

# TESTSTAND™ PROCESS MODELS

You can better understand the information in this document if you have already read the *Process Models* section in Chapter 1, *TestStand Architecture Overview*, and Chapter 14, *Process Models*, of the *TestStand User Manual*. This document assumes an understanding of process models that those chapters provide.

The purpose of this document is to provide a detailed overview of the architecture and implementation of the process models that ship with TestStand.

## Contents

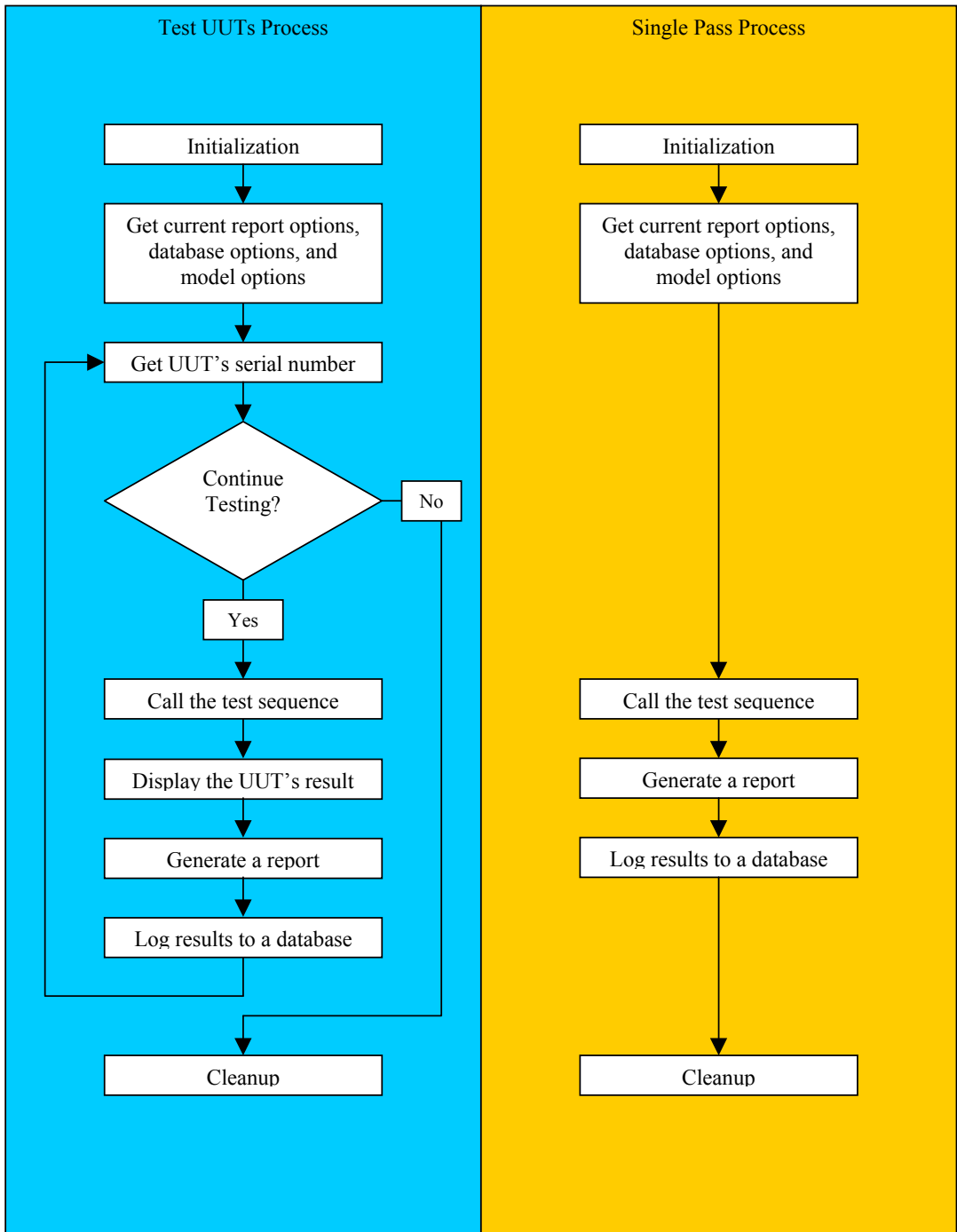
---

Architecture of the TestStand Models .....	2
Details of the Sequential Model .....	7
The Sequences .....	7
Test UUTs .....	11
Single Pass .....	13
Details of the Parallel Model .....	13
The Sequences .....	13
Test UUTs .....	18
Test UUTs – Test Socket Entry Point .....	19
Single Pass .....	20
Single Pass – Test Socket Entry Point .....	21
Details of the Batch Model .....	21
The Sequences .....	21
Test UUTs .....	28
Test UUTs – Test Socket Entry Point .....	31
Single Pass .....	33
Single Pass – Test Socket Entry Point .....	35
Support Files for the TestStand Process Models .....	36
Report Generation Functions and Sequences .....	37

# Architecture of the TestStand Models

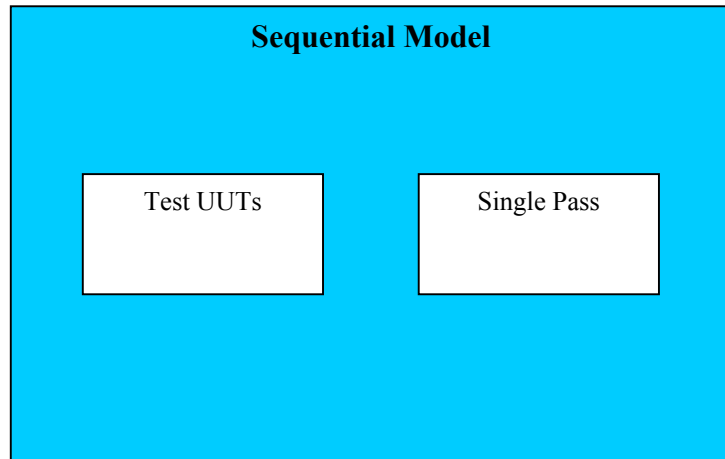
---

The sequential, parallel, and batch models all have the same basic structure within which they run a test sequence. Using their Test UUTs or Single Pass entry point, these models run the user's test sequence, generate a report, and log results to a database according to the configuration settings the user provides. The following diagram illustrates the basic process that these models follow:

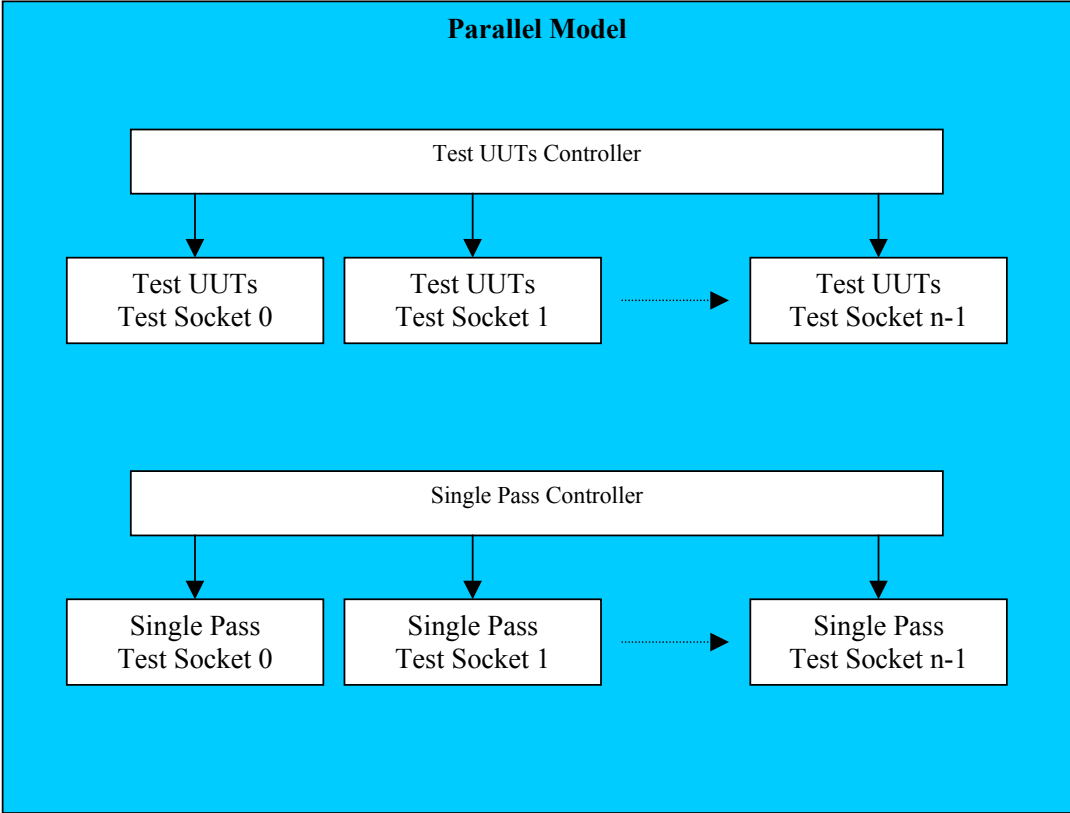


The main difference between the process models is how many UUTs on which they run this process at a time and how they relate and synchronize the UUTs with each other.

