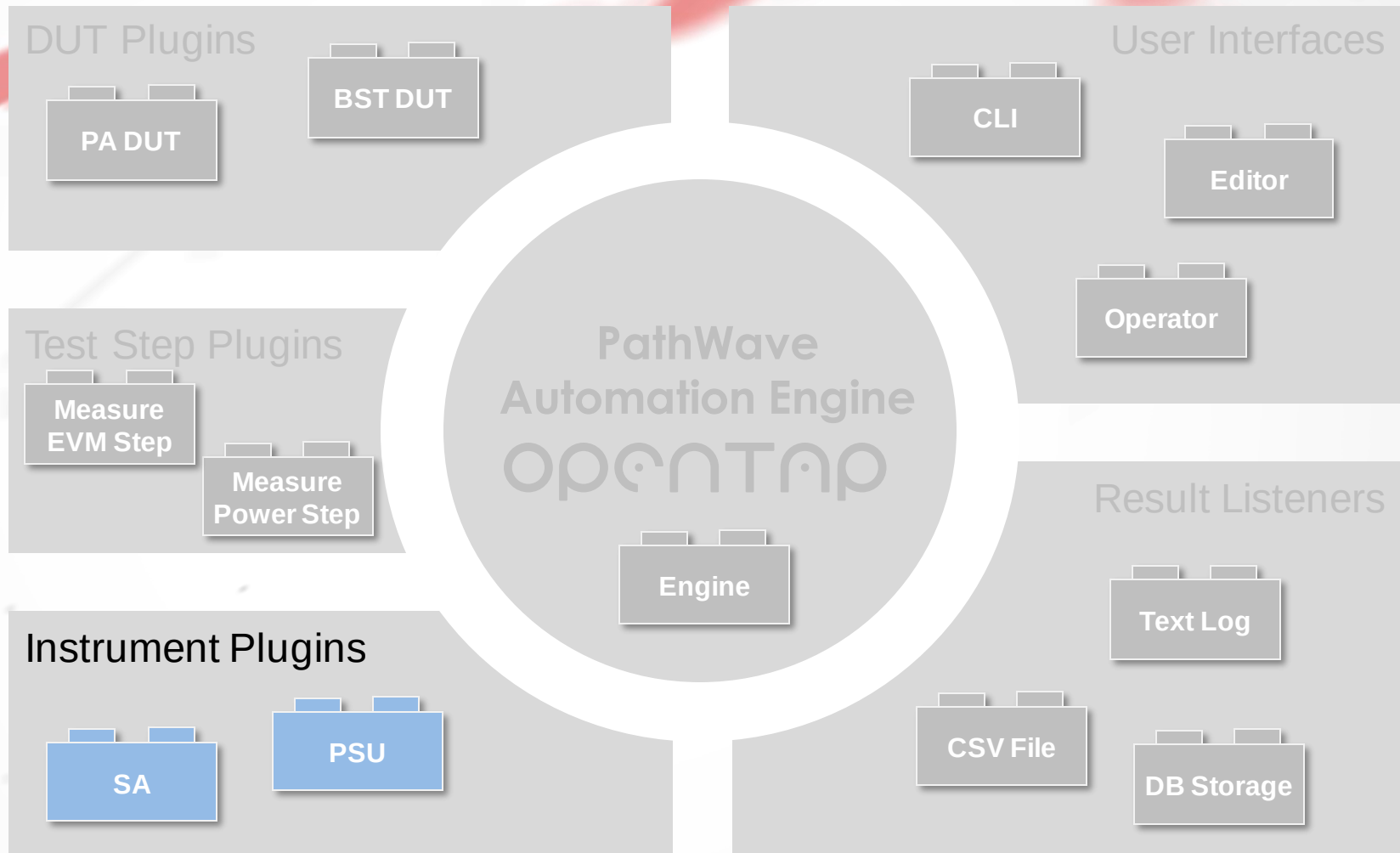
USAGE IN TESTSTEP

- Add the DUT to the TestStep class to use the DUT in the step as well as automatic config when the TestStep is loaded from xml

```
[Display("TestStepUsingDut", Group: "PluginDevelopment")]  
public class TestStepUsingDut : TestStep  
{  
    #region Settings  
    [Display("DUT")]  
    public FourPortDut Dut { get; set; }  
    #endregion  
}
```



Instruments



Instruments

OVERVIEW

- An **Instrument** is created by either:
 - Inheriting from `OpenTap.Instrument`
 - Implementing `OpenTap.IInstrument`

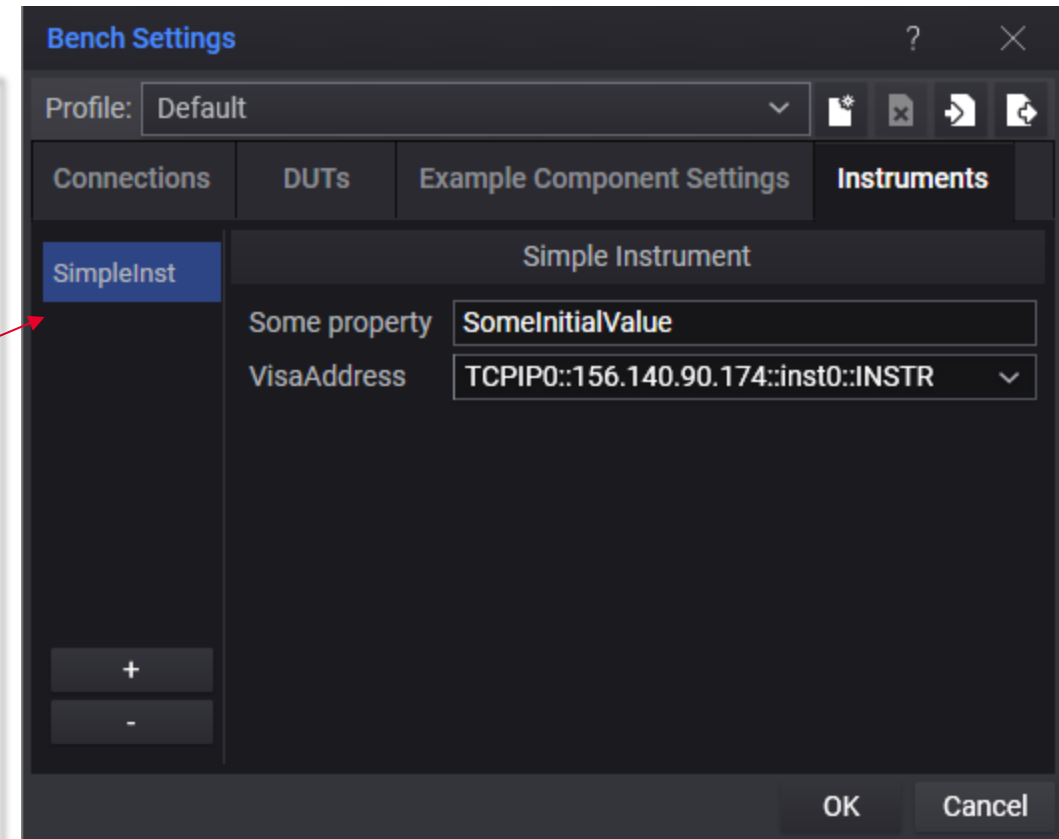
```
[Display("Simple Instrument",
  Groups: new[] { "Examples", "Plugin Development" })]
public class SimpleInstrument : Instrument
{
    [VisaAddress]
    public string VisaAddress { get; set; }

    [Display("Some property")]
    public string SomeProperty { get; set; }

    public SimpleInstrument()
    {
        Name = "SimpleInst";
    }

    public override void Open()
    {
        base.Open();
    }

    public override void Close()
    {
        base.Close();
    }
}
```



