



Spotlight Test Driver File

Version 1.3
April 05, 2005

PathMATE Technical Notes

Pathfinder Solutions LLC
33 Commercial Drive, Suite 2
Foxboro, MA 02035 USA
www.PathfinderMDA.com
888-662-7284

Table Of Contents

- 1. Introduction..... 1
- 2. Test Driver Structure 1
 - Header.....1
 - Processes1
 - Test Script.....2
- 3. Test Commands 2
 - Test Script Commands3
 - Control Point Statements5
 - Synchronization Statements6
- 4. Legacy Driver Commands..... 7
- Appendix A..... 8

1. Introduction

The Spotlight Test Driver file (.stdf) is a structured extension to the Spotlight driver file (.usd). The Test Driver file format defines new commands that support using Spotlight as an automated testing tool.

The Test Driver file is defined using the international standard XML. The schema supports testing over multiple target tasks as well as a structured command format.

Spotlight will continue to support the loading of .usd files until further notice. The test command extensions will only be supported in the .stdf format. Files with extensions of both .usd or .stdf will be associated with Spotlight.exe for the purpose of automatically starting the Spotlight application.

The reader to refer to Technical Note: PathMATE Spotlight User Interface for additional information.

2. Test Driver Structure

- 1.1. Header
 - 1.1.1. Version number
 - 1.1.2. System Name
- 1.2. Processes
 - 1.2.1. Executable (optional)
 - 1.2.2. IP address (optional)
 - 1.2.3. Task
 - 1.2.3.1. Task identifier
 - 1.2.3.2. Port number
 - 1.2.3.3. Break directives (optional)
 - 1.2.3.4. Trace directives (optional)
 - 1.2.3.5. Animation directives (optional)
- 1.3. Test Script
 - 1.3.1. Commands

Header

The Header (2.1) contains the version and system information. It also contains the location of the model file.

Example

```
<spotlightDriver xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.PathfinderMDA/spotlightDriver.xsd"
name="chaseDC" version="5.01.002" modelFile="../../../analysis/chaseDC.mdl">
```

Processes

The Processes (2.2) section defines the tasks under test. If the test application is local it can be invoked by supplying an executable (2.2.1) in the

filename attribute. Local executables will be startup up and connected to automatically.

If the executable is on a remote machine, the IP address or machine name can be supplied in the machine attribute. Remote processes must be started outside of Spotlight, and before the driver file is loaded. Connection to the remote processes occurs when the driver file is loaded. Multiple processes can be specified, each with their own identifier, to support distributed deployment.

Processes made up of one or more tasks. An identifier is supplied for each task, this identifier will be used in the command section to route directives to the proper task. Each task has its own port number to support its connection to Spotlight. Break (2.2.2.3) and Trace (2.2.2.4) directives may be specified to pre-load breakpoints and trace control to a task.

Examples

```
<process filename=".\\project\\bin\\chaseDC.exe" id="localTask">
  <task name="SYS_TASK_ID_MAIN" port="5150" >
  </task>
  <task name="SYS_TASK_ID_TREE" port="5151" >
  </task>
</process>
<process machine="127.0.0.1" id="remoteTask">
  <task name="SYS_TASK_ID_MAIN" port="5150" >
  </task>
</process>
```

Test Script

The Test (2.3) section contains a test name and a sequence of test commands. The section is defined below. There is only one script section per driver file. The script section is optional. It can be used to define and run tests in batch mode, or to initialize the system into a particular state to start a debugging session.

The script commands are enclosed within the XML script tags. Script commands are executed sequentially within the command block.

```
<script>
  <!--Commands here -->
</script>
```

Script commands and syntax are defined in the next section.

3. Test Commands

Two types of command structures are defined. A command directive defines a single action e.g. go, pause or writelog. A command block defines a set of command directives that are executed when a task state occurs e.g. On Break. Command names will be case sensitive.

- Executable – defines a local executable file to be started.
- IPAddress – causes a connection to a task at the specified address.
- Break – sets a breakpoint for a task.
- Trace – sets a trace point for a task.

Test directives are specified with the pattern `<cmd cmd=directive, args></cmd>` where **directive** is one of the command directive listed below and *args* represents any arguments required for the directive.

References to model elements can be either by name or by unique id. Commands that require model element references provide the attributes to specify the elements by name, and also a *uuid* attribute to specify the Universal Unique Identifier given to the element by the UML editor. The UUID simplifies rename processing, while the element names provide human readable references.

Test Script Commands

1. cmd=setTask – specified the active task. Subsequent commands are directed to that task.