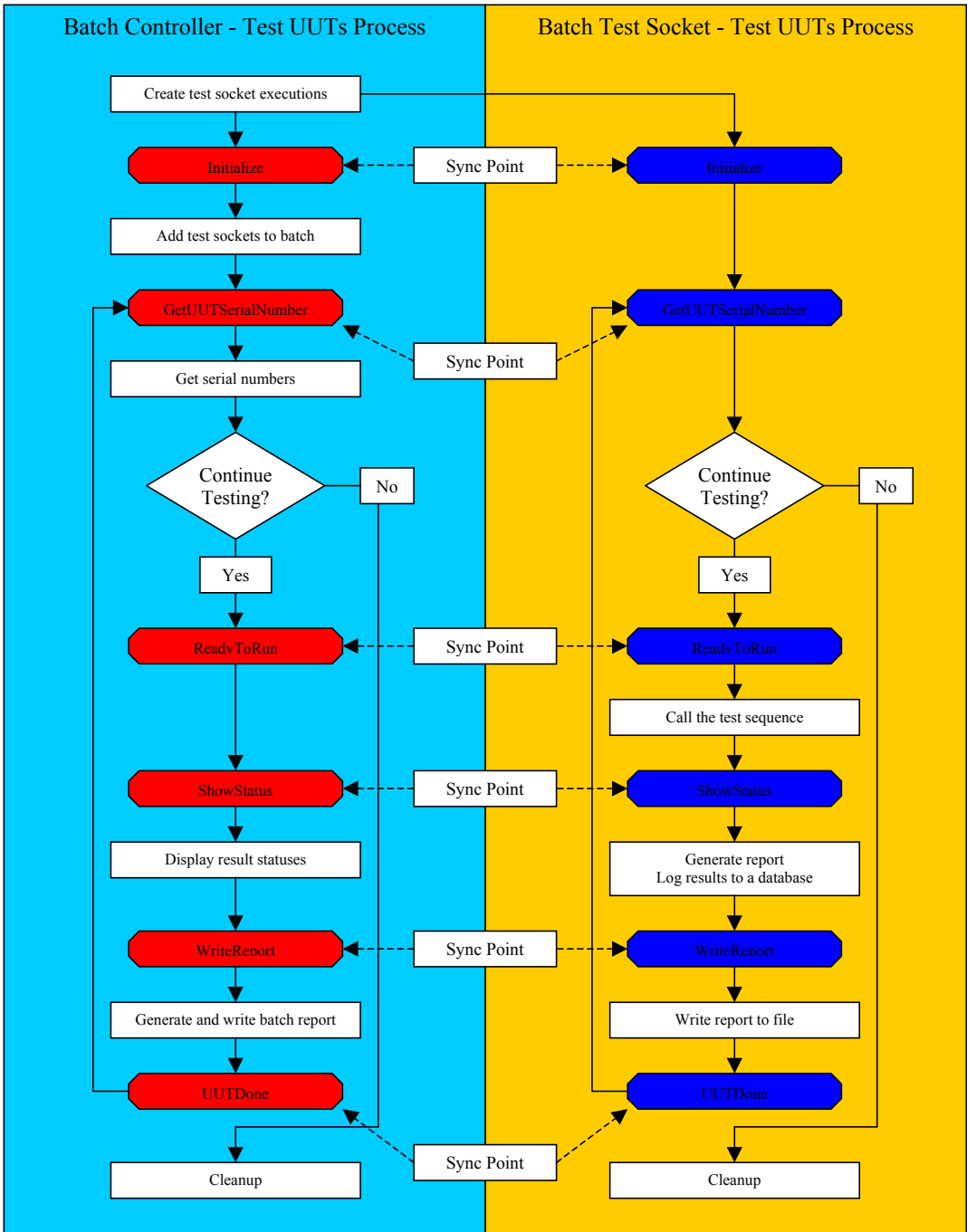
The simplest case is the sequential process model. The sequential model runs the Test UUTs or Single Pass process on only one UUT at a time. When you run the Test UUTs or Single Pass entry point on this model, the entry point sequence itself runs the Test UUTs or Single Pass process directly on the client sequence file.



The parallel model runs the Test UUTs or Single Pass process in parallel on the number of UUTs that the user specifies. Its Test UUTs and Single Pass entry points are actually controller sequences that launch and monitor separate TestStand executions for each instance of the Test UUTs or Single Pass process they run in parallel. The model refers to these instances as test sockets.



The batch model, similar to the parallel model, runs multiple instances of the Test UUTs or Single Pass process at a time. However these processes do not run independently like they do in the parallel model, instead the controlling sequence contains a higher level version of the process and synchronizes the test socket executions with it so that they run through the steps of the process as a group. The following diagram illustrates this:



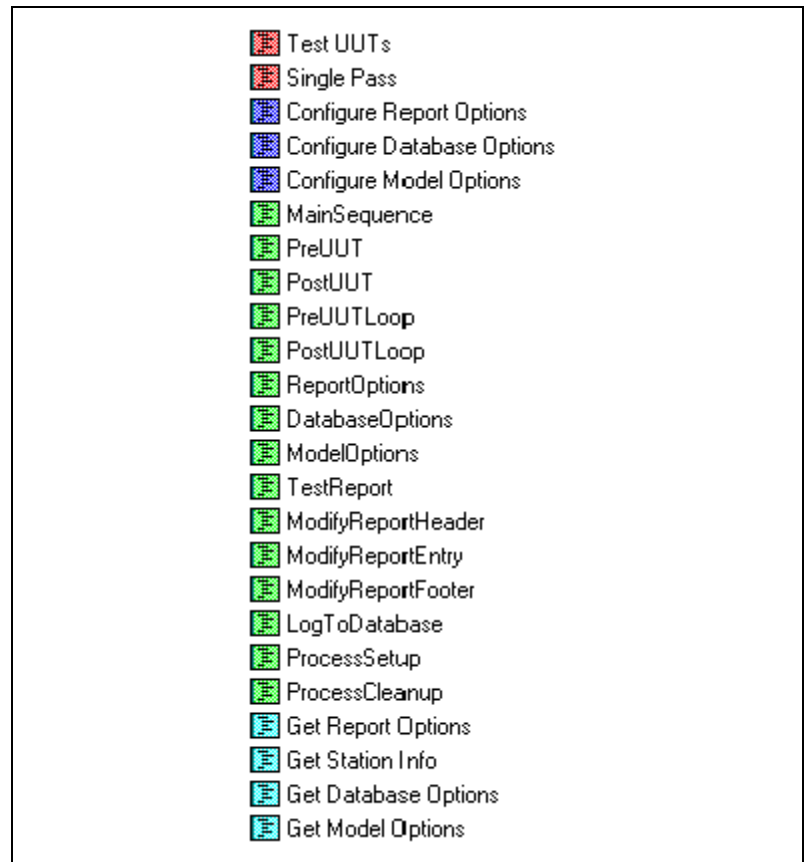
At each sync point in the diagram above, the controlling and test socket executions wait for each other until all are at the same sync point. When they all arrive at the same sync point, the controlling sequence executes its code first, then, before reaching the next sync point, tells the test socket executions to continue executing their code.

## Details of the Sequential Model

---

### The Sequences

The figure below shows a list of all the sequences in the sequential process model (SequentialModel.seq). The first two red sequences are execution entry points, the next three dark blue sequences are configuration entry points, the next fifteen green sequences are model callbacks that you can override in a client sequence file, and the remaining four light blue sequences are utility subsequences that the other sequences call.