Instruments

SCPI INSTRUMENTS

- A **SCPI-based Instrument** (adds SCPI functionality using VISA from Keysight IO Libraries) is created by:
 - Inheriting from `OpenTap.ScpiInstrument`

```
[Display("Simple Scpi Instrument",
    Groups: new[] { "Examples", "Plugin Development" })]
public class SimpleScpiInstrument : ScpiInstrument
{
    [VisaAddress]
    public string VisaAddress { get; set; }

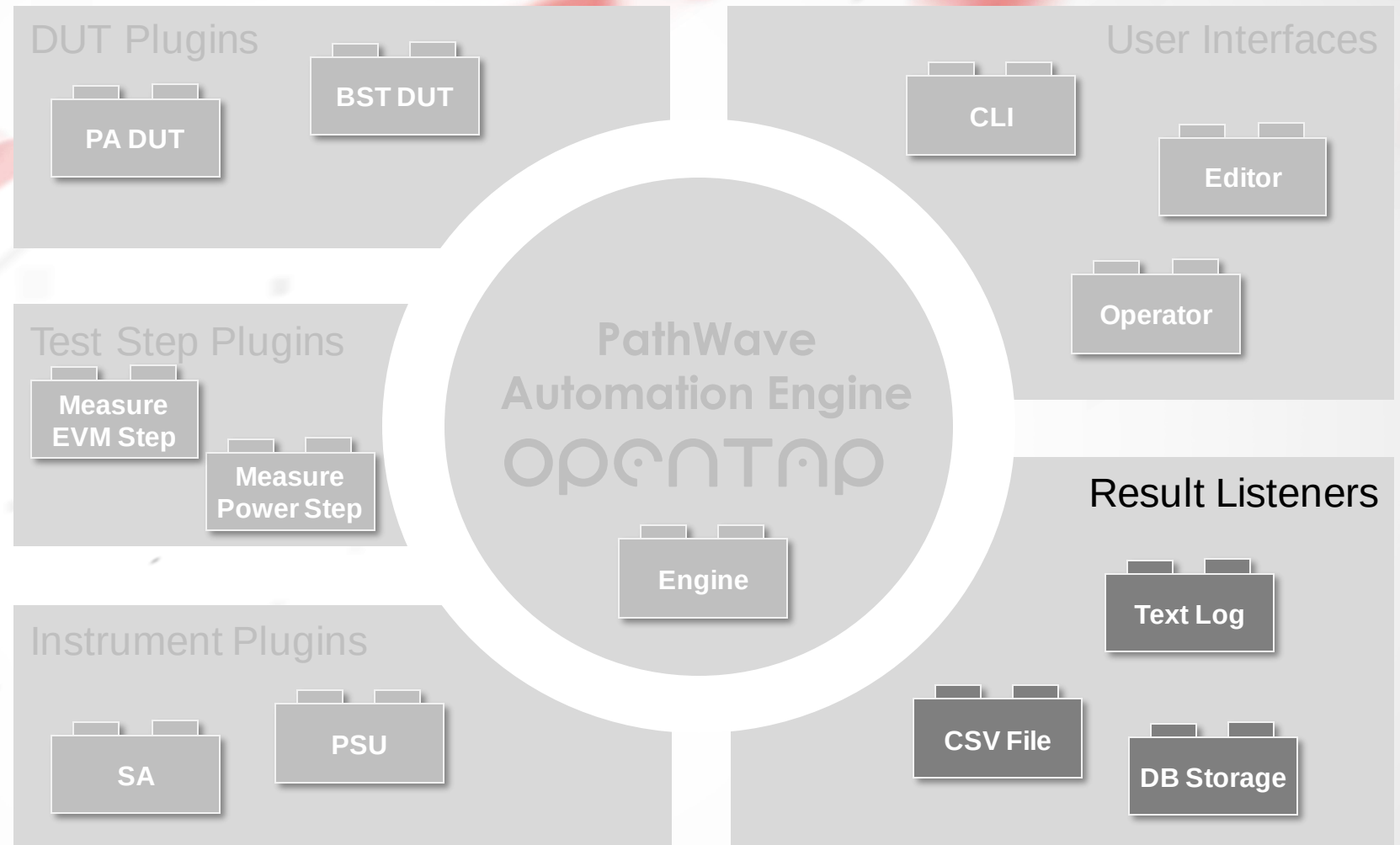
    [Display("Some property")]
    public string SomeProperty { get; set; }

    public SimpleScpiInstrument()
    {
        Name = "SimpleScpiInst";
    }

    public override void Open()
    {
        base.Open();
    }

    public override void Close()
    {
        base.Close();
    }
}
```

Results Listeners



Result Listeners

OVERVIEW

- A Result Listener plugin is created by either:
 - Inheriting from `OpenTap.ResultListener` or
 - Implementing `OpenTap.IResultListener`

```
[Display("MyResultListener1")]
public class MyResultListener : ResultListener
{
    // Add resource open code.
    public override void Open() { base.Open(); }

    // Add resource close code.
    public override void Close() { base.Close(); }

    // Add handling code for test plan run start.
    public override void OnTestPlanRunStart(TestPlanRun planRun) { }

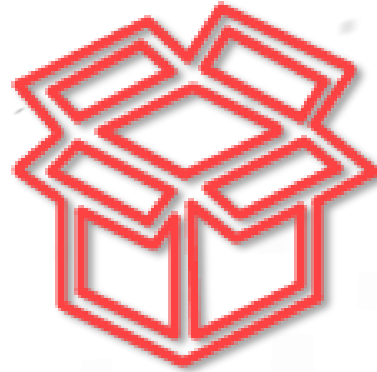
    // Add handling code for test step run start.
    public override void OnTestStepRunStart(TestStepRun stepRun) { }

    // Add handling code for result data.
    public override void OnResultPublished(Guid stepRun, ResultTable result)
    {
        OnActivity();
    }

    // Add handling code for test step run completed.
    public override void OnTestStepRunCompleted(TestStepRun stepRun) { }

    // Add handling for test plan run completed.
    public override void OnTestPlanRunCompleted(TestPlanRun planRun, Stream logStream) { }
}
```

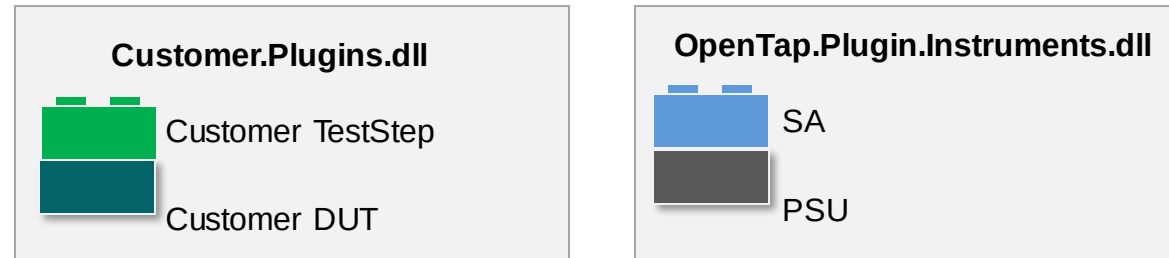
Plugin Packaging



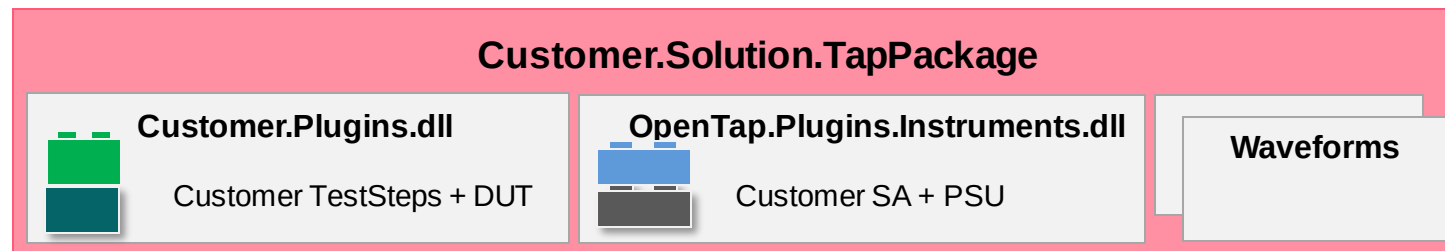
Plugin Packaging

OVERVIEW

- Each plugin is a .NET class. One .NET assembly (DLL) can contain several plugins.



