

Unit & Live Testing for SSIS

- SSIS Tester is the framework that makes unit and integration testing of SSIS packages possible
 - Test implementation follows common methodology as seen in MS Test or Nunit
 - Use any .Net language to implement tests
 - No need to change existing packages
-

System Requirements

- Visual Studio 2008
- Testing Tools for Visual Studio 2008
- SQL Server Integration Services 2008
- .NET Framework 3.5

Or

- Visual Studio 2010 / 2012
- Testing Tools for Visual Studio 2010
- SQL Server Integration Services 2008
- .NET Framework 3.5

Or

- Visual Studio 2010 / 2012
 - Testing Tools for Visual Studio 2010 / 2012
 - SQL Server Integration Services 2012 / 2014
 - .NET Framework 4.0
-

Framework

- Test Engine
 - Monitoring UI
 - Pipeline Components – Data Tap, Fake Source & Fake Destination
-

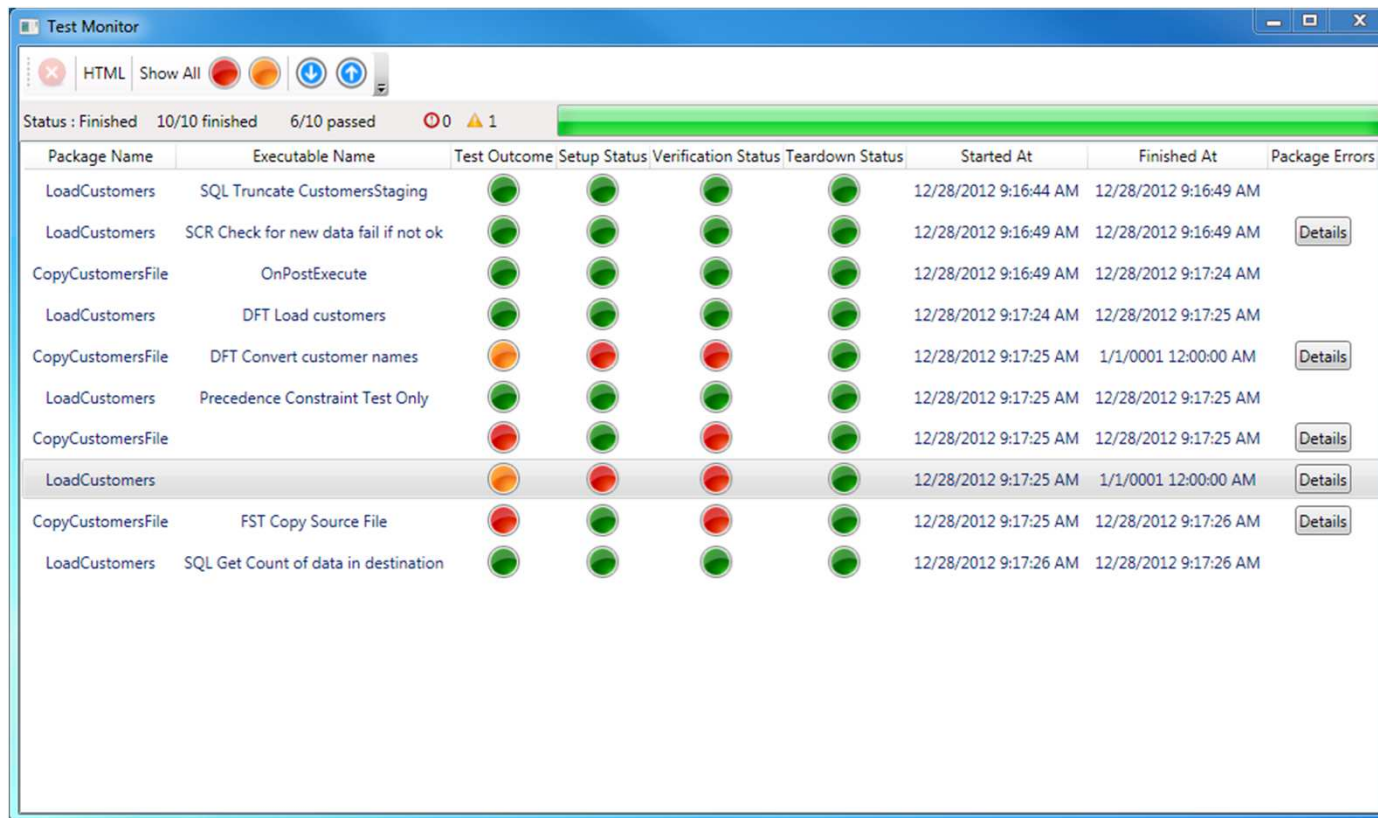
Framework - Test Engine

- Loads packages and tests into repositories
 - Executes tests, packages, single tasks or precedence constraints
 - Exposes testing API
-