The task must be specified using the *process* and *task* attributes. If the task does not exist, an error is generated.

```
<cmd cmd="setTask" process="localTask" task="SYS_TASK_ID_MAIN" />
```

2. cmd=go – begins processing on the active task.

```
<cmd cmd="go" />
```

3. cmd=pause – pauses execution of the active task. Allows stimulus to be injected or data about the state of the system under test to be collected in subsequent test script commands.

```
<cmd cmd="pause" />
```

4. cmd=delay – pauses the test driver for a number of milliseconds before executing the next command. The application mode is unchanged, and may be running or paused, depending on previous commands.

```
<cmd cmd="delay" msec="1000" />
```

5. cmd=writelog – writes a string to the log. Useful for debugging and tracing test driver and system under test interaction.

The *message* attribute specifies the contents of the message to be placed in the file.

```
<cmd cmd="writelog" message="starting test" />
```

6. cmd=openfile – sets the log file of the test case. File will always be opened in APPEND mode, and create if it does not exist. Full and relative pathnames to the log file will be supported.

```
<cmd cmd="openfile" filename="chaseDCLog.log" />
```

7. cmd=closefile – closes the log file of the test case.

```
<cmd cmd="closefile" filename="chaseDCLog" />
```

8. cmd=reset – resets the task by deleting all instances and clearing the event queue. Processing can then begin fresh at initialization. This only effects models and does not reset the state of any realized domains.

```
<cmd cmd="reset" />
```

9. cmd=invoke – invokes a domain operation, passing the input parameters to the active task. The return value and output parameters will be stored in the log file, but are not available to the test driver as part of this release.

The domain operation may be specified using the domain name and operation name, or by using the *uuid* attribute. Parameter values are specified using the *property* XML elements within the command scope. *uuid* is also supported for parameters.

```
<cmd cmd="invoke" domain="EV" operation="Init" uuid="" >
  <property name="arg1" value="val" uuid="" />
</cmd>
```

10. cmd=dumpPopulation – Dumps information about the system under test to a file. How much information is stored depends on the scope parameter.

- System dumps all instance, association, and event queue data
- Domain dumps all instance data for a domain
- Class dumps all the instances of a class
- Event queue only dumps the event queue

Dumps will be stored in an XML file and will be able to be loaded using the Load Population command. Each instance will have a unique identifier given to it by software mechanisms. This identifier can be used to refer to other instances in the dump, such as across associations. The unique ID is in addition to any attributes marked with the "Identifier" property.

The *name* attribute is the fully qualified name of the class, specified by the Engine User's Guide, e.g., *system.domain.class*.

```
<cmd cmd="dumpPopulation" scope="system" name=""
filename="chaseDCPopulation.xml" />
```

11. `cmd=loadPopulation` – Reads in an XML file containing the initial population of a test case. Class instances, associations, and events are created in the system under test. The file can be created manually or by using the Dump Population command from a previous test case. This feature is very useful for setting up the initial conditions of the test case, especially error conditions.

```
<cmd cmd="loadPopulation" filename="chaseDCPopulation.xml" />
```

12. `cmd=generateEvent` – Generates an event to the specified instance, passing along the specified event parameters.

The *name* attribute is the fully qualified name of the event, specified by the Engine User's Guide, *system.domain.class.event*. The *destination* is the destination instance identifier for the event. Only objects from loaded instance populations can be specified here. Event parameter values are specified using the *property* XML elements within the command scope. *uuid* is supported for the event and its parameters.

```
<cmd cmd="generateEvent" name="chaseDC.EV.Dog.Woof" destination="Dog1"
uuid="" />
  <property name="arg1" value="val" uuid="" />
</cmd>
```

13. `cmd=openSeqChart` – Instructs Spotlight to create a sequence chart based on the interactions in the test case. The name of the sequence chart will be named using the *name* attribute and located in the model under the "Logical View". Note that if the script is executed multiple times, multiple diagrams with the same name will be created.

```
<cmd cmd="openSeqChart" name="seqChartName" />
```

14. `cmd=closeSeqChart` – stops sending trace information to the sequence chart in the UML editor.

```
<cmd cmd="closeSeqChart" />
```

15. `cmd=initialize` - Invokes the system or a domain's initialization code.

The scope attribute determines which initialization code is run, system or domain. For the domain scope, the domain is set through the *domain* or *uuid* attributes.

```
<cmd cmd="initialize" scope="domain" domain="EV" uuid="" />
```

Control Point Statements

Break and trace points are defined as control points, or just points within the test driver syntax. Breakpoints can be set under the task elements for initialization of breakpoints for an interactive Spotlight session, or within the script section to manage the control flow of the system under test. Syntax for control point statements uses the XML point element. The *break* and *trace* attributes are booleans that determine if the point is a breakpoint or tracepoint, respectively, if true.