- Plugins are distributed in packages (.TapPackage files). These packages can contain one or more DLLs and other required files (such as waveforms).



.TapPackage files can be opened like zip files when renamed to .zip.

Plugin Packaging

PACKAGE DEFINITION

- OpenTAP Packages are defined in Package.xml.
- *Package.xml is included in projects generated using SDK.*

Package Description

```
<?xml version="1.0" encoding="UTF-8"?>
<Package Name="Demonstration" xmlns="http://opentap.io/schemas/packages" Version="2.1.3" Configuration="$(GitBranch)">
  <Description>
    This is a demonstration plugin package.
    <Status>Released</Status>
    <Organization>Keysight Technologies</Organization>
    <Contacts>tap.support@keysight.com</Contacts>
    <Prerequisites>None</Prerequisites>
    <Hardware>Emulated PSU</Hardware>
    <Links>http://www.keysight.com/find/TAP</Links>
  </Description>
  <Files>
    <File Path="OpenTap.Plugins.Demo.Battery.dll"/>
    <File Path="OpenTap.Plugins.Demo.ResultsAndTiming.dll"/>
    <File Path="SampleLog.txt" />
    <File Path="DataGenForResultsViewer.TapPlan"/>
    <File Path="DataGenForTimingAnalysis.TapPlan"/>
  </Files>
</Package>
```