Framework - Monitoring UI

- Real time test execution progress
 - Detailed error explanation of failed tests
 - Filter test outcomes
 - Export results to HTML
 - Cancel test execution
-

Framework - Monitoring UI

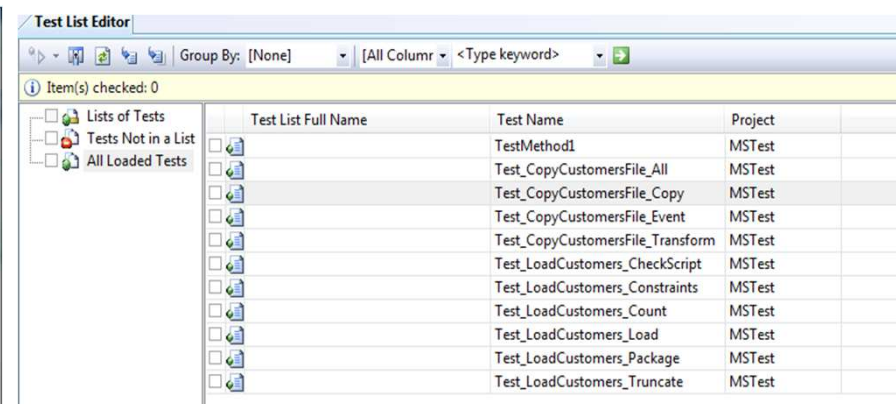
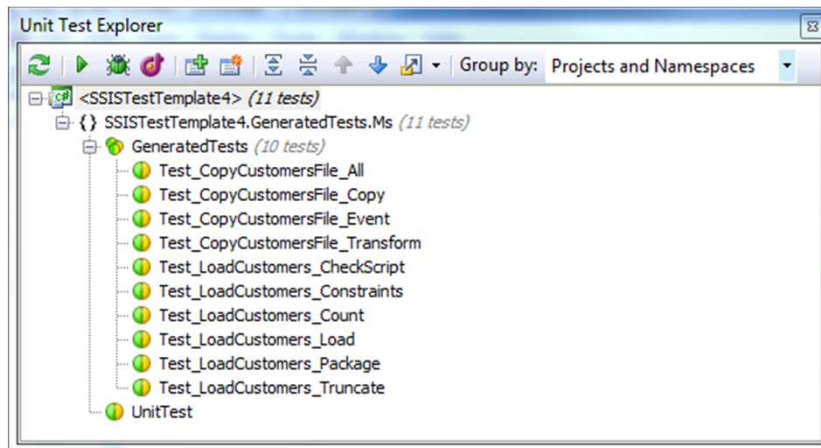


The screenshot shows the Test Monitor application window. At the top, there is a status bar indicating 'Status : Finished', '10/10 finished', and '6/10 passed'. Below this is a table with columns for Package Name, Executable Name, Test Outcome, Setup Status, Verification Status, Teardown Status, Started At, Finished At, and Package Errors. The table contains 10 rows of test results, with some rows highlighted in grey. A 'Details' button is visible next to several rows.

Package Name	Executable Name	Test Outcome	Setup Status	Verification Status	Teardown Status	Started At	Finished At	Package Errors
LoadCustomers	SQL Truncate CustomersStaging	Pass	Pass	Pass	Pass	12/28/2012 9:16:44 AM	12/28/2012 9:16:49 AM	
LoadCustomers	SCR Check for new data fail if not ok	Pass	Pass	Pass	Pass	12/28/2012 9:16:49 AM	12/28/2012 9:16:49 AM	Details
CopyCustomersFile	OnPostExecute	Pass	Pass	Pass	Pass	12/28/2012 9:16:49 AM	12/28/2012 9:17:24 AM	
LoadCustomers	DFT Load customers	Pass	Pass	Pass	Pass	12/28/2012 9:17:24 AM	12/28/2012 9:17:25 AM	
CopyCustomersFile	DFT Convert customer names	Fail	Fail	Fail	Pass	12/28/2012 9:17:25 AM	1/1/0001 12:00:00 AM	Details
LoadCustomers	Precedence Constraint Test Only	Pass	Pass	Pass	Pass	12/28/2012 9:17:25 AM	12/28/2012 9:17:25 AM	
CopyCustomersFile		Fail	Pass	Fail	Pass	12/28/2012 9:17:25 AM	12/28/2012 9:17:25 AM	Details
LoadCustomers		Fail	Fail	Fail	Pass	12/28/2012 9:17:25 AM	1/1/0001 12:00:00 AM	Details
CopyCustomersFile	FST Copy Source File	Fail	Pass	Fail	Pass	12/28/2012 9:17:25 AM	12/28/2012 9:17:26 AM	Details
LoadCustomers	SQL Get Count of data in destination	Pass	Pass	Pass	Pass	12/28/2012 9:17:26 AM	12/28/2012 9:17:26 AM	