- **Test UUTs**—This sequence is an execution entry point that initiates a loop that repeatedly identifies and tests UUTs. When a window for a client sequence file is active, the **Test UUTs** item appears in the **Execute** menu. This document contains more information on the Test UUTs entry point later in this section.
- **Single Pass**—This sequence is an execution entry point that tests a single UUT without identifying it. In essence, the **Single Pass** entry point performs a single iteration of the loop that the **Test UUTs** entry point performs. When a window for a client sequence file is active, the **Single Pass** item appears in the **Execute** menu. This document contains more information on the Single Pass entry point later in this section.
- **Configure Report Options**—This sequence is a configuration entry point that displays a dialog box in which you can specify the contents, format, and pathname of the test report. The settings you make in the dialog box apply to the test station as a whole. The entry point saves the station report options to disk. The entry point appears as **Report Options** in the **Configure** menu. Refer to Chapter 15, *Managing Reports*, for more information on the report options.
- **Configure Database Options**—This sequence is a configuration entry point that displays a dialog box in which you can specify the database logging options. The settings you make in the dialog box apply to the test station as a whole. The entry point saves the station database options to disk. The entry point appears as **Database Options** in the **Configure** menu. Refer to Chapter 18, *Databases*, for more information on the report options.
- **Configure Model Options**—This sequence is a configuration entry point that displays a dialog box in which you can specify model options other than database or report options. The settings you make in the dialog box apply to the test station as a whole. The entry point saves the station model options to disk. The entry point appears as **Model Options** in the **Configure** menu. In the sequential model, this entry point is disabled because the model does not support any model options. Refer to Chapter 14, *Process Models*, for more information on the model options.
- **MainSequence**—This sequence is a model callback that the **Test UUTs** and **Single Pass** entry points call to test a UUT. The **MainSequence** callback is empty in the process model file. The client file must contain a **MainSequence** callback that performs the tests on a UUT.
- **PreUUT**—This sequence is a model callback that displays a dialog box in which the operator enters the UUT serial number. The **Test UUTs** entry point calls the **PreUUT** callback at the beginning of each iteration of the UUT loop. If the operator indicates through the dialog box that



no more UUTs are available for testing, the UUT loop terminates. If the operator chooses to stop testing, the `IdentifyUUT` step sets `ContinueTesting` parameter, which is a local variable that the Test UUTs sequence passes, to `False`. If the operator enters a serial number, the `IdentifyUUT` step stores the serial number in the `UUT.SerialNumber` parameter, which is a local variable that the Test UUTs sequence passes.

- `PostUUT`—This sequence is a model callback that displays a banner indicating the result of the test that the `MainSequence` callback in the client file performs on the UUT. The `Test UUTs` entry point calls the `PostUUT` callback at the end of each iteration of the UUT loop.
- `PreUUTLoop`—This sequence is a model callback that the `Test UUTs` entry point calls before the UUT loop begins. The `PreUUTLoop` callback in the process model file is empty.
- `PostUUTLoop`—This sequence is a model callback that the `Test UUTs` entry point calls after the UUT loop terminates. The `PostUUTLoop` callback in the process model file is empty.
- `ReportOptions`—This sequence is a model callback that the execution entry points call through the `GetReportOptions` subsequence. After reading the test station report options from disk, `GetReportOptions` calls the `ReportOptions` callback to give the client sequence file an opportunity to modify the report options. For example, you might want to force the report format to be ASCII text for a particular client sequence file. The `ReportOptions` callback in the process model file is empty.
- `DatabaseOptions`—This sequence is a model callback that the execution entry points call through `GetDatabaseOptions` subsequence. After reading the test station database options from disk, `GetDatabaseOptions` calls the `DatabaseOptions` callback to give the client sequence file an opportunity to modify the database options. The `DatabaseOptions` callback in the process model file is empty.
- `ModelOptions`—This sequence is a model callback that the execution entry points call through the `GetModelOptions` subsequence. After reading the test station model options from disk, `GetModelOptions` calls the `ModelOptions` callback to give the client sequence file an opportunity to modify the model options. The `ModelOptions` callback in the process model file is empty.
- `TestReport`—This sequence is a model callback that the execution entry points call to generate the contents of the test report for one UUT. You can override the `TestReport` callback in the client file if you want to change its behavior entirely. The default process model defines a test report for a single UUT as consisting of a header, an entry for each step result, and a footer. If you do not override the `TestReport` callback, you can override the `ModifyReportHeader`,

`ModifyReportEntry`, and `ModifyReportFooter` model callbacks to customize the test report.

Depending on the setting you make in the Report Options dialog box, the `TestReport` callback determines whether `TestStand` builds the report body with sequences or with a DLL. If you select the sequence report generation option, `TestReport` calls the `AddReportBody` sequence in either `ReportGen_txt.seq` or `ReportGen_html.seq` to build the report body. The sequence report generator uses a series of sequences with steps that recursively process the result list for the execution. If you select the DLL report generation option, `TestReport` calls a single function in `modelsupport2.dll` to build the entire report body before returning. You can access the project and source code for this LabWindows/CVI-built DLL. If you select the DLL option, `TestStand` generates reports faster, but `TestStand` does not call `ModifyReportEntry` callbacks.

- `ModifyReportHeader`—This sequence is a model callback that the `TestReport` model callback calls so that the client sequence file can modify the report header. `ModifyReportHeader` receives the following parameters: the UUT, the tentative report header text, and the report options. The `ModifyReportHeader` callback in the process model file is empty.
- `ModifyReportEntry`—This sequence is a model callback that the `TestReport` model callback calls so that the client sequence file can modify the entry for each step result. Through subsequences, `TestReport` calls `ModifyReportEntry` for each result in the result list for the UUT. `ModifyReportEntry` receives the following parameters: an entry from the result list, the UUT, the tentative report entry text, the report options, and a level number that indicates the call stack depth at the time the step executed. The `ModifyReportEntry` callback in the process model file is empty.



**Note** In the Report Options dialog box, you can choose to use sequences or a DLL to produce the report body. If you select the DLL option, `TestStand` generates reports faster, but `TestStand` does not call `ModifyReportEntry` callbacks.

- `ModifyReportFooter`—This sequence is a model callback that the `TestReport` model callback calls so that the client sequence file can modify the report footer. `ModifyReportFooter` receives the following parameters: the UUT, the tentative report footer text, and the report options. The `ModifyReportFooter` callback in the process model file is empty.
- `LogToDatabase`—This sequence is a model callback that the execution entry points call to populate a database with the results for one UUT. You can override the `LogToDatabase` callback in the client file if you want to change its behavior entirely. `LogToDatabase`

receives the following parameters: the UUT, the result list for the UUT, and the database options.

- **Process Setup**—This sequence is a model callback that the execution entry points call from their setup step groups to give the client file an opportunity to execute any setup steps that must run only once during the execution of the process model.
- **Process Cleanup**— This sequence is a model callback that the execution entry points call from their cleanup step groups to give the client file an opportunity to execute any cleanup steps that must run only once during the execution of the process model.
- **Get Report Options**—This sequence is a utility sequence that the execution entry points call at the beginning of execution. `Get Report Options` reads the report options and then calls the `ReportOptions` callback to give you an opportunity to modify the report options in the client file.
- **Get Station Info**—This sequence is a utility sequence that the execution entry points call at the beginning of execution. `Get Station Info` callback identifies the test station name and the current user.
- **Get Database Options**—This sequence is a utility sequence that the execution entry points call at the beginning of execution. `Get Database Options` reads the database options and then calls the `DatabaseOptions` callback to give you an opportunity to modify the database options in the client file.
- **Get Model Options**—This sequence is a utility sequence that the execution entry points call at the beginning of execution. `Get Model Options` reads the model options and then calls the `ModelOptions` callback to give you an opportunity to modify the model options in the client file.

## Test UUTs

The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Call <code>PreUUTLoop</code> model callback.	Callback in model file is empty.
2	Call <code>Get Model Options</code> utility sequence.	Reads model options from disk. Calls <code>ModelOptions</code> model callback to allow client to modify options.
3	Call <code>Get Station Info</code> utility sequence.	Identifies the test station name and the current user.

Action Number	Description	Remarks
4	Call Get Report Options utility sequence.	Reads report options from disk. Calls ReportOptions model callback to allow client to modify options.
5	Call Get Database Options utility sequence.	Reads database options from disk. Calls DatabaseOptions model callback to allow client to modify options.
6	Increment the UUT index.	—
7	Call PreUUT model callback.	Obtains the UUT serial number from the operator.
8	If no more UUTs, go to action 17.	—
9	Determine the report file pathname.	—
10	Clear information from previous loop iteration.	Discard previous results and clear the report and failure stack.
11	Call MainSequence model callback.	MainSequence callback in client performs the tests on the UUT.
12	Call PostUUT model callback.	Displays a pass, fail, error, or terminate banner.
13	Call TestReport model callback.	Generates test report for the UUT.
14	Call LogToDatabase model callback.	Log test results to database for the UUT.
15	Write the UUT report to disk.	Can append to an existing file or create a new file.
16	Loop back to action 6	—
17	Call PostUUTLoop model callback.	Callback in model file is empty.