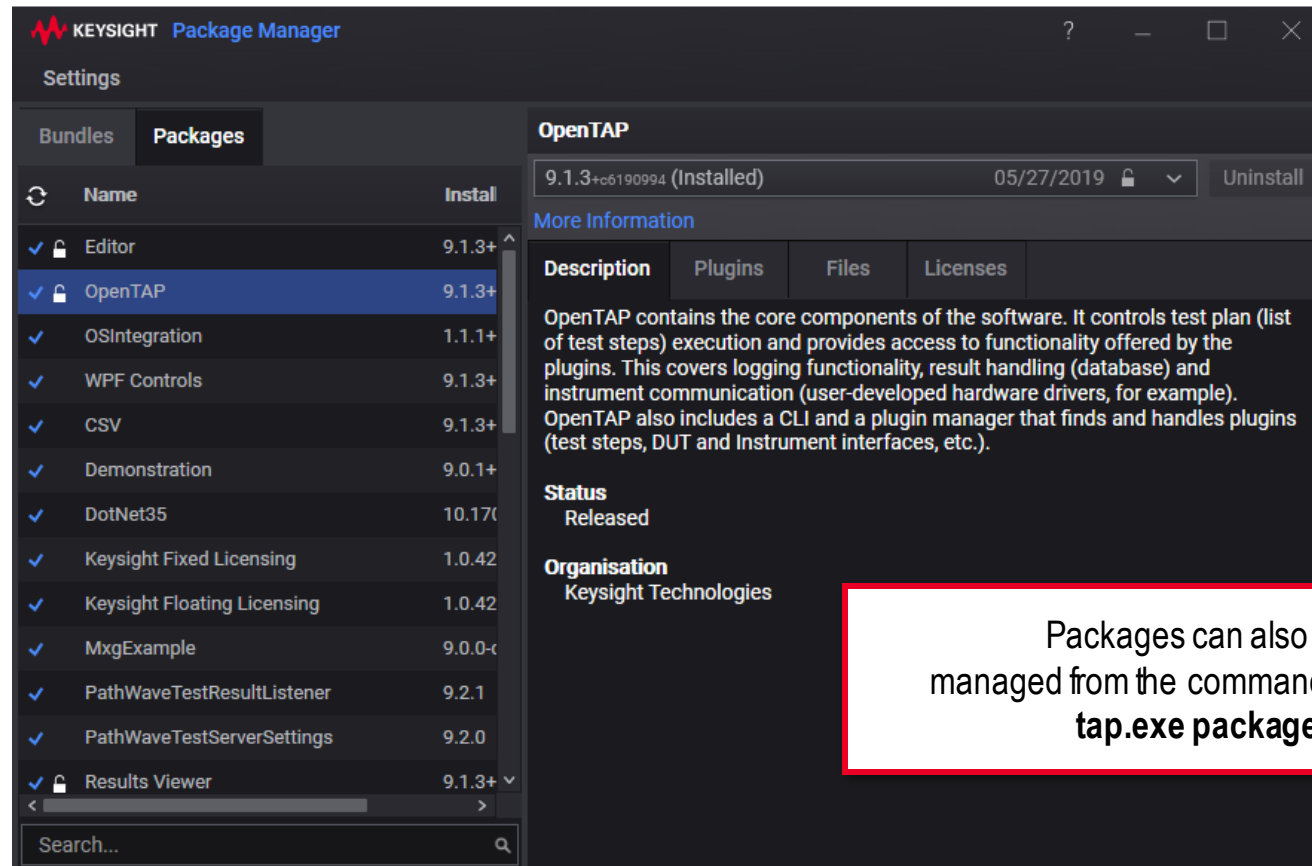
Package version

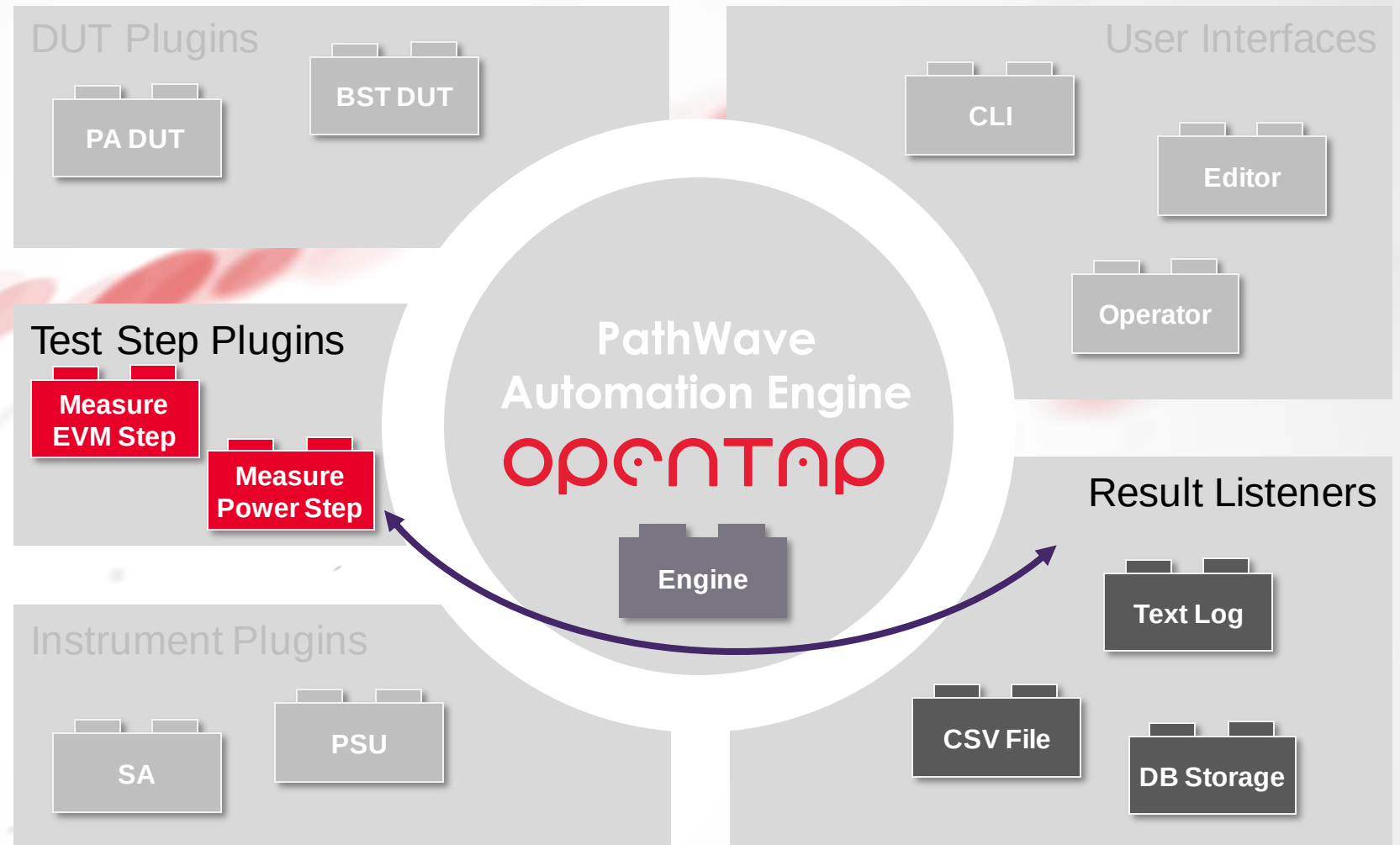
Included Files

Plugin Packaging

MANAGING PACKAGES

- Packages are installed and removed with Package Manager



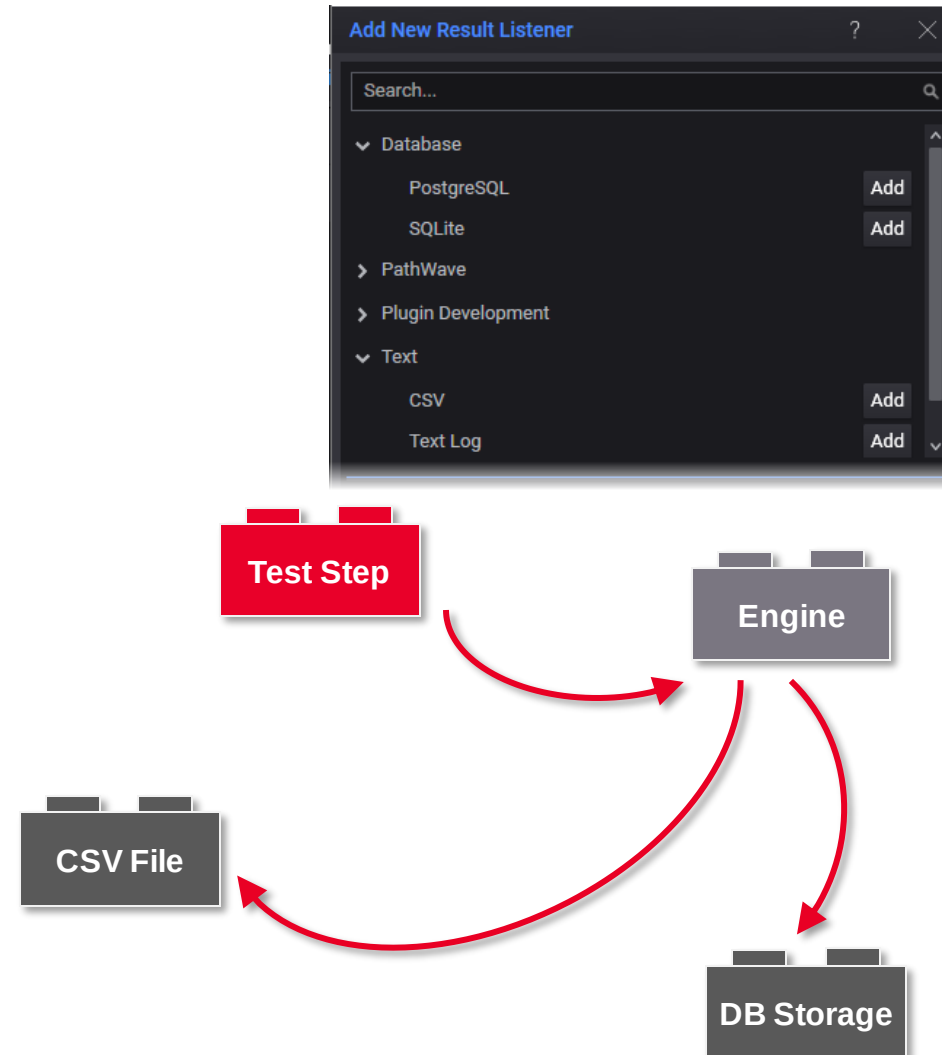


OpenTAP Results System

OpenTAP Results System

OVERVIEW

1. End user configures which Result Listeners to use:
 - Text Log
 - CSV File
 - PostgreSQL
 - SQLite Database
 - Custom ResultListener Plugin
2. TestStep stores results using generic API
3. ResultListeners receives results



OpenTAP Results System

PUBLISHING RESULTS

- OpenTAP provides four ways to publish results

Method Name	General Use	Scope
Publish<T>(T result)	For a type T, publishes all the public scalar properties as a single row with N column: • Names of the properties become the column names • Values become the row values	Single Row
Publish<T>(string name, T result)	Similar to the above, but assigns a unique name to the table name.	SingleRow
Publish(string name, List<string> columnNames, params IConvertible[] results)	Publishes a row of data with N column names and N values. The number of columnNames must match the number of size of the Results array.	SingleRow
PublishTable(string name, List<string> columnNames, params Array[] results)	Publishes N rows of data: • The columnNames parameter defines the Column Names. • The columnData parameter is an array of data, with N columns and M rows. Can be called repeatedly to fill up a table.	N Rows

OpenTAP Results System

PUBLISHTABLE()

- Publishing results in a TestStep using PublishTable().

```
// Generate data/limits to be stored.
int[] xValues = new int[PointCount];
double[] yValues = new double[PointCount];
double[] yLimitHigh = new double[PointCount];
double[] yLimitLow = new double[PointCount];

for (var i = 0; i < PointCount; i++)
{
    xValues[i] = 10 * i;
    yValues[i] = Math.Sin(i * 2 * Math.PI / PointCount);

    yLimitHigh[i] = yValues[i] + 0.1;
    yLimitLow[i] = yValues[i] - 0.2;
}

// Store results.
Results.PublishTable("X versus Y",
    new List<string> { "X Values", "Y Values", "High Limit", "Low Limit" },
    xValues, yValues,
    yLimitHigh, yLimitLow);
```

OpenTAP Results System

RESULT LISTENERS

- ResultListeners receive results:

```
public class MyResultListener : ResultListener
{
    // Called on test plan run start.
    public override void OnTestPlanRunStart(TestPlanRun planRun) { }

    // Called on a test step is started.
    public override void OnTestStepRunStart(TestStepRun stepRun) { }

    // Called when a result is received.
    public override void OnResultPublished(Guid stepRun, ResultTable result) { }

    // Called when a test step is completed. At this point no more result will be sent to
    // the result listener for this test step run.
    public override void OnTestStepRunCompleted(TestStepRun stepRun) { }

    // Called when test plan finishes.
    public override void OnTestPlanRunCompleted(TestPlanRun planRun, Stream logStream) { }
}
```