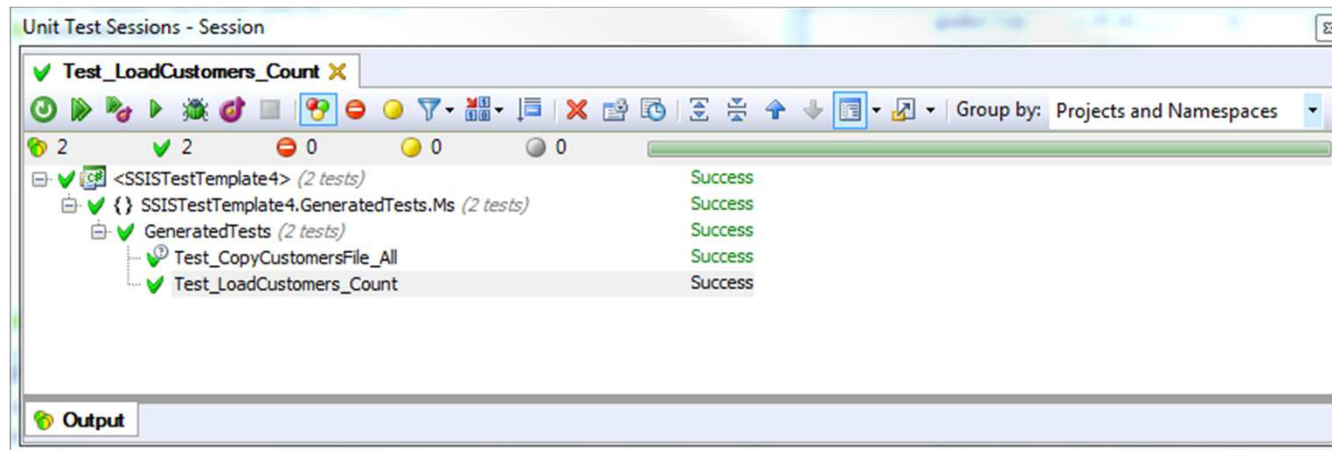
Framework - Visual Studio

- Tests can be executed by using VS Test Editor or ReSharper



Framework - Visual Studio

- Tests can be executed by using VS Test Editor or ReSharper



Framework – Pipeline Components

- Data Tap
 - taps data between components in a data flow
 - captured data can be used in a test to make assertions
 - the goal is to precisely test passing data
 - works with unit & live tests
 - works with SQL Server 2008, 2012 & 2014
-

Framework – Pipeline Components

- Fake Source
 - replaces a source component and channels in data defined in a unit test
 - columns and data types are inherited from the original source
 - the goal is to avoid external source dependencies
 - works with unit tests
 - works with SQL Server 2008, 2012 & 2014
-