## Single Pass

The following table lists the most significant steps of this entry point:









































Action Number	Description	Remarks
1	Call <code>Get Model Options</code> utility sequence.	Reads model options from disk. Calls <code>ModelOptions</code> model callback to allow client to modify options.
2	Call <code>Get Station Info</code> utility sequence.	Identifies the test station name and the current user.
3	Call <code>Get Report Options</code> utility sequence.	Reads report options from disk. Calls <code>ReportOptions</code> model callback to allow client to modify options.
4	Call <code>Get Database Options</code> utility sequence.	Reads database options from disk. Calls <code>DatabaseOptions</code> model callback to allow client to modify options.
5	Determine the report file pathname.	—
6	Call <code>MainSequence</code> model callback.	<code>MainSequence</code> callback in client performs the tests on the UUT.
7	Call <code>TestReport</code> model callback.	Generates test report for the UUT.
8	Write the UUT report to disk.	Can append to an existing file or create a new file.
9	Call <code>LogToDatabase</code> model callback.	Log test results to database for the UUT.

## Details of the Parallel Model

### The Sequences

The figure below shows a list of all the sequences in the parallel process model (`ParallelModel.seq`). The first two red sequences are the main execution entry points, the next fourteen light blue sequences are utility sequences that the main execution entry points use, the next two red sequences are hidden execution entry points that the main execution entry points use to start the test socket executions, the next three dark blue sequences are configuration entry points, the next fifteen green sequences are model callbacks that you can override in a client sequence file, and the

remaining four light blue sequences are utility subsequences that the other sequences call.

-  Test UUTs
-  Single Pass
-  Initialize TestSocket
-  Tile Execution Windows
-  Monitor Threads
-  ProcessDialogRequests
-  Run UUT Info Dialog
-  Continue TestSocket
-  Terminate TestSocket
-  Abort TestSocket
-  Restart TestSocket
-  Terminate All TestSockets
-  Abort All TestSockets
-  Stop All TestSockets
-  View TestSocket Report
-  View TestSocket Report - Current Only
-  Test UUTs -- Test Socket Entry Point
-  Single Pass -- Test Socket Entry Point
-  Configure Report Options
-  Configure Database Options
-  Configure Model Options
-  MainSequence
-  PreUUT
-  PostUUT
-  PreUUTLoop
-  PostUUTLoop
-  ReportOptions
-  DatabaseOptions
-  ModelOptions
-  TestReport
-  ModifyReportHeader
-  ModifyReportEntry
-  ModifyReportFooter
-  LogToDatabase
-  ProcessSetup
-  ProcessCleanup
-  Get Station Info
-  Get Report Options
-  Get Database Options
-  Get Model Options

- **Test UUTs**—This sequence is an execution entry point that controls the test socket executions it creates using the `Test UUTs - Test Socket Entry Point` sequence. When a window for a client sequence file is active, the **Test UUTs** item appears in the **Execute** menu. This document contains more information on the Test UUTs entry point later in this section.
- **Single Pass**—This sequence is an execution entry point that controls the test socket executions it creates using the `Single Pass - Test Socket Entry Point` sequence. When a window for a client sequence file is active, the **Single Pass** item appears in the **Execute** menu. This document contains more information on the Single Pass entry point later in this section.
- **Initialize TestSocket**—The controlling execution calls this sequence to initialize the data for and create the test socket executions.
- **Tile Execution Windows**—The controlling execution calls this sequence to tile the test socket execution windows by building a list of executions and posting a `UIMessage` to the operator interface requesting it to tile the execution windows.
- **Monitor Threads**—The `ProcessDialogRequests` sequence calls this sequence periodically from the controlling execution to poll to see whether any of the test socket executions have been terminated or aborted. If any have, it updates the `ModelData` for that test socket to indicate its new state and tells the dialog, if any, to update its display for that test socket.
- **ProcessDialogRequests**—The controlling execution calls this sequence from the `Test UUTs` sequence. It loops waiting for requests that the dialog enqueues into the `ModelData.DialogRequestQueue`. Those requests are names of sequences to call. When this sequence receives such a request it calls the requested sequence. Additionally, this sequence periodically calls the `Monitor Threads` sequence to check to make sure the test socket executions are still running and update information about them if they are not.
- **Run UUT Info Dialog**—The controlling execution calls this sequence from a new thread. This sequence initializes and runs the modeless dialog that the `Test UUTs` entry point uses to allow the user to control the test socket executions.
- **Continue TestSocket**—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence sets a notification for the test socket that the request specifies allowing the test socket execution to continue. The test socket execution waits on this notification in its default implementation of the `PreUUT` and `PostUUT` callbacks.

- `Terminate TestSocket`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence terminates the execution for the test socket that the request specifies.
- `Abort TestSocket`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence aborts the execution for the test socket that the request specifies.
- `Restart TestSocket`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence restarts the execution for the test socket that the request specifies. After restarting the execution, this sequence optionally, depending on model option settings, re-tiles the execution windows to include the one it restarts.
- `Terminate All TestSocket`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence terminates all of the test socket executions.
- `Abort All TestSocket`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence aborts all of the test socket executions.
- `Stop All TestSocket`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence sets a flag for each test socket execution telling them to stop after they complete their current UUT test sequence and sets a notification to allow them to execute to that point without interruption.
- `View TestSocket Report`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence launches a report viewer on the report file for the test socket that the request specifies.
- `View TestSocket Report - Current Only`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence launches a report viewer for the report last generated for the test socket that the request specifies. This sequence differs from the `View TestSocket Report` sequence in that it only shows the last report rather than the whole report file.
- `Test UUTs - Test Socket Entry Point`—This entry point is never displayed to the user. Instead it is used by the controlling execution to create the `TestSocket` executions. If you insert a new step in this sequence, disable the `Record Results` option for the step. This sequence implements the `Test UUTs` process for the test socket executions. This document contains more information on this entry point later in this section.
- `Single Pass - Test Socket Entry Point`—This entry point is never displayed to the user. Instead it is used by the controlling execution to create the `TestSocket` executions. If you insert a new step in this sequence, disable the `Record Results` option for the step. This



sequence implements the Single Pass process for the test socket executions. This document contains more information on this entry point later in this section.

- `Configure Report Options`, `Configure Database Options`, and `Configure Model Options`—See the descriptions of these sequences in the section of this document on the Sequential model for more information.
- `MainSequence`—This sequence is a model callback that the `Test UUTs - Test Socket Entry Point` and `Single Pass - Test Socket Entry Point` sequences call to test a UUT. The `MainSequence` callback is empty in the process model file. The client file must contain a `MainSequence` callback that performs the tests on a UUT.
- `PreUUT`—This sequence is a model callback that calls into the modeless dialog box that the controlling execution creates in which the operator enters the UUT serial numbers for the test sockets. The `Test UUTs - Test Socket Entry Point` sequence calls the `PreUUT` callback at the beginning of each iteration of the UUT loop. If the operator indicates through the dialog box that no more UUTs are available for testing, the UUT loop terminates. If the operator chooses to stop testing, the code for the dialog box sets `TestSocket.ContinueTesting` parameter to `False`. If the operator enters a serial number, the code for the dialog box stores the serial number in the `TestSocket.UUT.SerialNumber` parameter.
- `PostUUT`—This sequence is a model callback that calls into the modeless dialog that the controlling execution creates to tell it to display a banner indicating the result of the test that the `MainSequence` callback in the client file performs on the UUT. The `Test UUTs - Test Socket Entry Point` calls the `PostUUT` callback at the end of each iteration of the UUT loop.
- `PreUUTLoop`—This sequence is a model callback that the `Test UUTs - Test Socket Entry Point` sequence calls before the UUT loop begins. The `PreUUTLoop` callback in the process model file is empty.
- `PostUUTLoop`—This sequence is a model callback that the `Test UUTs - Test Socket Entry Point` sequence calls after the UUT loop terminates. The `PostUUTLoop` callback in the process model file is empty.
- `ReportOptions`, `DatabaseOptions`, `ModelOptions`, `TestReport`, `ModifyReportHeader`, `ModifyReportEntry`, `ModifyReportFooter`, and `LogToDatabase`—See the descriptions of these sequences in the section of this document on the Sequential model for more information.
- `Process Setup`—This sequence is a model callback that the `Test UUTs` and `Single Pass` execution entry points call from their setup

step groups to give the client file an opportunity to execute any setup steps that must run only once during the execution of the process model. These setup steps are run from the controlling execution only, the test socket executions do not call this callback.