OpenTAP Results System

STORING RESULTS

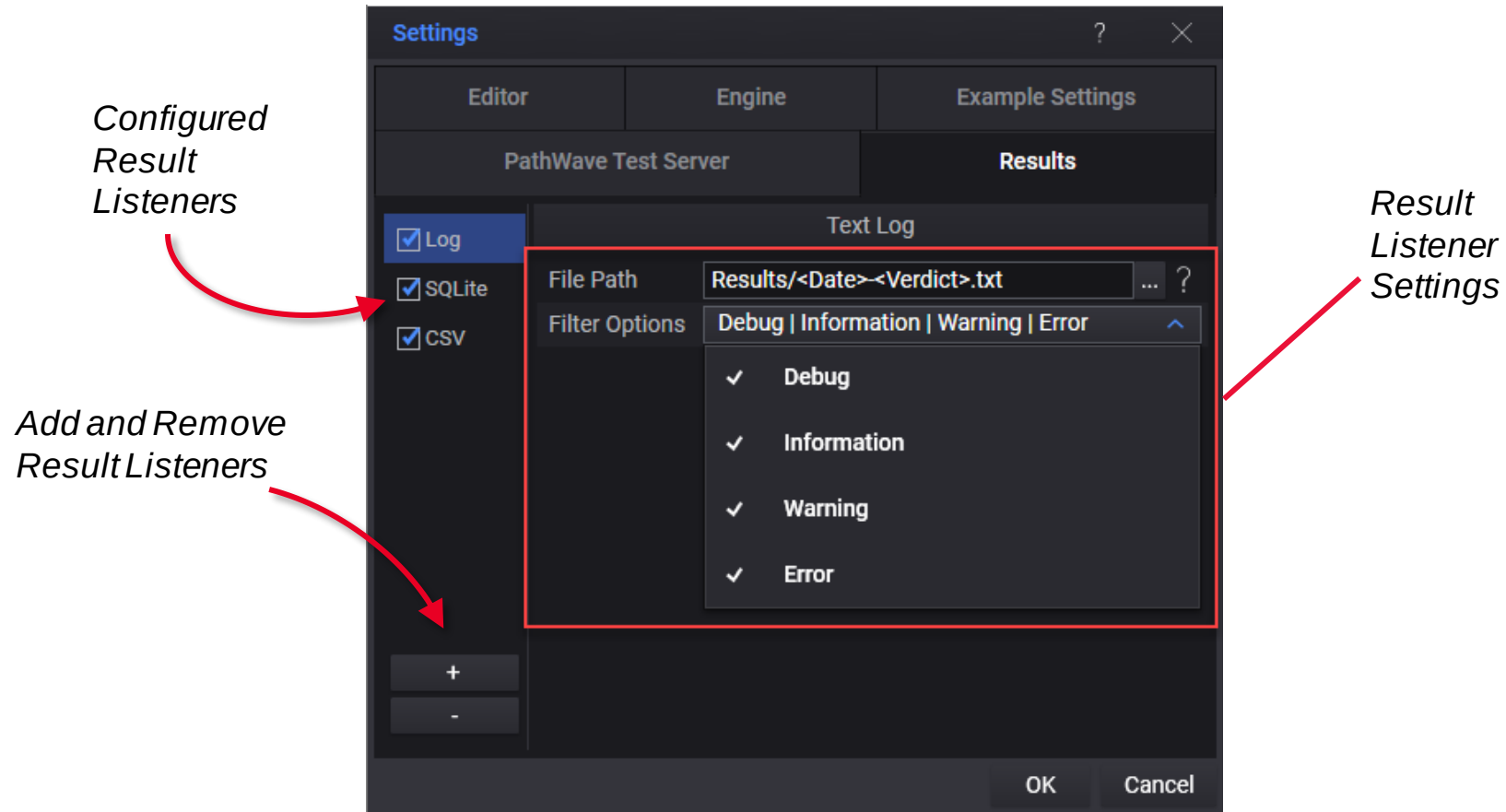
OpenTAP includes ResultListener Plugins for storing results in various ways.

- **Text Log File**
 - Allows the text log to be saved separately for each run
 - File names support macros (for example, to include pass/fail or IMEI/ESN)
- **CSV File**
 - Large 'pivot'-style table is saved as CSV
 - Allows easy post-processing/presentation, such as in Microsoft Excel
 - Can be modified to create customer-specific CSV Result types
- **SQLite Database**
 - Ultra lightweight database based on SQLite
 - During test plan run, automatically stores the test plan and all settings so that all test step results can be tracked back to the specific test plan
 - The lightweight database can be replaced by other production-grade or enterprise-grade storage solutions if needed (using ADO.NET data provider model)

OpenTAP Results System

USER CONFIGURATION

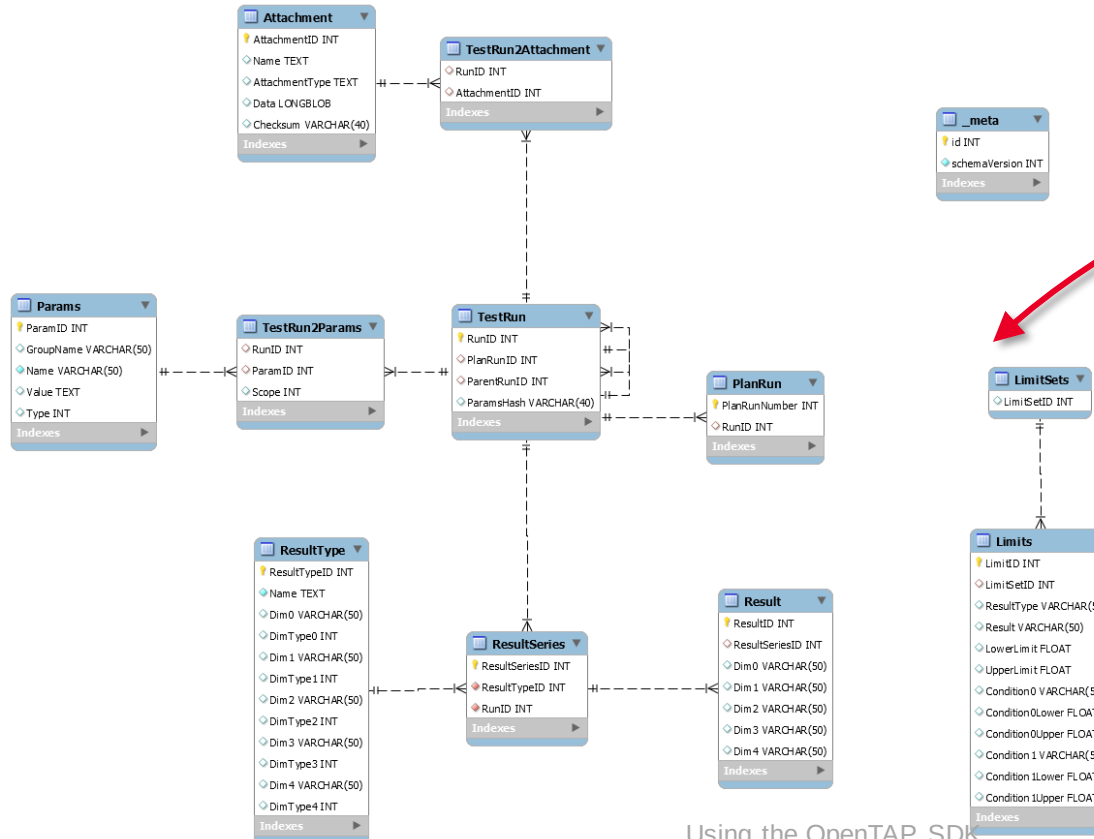
- The end user configures active ResultListeners.