Framework – Pipeline Components

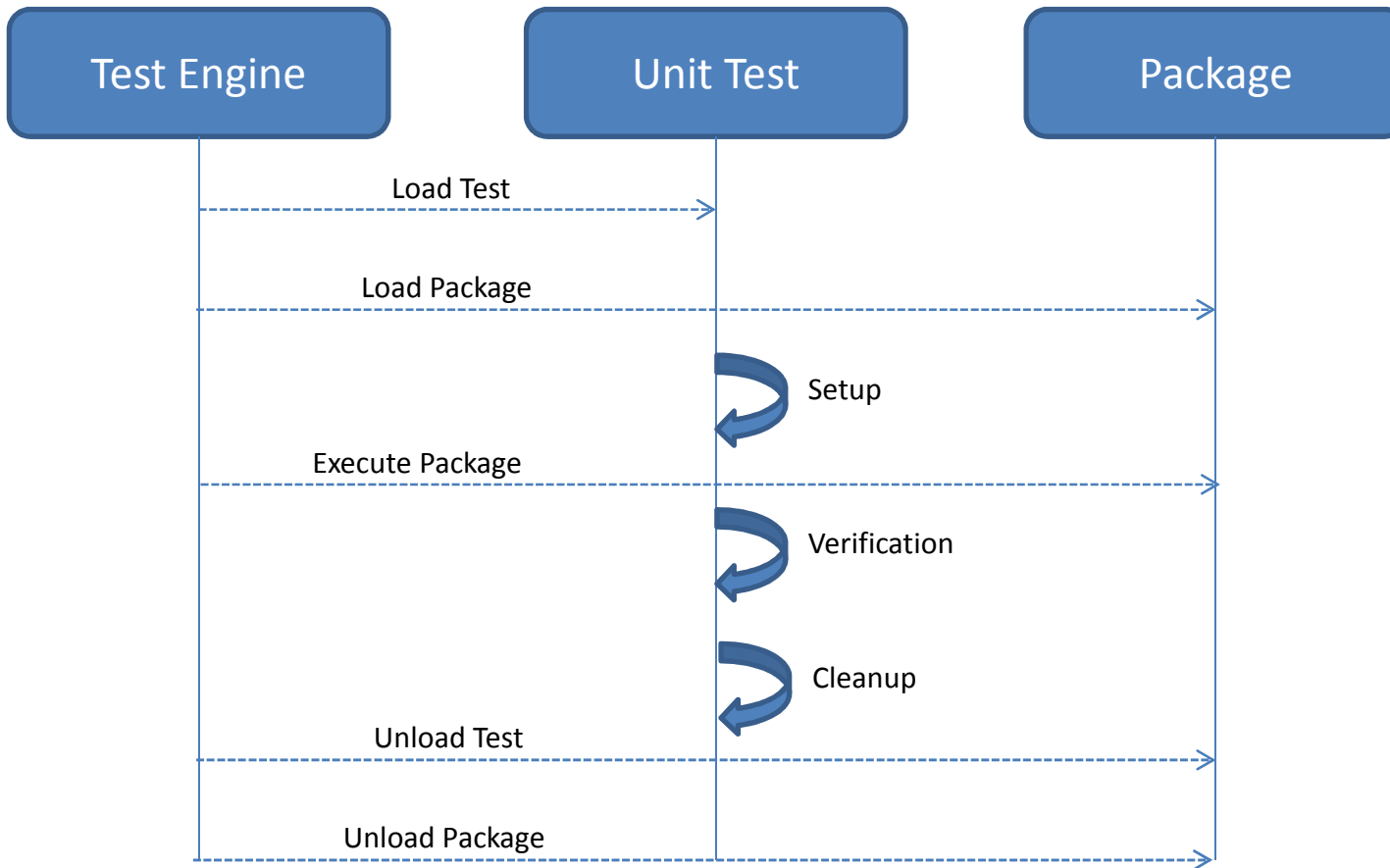
- Fake Destination
 - replaces a destination component and captures data from upstream components
 - columns and data types are inherited from the original destination
 - the goal is to avoid external destination dependencies
 - works with unit tests
 - works with SQL Server 2008, 2012 & 2014
-

Anatomy of Unit Test

- Implemented as a single class
 - Three steps - Setup, Verify and Cleanup
 - Setup prepares package „environment“ – sets connection strings, variables, creates source files...
 - Verify performs assertions after the package executed - checks values of variables, whether data has been copied to db...
 - Cleanup resets the state of the testing environment like it was before test execution
 - Targets one package, task or precedence constraint
 - Lives in a single repository
-

Unit Test Execution

1. Test engine loads unit test definition & target package
 2. Setup step is executed
 3. Package is executed
 4. Verify step is executed
 5. Clean-up step is executed
 6. Target package and test are unloaded
-