- **Process Cleanup**—This sequence is a model callback that the `Test UUTs` and `Single Pass` execution entry points call from their cleanup step groups to give the client file an opportunity to execute any cleanup steps that must run only once during the execution of the process model. These cleanup steps are run from the controlling execution only, the test socket executions do not call this callback.
- **Get Station Info, Get Report Options, Get Database Options, and Get Model Options**—See the descriptions of these sequences in the section of this document on the `Sequential` model for more information.

## Test UUTs

This entry point is the sequence that the controlling execution runs. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Call <code>Get Model Options</code> utility sequence.	Reads model options from disk. Calls <code>ModelOptions</code> model callback to allow client to modify options.
2	Call <code>Get Station Info</code> utility sequence.	Identifies the test station name and the current user.
3	Call <code>Get Report Options</code> utility sequence.	Reads report options from disk. Calls <code>ReportOptions</code> model callback to allow client to modify options.
4	Call <code>Get Database Options</code> utility sequence.	Reads database options from disk. Calls <code>DatabaseOptions</code> model callback to allow client to modify options.
5	Call <code>Run UUT Info Dialog</code> utility sequence.	Creates a modeless dialog that displays info and gathers serial numbers for the test socket executions.
6	Determine the report file pathname.	Determines the report file path name to use if the report options are configured so that all UUT results for the model are written to the same file.

Action Number	Description	Remarks
7	Create and initialize test socket executions.	See the table for the Test UUTs - Test Socket Entry Point sequence for more information on what the test socket executions do.
8	Call ProcessDialogRequests.	Waits for dialog requests in a loop until the model is ready to be shut down.

## Test UUTs – Test Socket Entry Point

This entry point is the sequence that the test socket executions run. The controlling execution creates the test socket executions in its Test UUTs entry point sequence. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Call PreUUTLoop model callback.	Callback in model file is empty.
2	Increment the UUT index.	—
3	Clear information from previous loop iteration.	Discard previous results and clear the report and failure stack.
4	Call PreUUT model callback.	Obtains the UUT serial number from the operator.
5	If no more UUTs, go to action 13.	—
6	Determine the report file pathname.	—
7	Call MainSequence model callback.	MainSequence callback in client performs the tests on the UUT.
8	Call PostUUT model callback.	Tells the modeless dialog that the controlling execution creates to display a pass, fail, error, or terminate banner for this test socket.
9	Call TestReport model callback.	Generates test report for the UUT.
10	Call LogToDatabase model callback.	Log test results to database for the UUT.

Action Number	Description	Remarks
11	Write the UUT report to disk.	Can append to an existing file or create a new file.
12	Loop back to action 2	—
13	Call PostUUTLoop model callback.	Callback in model file is empty.

## Single Pass

This entry point is the sequence that the controlling execution runs. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Call Get Model Options utility sequence.	Reads model options from disk. Calls ModelOptions model callback to allow client to modify options.
2	Call Get Station Info utility sequence.	Identifies the test station name and the current user.
3	Call Get Report Options utility sequence.	Reads report options from disk. Calls ReportOptions model callback to allow client to modify options.
4	Call Get Database Options utility sequence.	Reads database options from disk. Calls DatabaseOptions model callback to allow client to modify options.
5	Determine the report file pathname.	Determines the report file path name to use if the report options are configured so that all UUT results for the model are written to the same file.
6	Create and initialize test socket executions.	See the table for the Single Pass - Test Socket Entry Point sequence for more information on what the test socket executions do.
7	Wait for test socket executions to complete	—

## Single Pass – Test Socket Entry Point

This entry point is the sequence that the test socket executions run. The controlling execution creates the test socket executions in its *Single Pass* entry point sequence. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Determine the report file pathname.	—
2	Call <code>MainSequence</code> model callback.	<code>MainSequence</code> callback in client performs the tests on the UUT.
3	Call <code>TestReport</code> model callback.	Generates test report for the UUT.
4	Call <code>LogToDatabase</code> model callback.	Log test results to database for the UUT.
5	Write the UUT report to disk.	Can append to an existing file or create a new file.

## Details of the Batch Model

---

### The Sequences

The figure below shows a list of all the sequences in the batch process model (`BatchModel.seq`). The first two red sequences are the main execution entry points, the next fifteen light blue sequences are utility sequences that the main execution entry points use, the next two red sequences are hidden execution entry points that the main execution entry points use to start the test socket executions, the next light blue sequence is a utility sequence that the hidden test socket execution entry points use, the next three dark blue sequences are configuration entry points, the next fifteen green sequences are model callbacks that you can override in a client sequence file, the next four light blue sequences are utility subsequences that the other sequences call, and the remaining eight green sequences are model callbacks unique to the batch model that the main execution entry points of the model call.

 Test UUTs	 PostUUT
 Single Pass	 PreUUTLoop
 Restart TestSocket	 PostUUTLoop
 Initialize TestSocket	 ReportOptions
 Monitor Batch Threads	 DatabaseOptions
 Tile Execution Windows	 ModelOptions
 Add TestSocket Threads to Batch	 TestReport
 Notify TestSocket Threads	 ModifyReportHeader
 All TestSockets Waiting?	 ModifyReportEntry
 ProcessTestSocketRequests	 ModifyReportFooter
 WaitForTestSocket	 LogToDatabase
 ProcessDialogRequests	 ProcessSetup
 Run Batch Info Dialog	 ProcessCleanup
 View TestSocket Report	 Get Station Info
 View TestSocket Report - Current Only	 Get Report Options
 View Batch Report	 Get Database Options
 View Batch Report - Current Only	 Get Model Options
 Test UUTs -- Test Socket Entry Point	 PreBatch
 Single Pass -- Test Socket Entry Point	 PostBatch
 SendControllerRequest	 PreBatchLoop
 Configure Report Options	 PostBatchLoop
 Configure Database Options	 BatchReport
 Configure Model Options	 ModifyBatchReportHeader
 MainSequence	 ModifyBatchReportEntry
 PreUUT	 ModifyBatchReportFooter

- **Test UUTs**—This sequence runs in the controlling execution of the process model. It creates a separate execution for each test socket using the `Test UUTs -- Test Socket Entry Point` sequence, adds the main threads of those executions to a batch synchronization object, and controls the flow of execution using queues and notifications such that all test socket executions execute the main sequence of the client file together as a group. After a group of UUTs executes, this sequence generates a batch report and loops back around to run the client sequence on the next group of UUTs and controls the subsidiary test socket executions to keep them in sync with each other. When a window for a client sequence file is active, the **Test UUTs** item appears in the **Execute** menu. This document contains more information on the Test UUTs entry point later in this section.
- **Single Pass**—This sequence runs in the controlling execution of the process model. It creates a separate execution for each test socket using the `Single Pass -- Test Socket Entry Point` sequence, adds the main threads of those executions to a batch synchronization object,

and controls the flow of execution using queues and notifications such that all test socket executions execute the main sequence of the client file together as a group. After the group of UUTs executes, this sequence generates a batch report and waits for all subsidiary executions to complete. When a window for a client sequence file is active, the **Single Pass** item appears in the **Execute** menu. This document contains more information on the Single Pass entry point later in this section.

- `Restart TestSocket`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence restarts the execution for the test socket that the request specifies.
- `Initialize TestSocket`—This sequence, called by the controlling execution, initializes the data for and creates the test socket executions.
- `Monitor Batch Threads`—`ProcessDialogRequests`, `ProcessTestSocketRequests`, and `WaitForTestSocket` call this sequence periodically from the controlling execution to poll to see whether any of the test socket executions have been terminated or aborted. If any have, it updates the `ModelData` for that test socket to indicate its new state and tells the dialog, if any, to update its display for that test socket.
- `Tile Execution Windows`—This sequence, called by the controlling execution, tiles the test socket execution windows by building a list of executions and posting a `UIMessage` to the operator interface requesting it to tile the execution windows. This sequence only tiles running, non-disabled test socket executions.
- `Add TestSocket Threads to Batch`—The `Test UUTs` and `Single Pass` entry points call this sequence from the controlling execution to add the main threads of the test socket executions to a batch synchronization object. The threads remove themselves from the batch after running the main sequence of the client sequence file. This removal from the batch is done in the `Test UUTs -- Test Socket Entry Point` and the `Single Pass -- Test Socket Entry Point` sequences.
- `Notify TestSocket Threads`—The controlling execution calls this sequence to tell the running test socket execution threads to continue executing from their last call to `SendControllerRequest` in which they block. This sequence optionally waits for each test socket to get to its next call to `SendControllerRequest` (i.e. the next sync point) before telling the next test socket to go. Doing this ensures serial execution of the test socket executions for the sections of their sequences following the location at which they currently block.
- `All TestSockets Waiting?`—The sequence returns true if all running test sockets are waiting for the `WaitingForRequest` parameter or if all test sockets are `Stopped`.