OpenTAP Results System

WRITING YOUR OWN SQL BASED RESULTLISTENER

- **Why:** You have your own dialect of SQL, but want ResultViewer support.
- Schema published in %TAP_PATH\Packages\SDK\OpenTAP Developer Guide.pdf



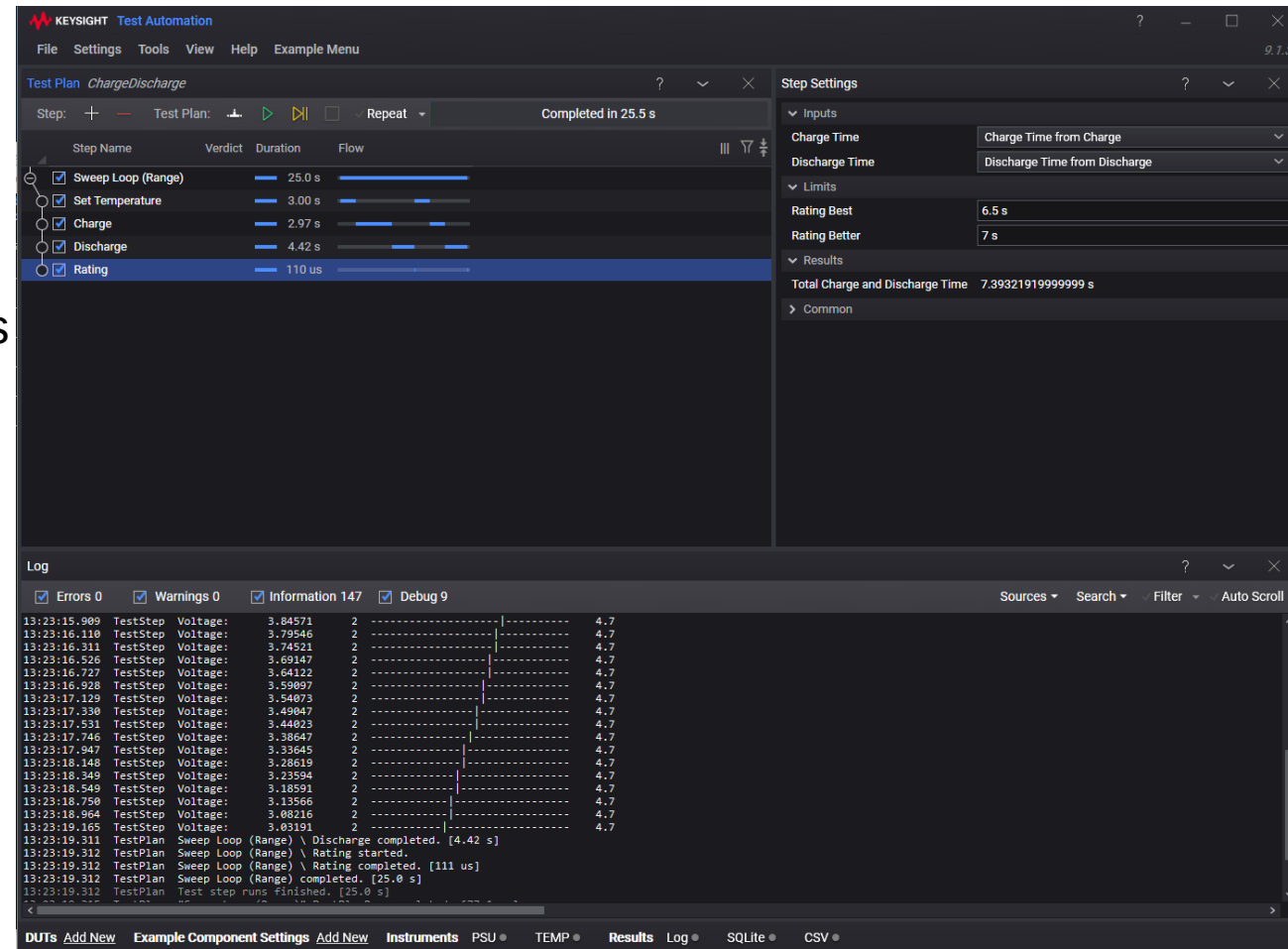
Use this schema to save:

- Results
- Testplan
- Settings
- Logs

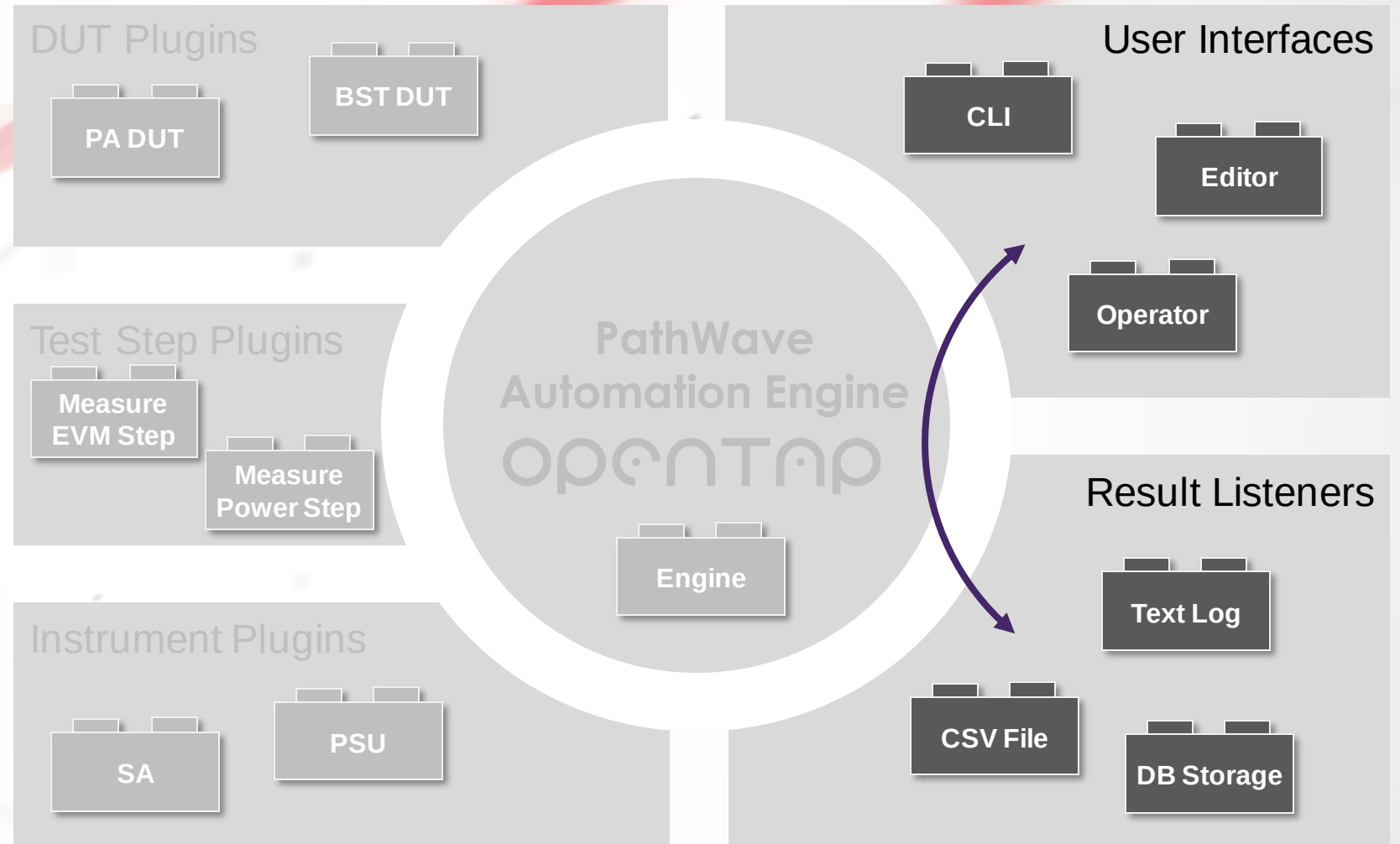
OpenTAP Results System

VIEWING RESULTS

- Results, logs and metadata are stored to Result Listeners during TestPlan execution.
- Results Publishing happens on separate threads to not affect execution time
- Multiple Result Listeners can be written to in parallel