 Steps implemented by developer

 Steps without developer control

What is Unit Test for

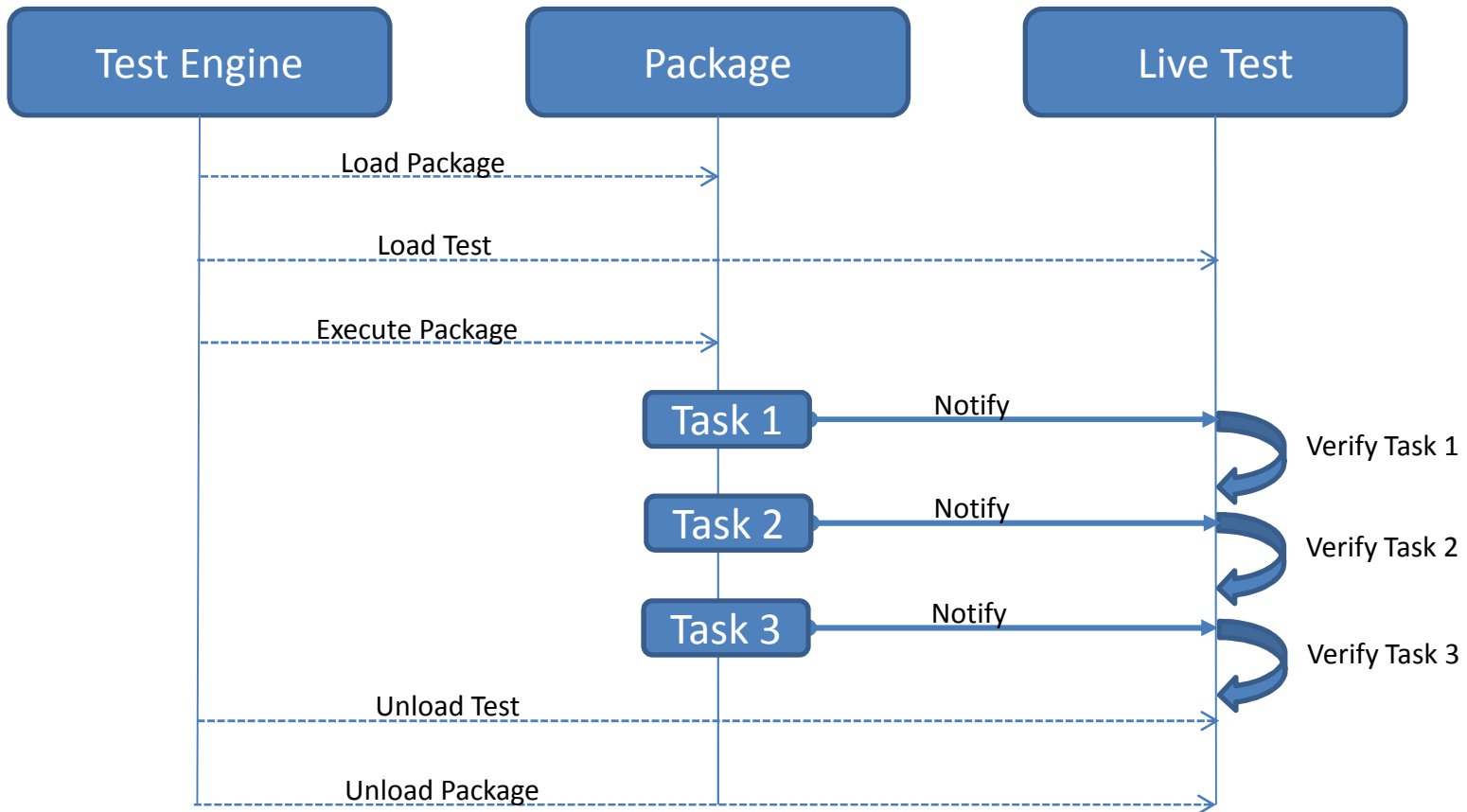
- Isolated tests of single „units of work“ (tasks, precedence constraints) at the package level
 - Isolated tests of single packages
 - Behaviour of packages, tasks and precedence constraints in the prepared (Setup step) environment
-

Anatomy of Live Test

- Implemented as a single class
 - Class can contain more „action methods“ that validate state of tasks or precedence constraints
 - „Action methods“ can access database, file system, package variables, properties, connections...
 - „Action methods“ validate state after the particular package, task or precedence constraint has executed
 - You do not control if and when the „action method“ is executed – it depends on the package execution
-

Live Test Execution

1. Test Engine loads live tests
 2. Test Engine loads, prepares & starts the „main“ package – entry point of your ETL
 3. „Main“ package executes all sub-packages via „Execute Package Tasks“
 4. Live test is executed for each target package and/or single task and/or precedence constraint
 5. Live test validate state after the target has executed
-



 Steps implemented by developer

 Steps without developer control

What is Live Test for

- Verify functional, performance and other aspects of an ETL process
 - Tests packages, their dependancies and their behaviour in the ETL process
 - Tests in a real, non-prepared environment (no Setup step)
-