- `ProcessTestSocketRequests`—The controlling execution calls this sequence to wait for the test socket executions to sync up with it at the appropriate point in their execution. When all running test sockets are at their appropriate point in their executions, the sequence returns allowing the controlling execution to continue. While waiting for the test sockets, this sequence monitors the test socket threads to make sure they are still running. If all test sockets stop running this sequence will return to allow the controlling sequence to continue.
- `WaitForTestSocket`—The controlling execution calls this sequence from the `Notify TestSocket Threads` sequence to wait for a test socket execution to get to its next controller request (i.e. its next sync point) before telling the next test socket execution to continue, thus allowing the controlling execution to guarantee that only one test socket runs particular sections of its sequence at a time. This is used to write the test socket reports to a file in test socket index order when the configuration of report options specifies that they are to write reports to the same file.
- `ProcessDialogRequests`—This sequence is called by the controlling execution from the `Test UUTs` sequence. It loops waiting for requests that the dialog enqueues into the `ModelData.DialogRequestQueue`. Those requests are names of sequences to call. When this sequence receives such a request it calls the requested sequence. Additionally this sequence periodically calls the `Monitor Batch Threads` sequence to check to make sure the test socket executions are still running and update information about them if they are not.
- `Run Batch Info Dialog`—The controlling execution calls this sequence from a new thread from the `Test UUTs` entry point. This sequence initializes and runs the dialog that the `Test UUTs` entry point uses to allow the user to enter serial numbers and see the results for a particular run of the batch.
- `View TestSocket Report`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence launches a report viewer on the report file for the test socket that the request specifies.
- `View TestSocket Report - Current Only`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence launches a report viewer for the report last generated for the test socket that the request specifies. This sequence differs from the `View TestSocket Report` sequence in that it only shows the last report rather than the whole report file.
- `View Batch Report`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence launches a report viewer on the report file for the batch report.



- `View Batch Report - Current Only`—This is a dialog request callback that the `ProcessDialogRequests` sequence calls. This sequence launches a report viewer for the batch report last generated. This sequence differs from the `View Batch Report` sequence in that it only shows the last report rather than the whole report file.
- `Test UUTs -- Test Socket Entry Point`—This entry point is never displayed to the user. Instead, the controlling execution uses it to create the test socket executions. If you insert a new step in this sequence, disable the `Record Results` option for the step. This sequence implements the `Test UUTs` process for the test socket executions. This document contains more information on this entry point later in this section.
- `Single Pass - Test Socket Entry Point`— This entry point is never displayed to the user. Instead, the controlling execution uses it to create the test socket executions. If you insert a new step in this sequence, disable the `Record Results` option for the step. This sequence implements the `Single Pass` process for the test socket executions. This document contains more information on this entry point later in this section.
- `SendControllerRequest` — The test socket executions call this sequence to sync up with the controlling execution at various locations in their sequences. They pass a string parameter that indicates the reason and location at which they are attempting to sync up with the other executions. When all of the test socket executions that are running sync up with the controlling sequence at the same location by calling this sequence, the controlling execution's sequence then performs some operations and tells the test socket executions when to continue.
- `Configure Report Options`, `Configure Database Options`, and `Configure Model Options`—See the descriptions of these sequences in the section of this document on the `Sequential` model for more information.
- `MainSequence`—This sequence is a model callback that the `Test UUTs - Test Socket Entry Point` and `Single Pass - Test Socket Entry Point` sequences call to test a UUT. The `MainSequence` callback is empty in the process model file. The client file must contain a `MainSequence` callback that performs the tests on a UUT.
- `PreUUT`—This is a model callback sequence that the test socket executions call to support backwards compatibility with the `TestStand 1.0.x` process model. The implementation of this sequence is empty in the batch process model. You can override this callback in the client sequence file to get the serial number for the UUT, but if doing so, you should also override the `PreBatch` callback, which currently is where

the batch model displays a dialog to get the serial numbers for all of the UUTs in the batch. There is an example of overriding these callbacks in the

`<TestStand>\Examples\Callbacks\BatchModel` directory.

- `PostUUT`—This is a model callback sequence that the test socket executions call to support backwards compatibility with the TestStand 1.0.x process model. The implementation of this sequence is empty in the batch process model. You can override this callback in the client sequence file to display the result status for a UUT, but if doing so, you should also override the `PostBatch` callback, which currently is where the batch model displays a dialog to show the result status for all of the UUTs in the batch. There is an example of overriding these callbacks in the `<TestStand>\Examples\Callbacks\BatchModel` directory.
- `PreUUTLoop`—This sequence is a model callback that the Test UUTs - Test Socket Entry Point sequence calls before the UUT loop begins. The `PreUUTLoop` callback in the process model file is empty.
- `PostUUTLoop`—This sequence is a model callback that the Test UUTs - Test Socket Entry Point sequence calls after the UUT loop terminates. The `PostUUTLoop` callback in the process model file is empty.
- `ReportOptions`, `DatabaseOptions`, `ModelOptions`, `TestReport`, `ModifyReportHeader`, `ModifyReportEntry`, `ModifyReportFooter`, and `LogToDatabase`—See the descriptions of these sequences in the section of this document on the Sequential model for more information.
- `Process Setup`—This sequence is a model callback that the Test UUTs and Single Pass execution entry points call from their setup step groups to give the client file an opportunity to execute any setup steps that must run only once during the execution of the process model. These setup steps are run from the controlling execution only, the test socket executions do not call this callback.
- `Process Cleanup`—This sequence is a model callback that the Test UUTs and Single Pass execution entry points call from their cleanup step groups to give the client file an opportunity to execute any cleanup steps that must run only once during the execution of the process model. These cleanup steps are run from the controlling execution only, the test socket executions do not call this callback.
- `Get Station Info`, `Get Report Options`, `Get Database Options`, and `Get Model Options`—See the descriptions of these sequences in the section of this document on the Sequential model for more information.

- **PreBatch**—Displays a dialog box in which the operator enters the Batch and UUT serial numbers. Override this in client file to change or replace this action. There is an example of overriding this callback in the `<TestStand>\Examples\Callbacks\BatchModel` directory.
- **PostBatch**—Displays a pass, fail, error, or terminated banner for each TestSocket and allows viewing of Batch and UUT reports. Override this in client file to change or replace this action. There is an example of overriding this callback in the `<TestStand>\Examples\Callbacks\BatchModel` directory.
- **PreBatchLoop**—The model calls this callback before looping on batches of UUTs. This callback is empty in the model file. Override this callback in the client file to perform an action before any batches of UUTs are tested.
- **PostBatchLoop**—The model calls this callback after looping on batches of UUTs. This callback is empty in the model file. Override this callback in the client file to perform an action after all batches of UUTs are tested.
- **BatchReport**—This sequence is a model callback that the `Test UUTs` and `Single Pass` execution entry points call to generate the contents of the batch report for the UUTs that ran in the last batch. You can override the `BatchReport` callback in the client file if you want to change its behavior entirely. The batch process model defines a batch report for a single group of UUTs as consisting of a header, an entry for each UUT result, and a footer. If you do not override the `BatchReport` callback, you can override the `ModifyBatchReportHeader`, `ModifyBatchReportEntry`, and `ModifyBatchReportFooter` model callbacks to customize the batch report.
- **ModifyBatchReportHeader**—This sequence is a model callback that the `BatchReport` model callback calls so that the client sequence file can modify the batch report header. `ModifyBatchReportHeader` receives the following parameters: the batch serial number, the tentative report header text, and the report options. The `ModifyBatchReportHeader` callback in the process model file is empty.
- **ModifyBatchReportEntry**—This sequence is a model callback that the `BatchReport` model callback calls so that the client sequence file can modify the entry for each test socket's UUT result in the batch report. Through subsequences, `BatchReport` calls `ModifyBatchReportEntry` for each test socket. `ModifyBatchReportEntry` receives the following parameters: the `TestSocket` data, the tentative report entry text, and the report options. The `ModifyBatchReportEntry` callback in the process model file is empty.