Viewing Results

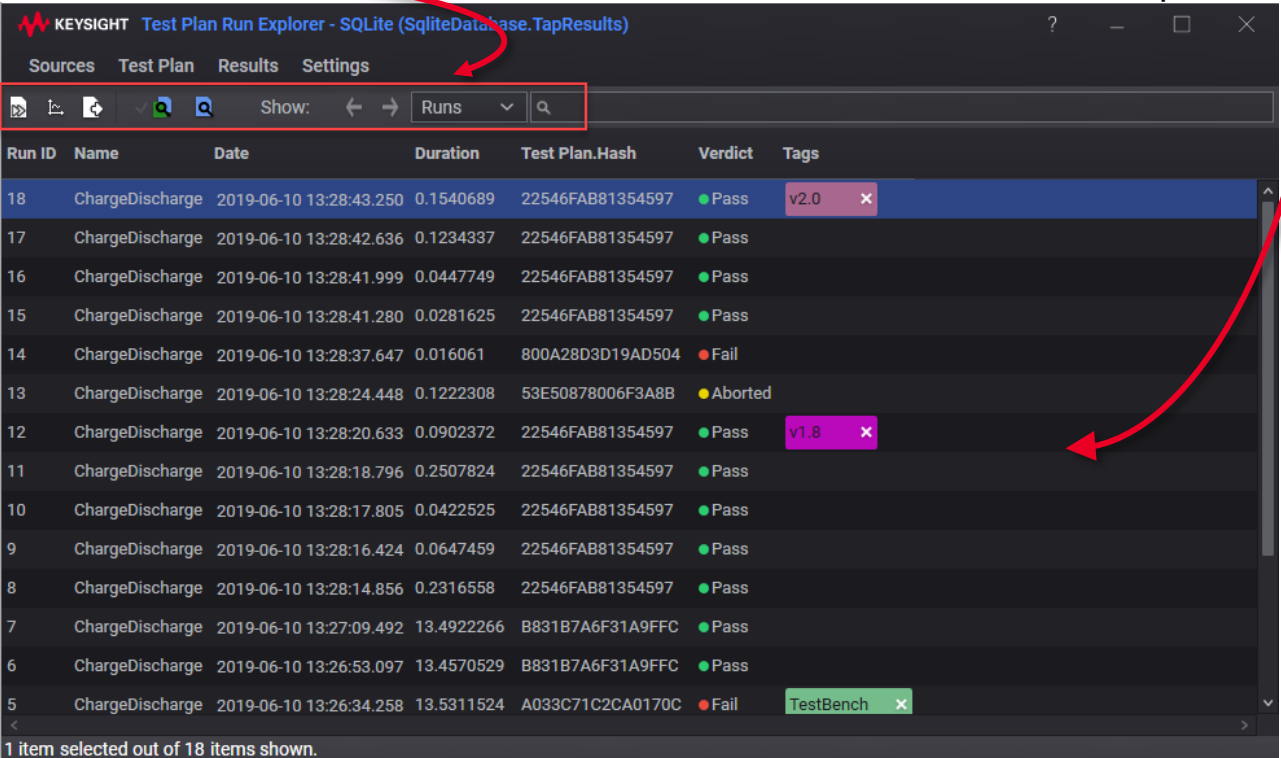


Viewing Results

TEST PLAN RUN EXPLORER

- The Test Plan Run Explorer gives users an overview of all TestPlan runs in a session.
- Users can:
 - Open a TestPlan in the Editor
 - View results in the Result Viewer
 - Export results
 - Compare logs from different runs
 - Search all runs
 - Apply limits to runs

TestPlan runs are displayed with *Run ID, plan name, verdict* and optional user-defined tags



KEYSIGHT Test Plan Run Explorer - SQLite (SqliteDatabase.TapResults)

Sources Test Plan Results Settings

Show: [Icons] Runs [Search]

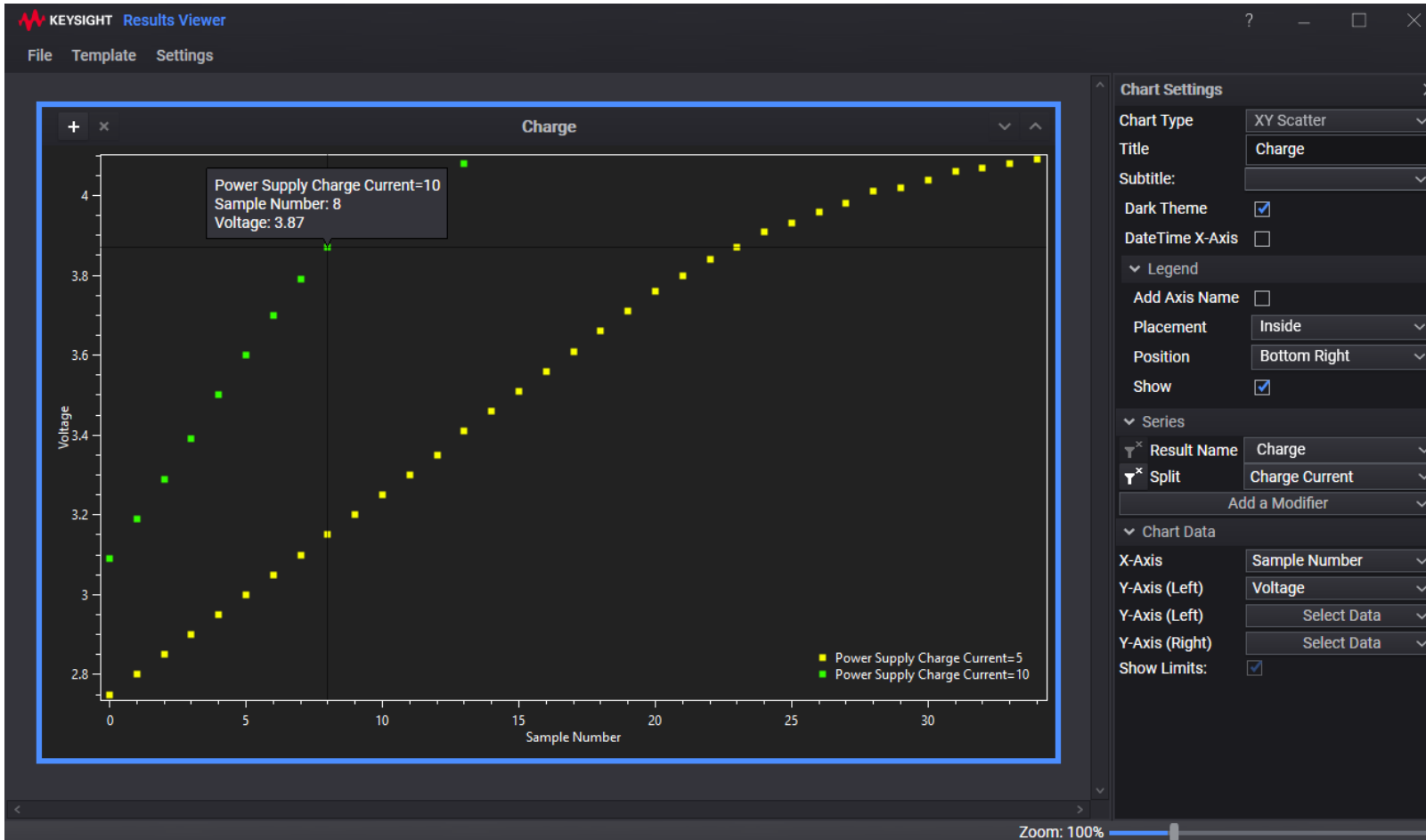
Run ID	Name	Date	Duration	Test Plan.Hash	Verdict	Tags
18	ChargeDischarge	2019-06-10 13:28:43.250	0.1540689	22546FAB81354597	Pass	v2.0
17	ChargeDischarge	2019-06-10 13:28:42.636	0.1234337	22546FAB81354597	Pass	
16	ChargeDischarge	2019-06-10 13:28:41.999	0.0447749	22546FAB81354597	Pass	
15	ChargeDischarge	2019-06-10 13:28:41.280	0.0281625	22546FAB81354597	Pass	
14	ChargeDischarge	2019-06-10 13:28:37.647	0.016061	800A28D3D19AD504	Fail	
13	ChargeDischarge	2019-06-10 13:28:24.448	0.1222308	53E50878006F3A8B	Aborted	
12	ChargeDischarge	2019-06-10 13:28:20.633	0.0902372	22546FAB81354597	Pass	v1.8
11	ChargeDischarge	2019-06-10 13:28:18.796	0.2507824	22546FAB81354597	Pass	
10	ChargeDischarge	2019-06-10 13:28:17.805	0.0422525	22546FAB81354597	Pass	
9	ChargeDischarge	2019-06-10 13:28:16.424	0.0647459	22546FAB81354597	Pass	
8	ChargeDischarge	2019-06-10 13:28:14.856	0.2316558	22546FAB81354597	Pass	
7	ChargeDischarge	2019-06-10 13:27:09.492	13.4922266	B831B7A6F31A9FFC	Pass	
6	ChargeDischarge	2019-06-10 13:26:53.097	13.4570529	B831B7A6F31A9FFC	Pass	
5	ChargeDischarge	2019-06-10 13:26:34.258	13.5311524	A033C71C2CA0170C	Fail	TestBench

1 item selected out of 18 items shown.

Viewing Results

RESULTS VIEWER

- Users can view results in different ways with Results Viewer.



Viewing Results

TIMING ANALYZER

- Timing Analyzer results are based on log files. How does it get them?

The image shows two overlapping software windows from Keysight. The background window is the 'Timing Analyzer' with a 'File' menu open, highlighting 'Open Log File(s)...'. The foreground window is the 'Test Plan Run Explorer - SQLite (SqliteDatabase.TapResults)' with the 'Results' tab selected. A table lists test runs, and a context menu is open over the table, highlighting 'Analyze Session Log'.

In Timing Analyzer, open the log from the **File** menu

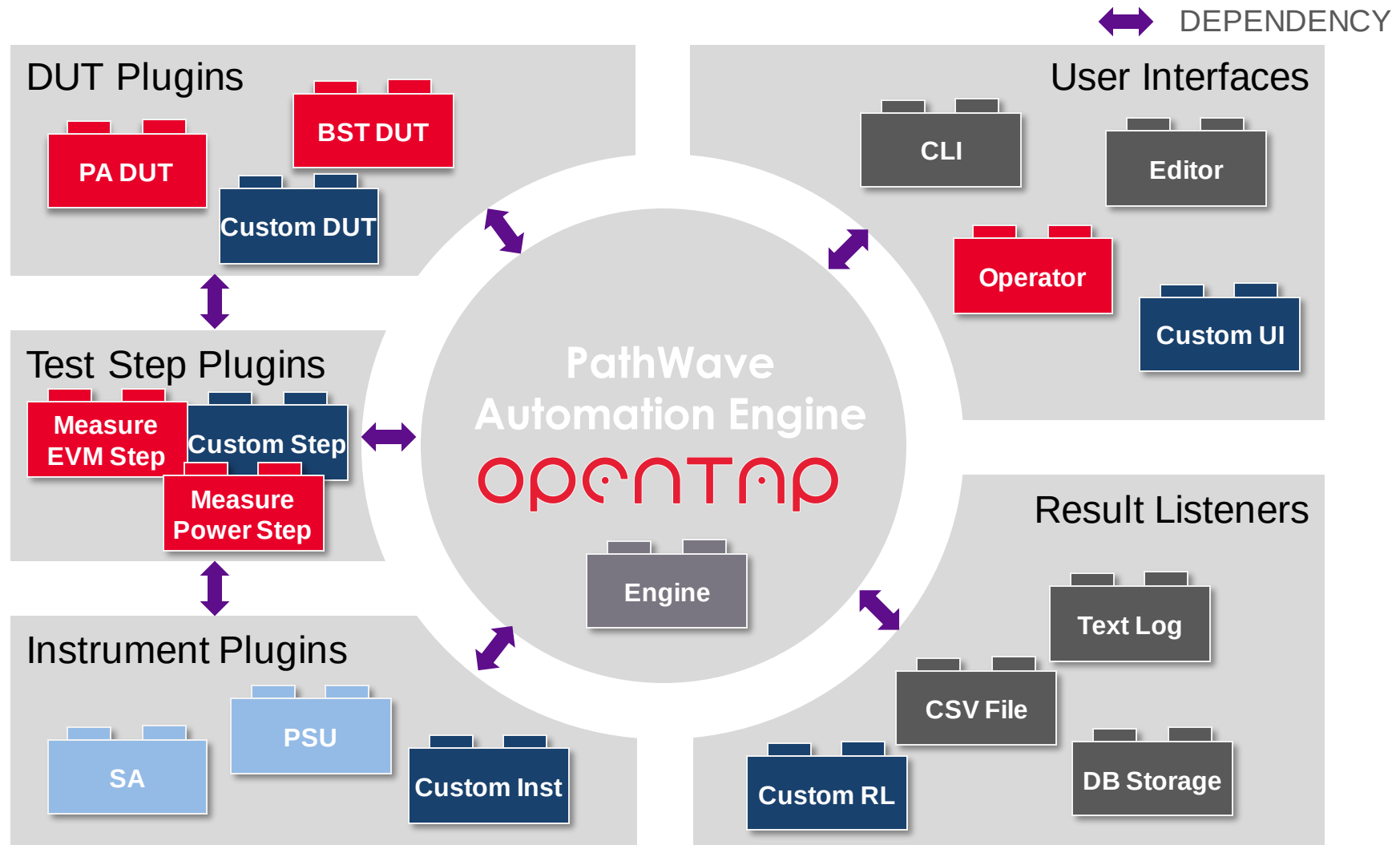
In the GUI, right-click in the Log panel and select **Analyze Session Log**

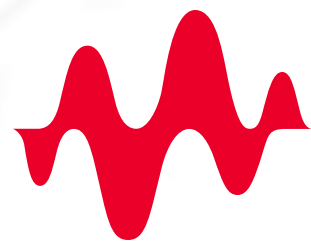
In Test Plan run Explorer, click **Analyze log(s)**

Run ID	Name	Duration	Test Plan.Hash	Verdict	
13	ChargeDischarge	2019-06-10 13:28:24.448	0.1222308	53E50878006F3A8B	Aborted
902372			22516FAB81354597		Pass
507824			225461AB81354597		Pass

Time	Component	Message
13:28:43.250	TestPlan	Starting TestPlan 'ChargeDischarge' on 06/10/2019 13:28:43, 1 of 6 TestSteps enabled
13:28:43.252	TestPlan	Resource "CSV" opened. [43.9 us]
13:28:43.395	CSV	Resource "Log" opened. [63.4 us]
13:28:43.395	Log	Opening database connection (SqliteDatabase.TapResults).
13:28:43.408	SQLite	Resource "Log" opened. [16.0 ms]
13:28:43.411	SQLite	Resource "Log" opened. [16.0 ms]
13:28:43.549	TestPlan	Clear Log Panel
13:28:43.549	TestPlan	Copy
13:28:43.549	TestPlan	Open Session Log
13:28:43.549	TestPlan	Open Session Log Folder
13:28:43.549	TestPlan	Analyze Session Log
13:28:43.551	Log	Resource "SQLite (SqliteDatabase.TapResults)" closed. [1.37 ms]

Extending TAP





KEYSIGHT
TECHNOLOGIES