- `ModifyBatchReportFooter`—This sequence is a model callback that the `BatchReport` model callback calls so that the client sequence file can modify the batch report footer. `ModifyBatchReportFooter` receives the following parameters: the tentative report footer text, and the report options. The `ModifyBatchReportFooter` callback in the process model file is empty.

## Test UUTs

This entry point is the sequence that the controlling execution runs. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Call <code>Get Model Options</code> utility sequence.	Reads model options from disk. Calls <code>ModelOptions</code> model callback to allow client to modify options.
2	Call <code>PreBatchLoop</code> model callback.	Callback in model file is empty.
3	Call <code>Get Station Info</code> utility sequence.	Identifies the test station name and the current user.
4	Call <code>Get Report Options</code> utility sequence.	Reads report options from disk. Calls <code>ReportOptions</code> model callback to allow client to modify options.
5	Call <code>Get Database Options</code> utility sequence.	Reads database options from disk. Calls <code>DatabaseOptions</code> model callback to allow client to modify options.
6	Create and initialize test socket executions.	See the table for the <code>Test UUTs - Test Socket Entry Point</code> sequence for more information on what the test socket executions do.
7	Call <code>Run Batch Info Dialog</code> .	Calls this sequence in new thread and waits for it to initialize the dialog box code.
8	Wait for test sockets to get to <code>Initialize</code> synchronization point.	Calls the <code>ProcessTestSocketRequests</code> sequence to wait for and monitor test socket executions.
9	Call <code>Add TestSocket Threads to Batch</code> .	Adds test socket execution threads to the batch synchronization object. This allows the user's test sequence to use batch synchronization.

<b>Action Number</b>	<b>Description</b>	<b>Remarks</b>
10	Allow test socket executions that are waiting at Initialize synchronization point to continue.	Calls the Notify TestSocket Threads sequence.
11	Increment the Batch index.	—
12	Wait for test sockets to get to GetUUTSerialNumber synchronization point.	Calls the ProcessTestSocketRequests sequence to wait for and monitor test socket executions.
13	Call PreBatch model callback.	Obtains the Batch and UUT serial numbers from the operator.
14	If no more UUTs, set test socket data to tell test sockets to stop running their UUT loops.	Sets the ContinueTesting test socket data variable to false for all of the test sockets and marks them all as enabled so that they will be re-added to the batch and exit normally.
15	Remove disabled test socket threads from batch and re-add re-enabled test socket threads.	Disabled test sockets need to be removed from the batch so that they don't block the threads that are running.
16	Allow test socket executions that are waiting at GetUUTSerialNumber synchronization point to continue.	Calls the Notify TestSocket Threads sequence.
17	If no more UUTs, go to action 34.	—
18	Wait for test sockets to get to ReadyToRun synchronization point.	Calls the ProcessTestSocketRequests sequence to wait for and monitor test socket executions.
19	Determine the report file pathname for Batch and UUT report files.	Determines the report file path name to use if the report options are configured so that all UUT results for the model are written to the same file or if they are written to the same file as the batch reports.
20	Allow test socket executions that are waiting at ReadyToRun synchronization point to continue.	Calls the Notify TestSocket Threads sequence.
21	Wait for test sockets to get to ShowStatus synchronization point.	Calls the ProcessTestSocketRequests sequence to wait for and monitor test socket executions.

Action Number	Description	Remarks
22	Call <code>Add TestSocket Threads to Batch</code> .	The test socket executions remove themselves from the batch after executing <code>MainSequence</code> in order to cleanup the state of the batch in case the sequence was terminated or the user didn't match enters and exits properly. This is where the test socket execution threads are added again to the batch.
23	Call <code>PostBatch</code> model callback.	Displays a pass, fail, error, or terminate banner for all of the test sockets in the batch.
24	Allow test socket executions that are waiting at <code>ShowStatus</code> synchronization point to continue.	Calls the <code>Notify TestSocket Threads</code> sequence.
25	Wait for test sockets to get to <code>WriteReport</code> synchronization point.	Calls the <code>ProcessTestSocketRequests</code> sequence to wait for and monitor test socket executions.
26	Call <code>BatchReport</code> model callback.	Generates batch report for the last run of the batch of UUTs.
27	Write the batch report to disk.	Can append to an existing file or create a new file.
28	Allow test socket executions that are waiting at <code>WriteReport</code> synchronization point to continue.	Calls the <code>Notify TestSocket Threads</code> sequence passing <code>true</code> for the <code>ReleaseThreadsSequentially</code> parameter so that only one UUT report is written at a time in test socket index order.
29	Wait for test sockets to get to <code>UUTDone</code> synchronization point.	Calls the <code>ProcessTestSocketRequests</code> sequence to wait for and monitor test socket executions.
30	Tell status dialog box that report generation is done.	This is so that it can enable the view report buttons so that the user can view the reports from the dialog box.

Action Number	Description	Remarks
31	Wait for status dialog box.	If the status dialog box was displayed by the PostBatch callback (i.e. it wasn't overridden) then the sequence waits for the user to dismiss the dialog here if they have not already done so.
32	Allow test socket executions that are waiting at UUTDone synchronization point to continue.	Calls the Notify TestSocket Threads sequence.
33	Loop back to action 11.	—
34	Wait for test socket executions to complete.	—
35	Call PostBatchLoop model callback.	Callback in model file is empty.

## Test UUTs – Test Socket Entry Point

This entry point is the sequence that the test socket executions run. The controlling execution creates the test socket executions in its Test UUTs entry point sequence. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Sync with controlling execution for Initialize synchronization point	Calls SendControllerRequest and blocks until the controlling execution sets the test socket's notification.
2	Call PreUUTLoop model callback.	Callback in model file is empty.
3	Increment the UUT index.	—
4	Clear information from previous loop iteration.	Discard previous results and clear the report and failure stack.
5	Sync with controlling execution for GetUUTSerialNumber synchronization point	Calls SendControllerRequest and blocks until the controlling execution sets the test socket's notification.
6	Call PreUUT model callback.	Callback in model file is empty.
7	If no more UUTs, go to action 20.	—

<b>Action Number</b>	<b>Description</b>	<b>Remarks</b>
8	Sync with controlling execution for ReadyToRun synchronization point	Calls SendControllerRequest and blocks until the controlling execution sets the test socket's notification.
9	Determine the report file pathname.	—
10	Call MainSequence model callback.	MainSequence callback in client performs the tests on the UUT.
11	Remove the test socket's thread from batch synchronization.	This is necessary to cleanup the state of the batch incase MainSequence was terminated or the user didn't match enters and exits properly. The controlling execution re-adds the thread to batch synchronization before continuing past the next synchronization point. Disabled test sockets do not get re-added to the batch.
12	Sync with controlling execution for ShowStatus synchronization point	Calls SendControllerRequest and blocks until the controlling execution sets the test socket's notification.
13	Call PostUUT model callback.	Callback in model file is empty.
14	Call TestReport model callback.	Generates test report for the UUT.
15	Call LogToDatabase model callback.	Log test results to database for the UUT.
16	Sync with controlling execution for WriteReport synchronization point	Calls SendControllerRequest and blocks until the controlling execution sets the test socket's notification.
17	Write the UUT report to disk.	Can append to an existing file or create a new file.
18	Sync with controlling execution for UUTDone synchronization point	Calls SendControllerRequest and blocks until the controlling execution sets the test socket's notification.
19	Loop back to action 3.	—
20	Call PostUUTLoop model callback.	Callback in model file is empty.



## Single Pass

This entry point is the sequence that the controlling execution runs. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Call Get Model Options utility sequence.	Reads model options from disk. Calls ModelOptions model callback to allow client to modify options.
2	Call Get Station Info utility sequence.	Identifies the test station name and the current user.
3	Call Get Report Options utility sequence.	Reads report options from disk. Calls ReportOptions model callback to allow client to modify options.
4	Call Get Database Options utility sequence.	Reads database options from disk. Calls DatabaseOptions model callback to allow client to modify options.
5	Create and initialize test socket executions.	See the table for the Single Pass - Test Socket Entry Point sequence for more information on what the test socket executions do.
6	Wait for test sockets to get to ReadyToRun synchronization point.	Calls the ProcessTestSocketRequests sequence to wait for and monitor test socket executions.
7	Call Add TestSocket Threads to Batch.	Adds test socket execution threads to the batch synchronization object. This allows the user's test sequence to use batch synchronization.
8	Determine the report file pathname for Batch and UUT report files.	Determines the report file path name to use if the report options are configured so that all UUT results for the model are written to the same file or if they are written to the same file as the batch reports.
9	Allow test socket executions that are waiting at ReadyToRun synchronization point to continue.	Calls the Notify TestSocket Threads sequence.

Action Number	Description	Remarks
10	Wait for test sockets to get to PostMainSequence synchronization point.	Calls the ProcessTestSocketRequests sequence to wait for and monitor test socket executions.
11	Call Add TestSocket Threads to Batch.	The test socket executions remove themselves from the batch after executing MainSequence in order to cleanup the state of the batch in case the sequence was terminated or the user didn't match enters and exits properly. This is where the test socket execution threads are added again to the batch.
12	Allow test socket executions that are waiting at PostMainSequence synchronization point to continue.	Calls the Notify TestSocket Threads sequence.
13	Wait for test sockets to get to WriteReport synchronization point.	Calls the ProcessTestSocketRequests sequence to wait for and monitor test socket executions.
14	Call BatchReport model callback.	Generates batch report for the last run of the batch of UUTs.
15	Write the batch report to disk.	Can append to an existing file or create a new file.
16	Allow test socket executions that are waiting at WriteReport synchronization point to continue.	Calls the Notify TestSocket Threads sequence passing true for the ReleaseThreadsSequentially parameter so that only one UUT report is written at a time in test socket index order.
17	Wait for test sockets to get to UUTDone synchronization point.	Calls the ProcessTestSocketRequests sequence to wait for and monitor test socket executions.
18	Allow test socket executions that are waiting at UUTDone synchronization point to continue.	Calls the Notify TestSocket Threads sequence.
19	Wait for test socket executions to complete	—

## Single Pass – Test Socket Entry Point

This entry point is the sequence that the test socket executions run. The controlling execution creates the test socket executions in its `Single Pass` entry point sequence. The following table lists the most significant steps of this entry point:

Action Number	Description	Remarks
1	Sync with controlling execution for <code>ReadyToRun</code> synchronization point	Calls <code>SendControllerRequest</code> and blocks until the controlling execution sets the test socket's notification.
2	Determine the report file pathname.	—
3	Call <code>MainSequence</code> model callback.	<code>MainSequence</code> callback in client performs the tests on the UUT.
4	Remove the test socket's thread from batch synchronization.	This is necessary to allow other test socket threads to do batch synchronization without counting this thread anymore. The controlling execution re-adds the thread to batch synchronization before the thread runs the main sequence again. Disabled test sockets do not get re-added to the batch.
5	Sync with controlling execution for <code>PostMainSequence</code> synchronization point	Calls <code>SendControllerRequest</code> and blocks until the controlling execution sets the test socket's notification.
6	Call <code>TestReport</code> model callback.	Generates test report for the UUT.
7	Call <code>LogToDatabase</code> model callback.	Log test results to database for the UUT.
8	Sync with controlling execution for <code>WriteReport</code> synchronization point	Calls <code>SendControllerRequest</code> and blocks until the controlling execution sets the test socket's notification.
9	Write the UUT report to disk.	Can append to an existing file or create a new file.
10	Sync with controlling execution for <code>UUTDone</code> synchronization point	Calls <code>SendControllerRequest</code> and blocks until the controlling execution sets the test socket's notification.

# Support Files for the TestStand Process Models

Many sequences in the TestStand process model files call functions in DLLs and subsequences in other sequence files. TestStand installs these supporting files and the DLL source files in the same directory that it installs the process model sequence files.

The table below lists the files that TestStand installs for the TestStand process models in the <TestStand>\Components\NI\Models\TestStandModels directory.

File Name	Description
SequentialModel.seq, ParallelModel.seq, and BatchModel.seq	Entry point and model callback sequences for the TestStand process models.
reportgen_html.seq	Subsequences that add the header, result entries, and footer for a UUT into an HTML test report.
reportgen_txt.seq	Subsequences that add the header, result entries, and footer for a UUT into an ASCII text test report.
modelsupport2.dll	DLL containing C functions that the process model sequences call. Includes functions that display the Report Options and Model Options dialog boxes, read and write those options from disk, determine the report file pathname, obtain the UUT serial number from the operator, and display status banners.
modelsupport2.prj	LabWindows/CVI project that builds modelsupport2.dll.
modelsupport2.fpl	LabWindows/CVI function panels for the functions in modelsupport2.dll.
modelsupport2.h	C header file that contains declarations for the functions in modelsupport2.dll.
modelsupport2.lib	Import library in Visual C/C++ format for modelsupport2.dll.
modelpanels.uir	LabWindows/CVI user interface resource file containing panels that the functions in modelsupport.dll use.
modelpanels.h	C header file containing declarations for the panels in modelpanels.uir.
main.c	C source for utility functions.
banners.c	C source for functions that display status banners.

File Name	Description
report.c	C source for functions that display the Report Options dialog box, read and write the report options from disk, and determine the report file pathname.
uutdlg.c	C source for the function that obtains the UUT serial number from the operator.
c_report.c	C source for generating HTML and ASCII reports for the DLL option in the Report Options dialog box.
modeloptions.c	C source for the functions that display Model Options dialog box and read and write the model options from disk.
batchUUTdlg.c and parallelUUTdlg.c	C source for the functions that display the UUT identification dialogs for the Batch and Parallel process models. The files are part of modelsupport2.dll but the default process model, SequentialModel.seq, does not call them.

You can view the contents of the reportgen\_html.seq and reportgen\_txt.seq sequence files in the sequence editor. Both of these files are model sequence files and contain an empty ModifyReportEntry callback. Each file has a PutOneResultInReport sequence that calls ModifyReportEntry. The client sequence file can override the ModifyReportEntry callback. TestStand requires that all sequence files that contain direct calls to model callbacks must also contain a definition of the callback sequence and must be model files.

The TestStand process model sequence files also contain an empty ModifyReportEntry callback, even though no sequences in those files call ModifyReportEntry directly. They contain a ModifyReportEntry callback so that ModifyReportEntry appears in the Sequence File Callbacks dialog box for the client sequence file.

## Report Generation Functions and Sequences

To customize the report generation for your test station, create your own process model or modify the default TestStand process model files. However, do not modify the files in the <TestStand>\Components\NI\Models\TestStandModels directory because later installations of TestStand will overwrite your modifications. Instead, make a copy of the components and modify the copy. Using Windows Explorer, copy the TestStandModels folder from <TestStand>\Components\NI\Models\TestStandModels to <TestStand>\Components\User\Models\TestStandModels. The following tables list the process model sequences and C functions that generate the report and the locations of the files that contain them. The table

below lists the default process model sequences that generate the report header and footer.

**Table 1.** Sequence in <TestStand>\Components\NI\Models\TestStandModels\

Report Format	Report Header or Footer	
	Header	Footer
HTML	AddReportHeader sequence in ReportGen_Html.Seq	AddReportFooter sequence in ReportGen_Html.Seq
Text	AddReportHeader sequence in ReportGen_Txt.Seq	AddReportFooter sequence in ReportGen_Txt.Seq

The table below lists the default process model sequences and C functions that generate the report body for each step result.

**Table 2.** Sequence or C Function in <TestStand>\Components\NI\Models\TestStandModels\

Report Format	Report Body Generator Selected in the Report Options Dialog Box	
	Sequence	DLL
HTML	PutOneResultInReport sequence in ReportGen_Html.Seq	PutOneResultInReport_Html function in c_report.c in the ModelSupport2.prj LabWindows/CVI project
Text	PutOneResultInReport sequence in ReportGen_Txt.Seq	PutOneResultInReport_Txt function in c_report.c in the ModelSupport2.prj LabWindows/CVI project

You can also alter the report generation for each client sequence file that you run. To alter report generation, you override the report-generation model callbacks in the client sequence file. The table below lists the report-generation model callbacks. Refer to the Sequence File Callbacks section in Chapter 5, *Sequence Files*, in the *TestStand User Manual* for more information on overriding model callbacks.

Section of the Report to Alter	Model Callback Sequence to Override
Report Header	ModifyReportHeader
Report Footer	ModifyReportFooter

Section of the Report to Alter	Model Callback Sequence to Override
Each Step Result	ModifyReportEntry (TestStand does not call the ModifyReportEntry callback if you select the DLL report body generator in the Report Options dialog box.)
Entire Report	TestReport

In addition, each step in the sequence can add text to its corresponding result in the report. To make these additions, the step stores the text to add to the report in its Step.Result.ReportText property.