The *category* defines the type of model element the control point applies to. It is an enumeration of

- class
- event
- service
- association
- pal

The type is specific to the category, given by the table 1.

Category	Applicable Types
class	create, delete, send, receive, transition, animation
event	cancel, receive, send
service	invoke
association	link, unlink
pal	statement

Table 1. Breakpoint Types by Category

The *element* attribute refers to the model element by its fully qualified name, as defined in the Engine Users Guide. The *uuid* is also used, to support rename within the driver file.

Break and trace points are supported within the <script> section of the driver file. A script section can be located under a task definition, primarily for setting task specific breakpoints.

Example control point statement.

```
<point break="true" trace="false" category="event" type="cancel"
element="chaseDC.Dog.Woof" uuid="" />
```

Synchronization Statements

Control is synchronized between the application and the test driver through breakpoints or when the system goes idle. These conditions are detected by the application and sent to Spotlight and the test script. When detected, the application is paused and control returns to the test script for continued processing. Synchronization statement, onbreak and onidle, are the points at which the test driver waits for the models to reach that state, thus, synchronizing the test script with the executing model. These statements can contain commands within the block, but since all commands within the script section are executed sequentially, the commands can also be located after the synchronization statements.

The idle condition is detected by the application when all events are processed and no other inputs can come in. In multi-task and multi-process modes, inputs may come in from other tasks, so idle mode is only applicable in single-task, single process deployment, and mainly for domain-level dynamic verification.

1. **onbreak** – the test driver will be notified of breakpoints triggered in the system under test. Execution is paused in the application until the go command is executed. The onbreak statement can contain commands within it.

```
<onbreak >
    <cmd />
    <point break="false" trace="false" category="class" type="transition"
element="chaseDC.Dog" uuid="" />
</onbreak>
```

2. **onidle** – triggered when the main event loop determines there is nothing left to do until an external event comes in. This will not be triggered if there are periodic services registered or if the system under test is multi-task or multi-process. The onidle statement can contain commands within it.

```
<!-- wait for idle condition before proceeding -->
<onidle >
    <cmd />
    <cmd cmd="dumpPopulation" scope="system | domain | class | events"
filename="chaseDCPopulation.xml" />
</onidle>
```

4. Legacy Driver Commands

The following commands are supported for the Spotlight driver file (.usd) format. Driver files in both formats will be supported on load, but saving will only support the new format.

- EXECUTABLE <executable_file_name>
- CONNECT
- BREAK <category> <object_name> <type>
- TRACE <category> <object_name> <type>
- DUMP
- GO
- INVOKE

Appendix A

The following file shows a sample test driver file for the chaseDC test case, with multiple processors and multiple tasks.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Driver file for PathMATE Spotlight (http://www.pathmate.com)-->

<spotlightDriver xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.PathfinderMDA/spotlightDriver.xsd"
name="chaseDC" modelFile="..\analysis\chaseDC.mdl" version="5.01.002" >

  <process filename="..\project\bin\chaseDC.exe" id="localTask">

    <task name="SYS_TASK_ID_MAIN" port="5150" >

      <script>

        <!-- not supported yet - class breakpoints can also apply to instances -->

        <point break="true" trace="false" category="class" type="create"
element="chaseDC.Dog" uuid="" />

        <point break="true" trace="false" category="class" type="delete"
element="chaseDC.Dog" uuid="" />

        <point break="true" trace="false" category="class" type="send"
element="chaseDC.Dog" uuid="" />

        <point break="true" trace="false" category="class" type="receive"
element="chaseDC.Dog" uuid="" />

        <point break="true" trace="false" category="class" type="transition"
element="chaseDC.Dog" uuid="" />

        <point break="true" trace="false" category="class" type="animation"
element="chaseDC.Dog" uuid="" />

        <!-- not supported yet - service invocation on the class -->

        <point break="true" trace="false" category="class" type="invoke"
element="chaseDC.Dog.eatFood" uuid="" />

      </script>

    </task>

    <task name="SYS_TASK_ID_TREE" port="5151" >

      <script>

        <point break="true" trace="false" category="event" type="cancel"
element="chaseDC.Dog.Woof" uuid="" />

        <point break="true" trace="false" category="event" type="receive"
element="chaseDC.Dog.Woof" uuid="" />

        <point break="true" trace="false" category="event" type="send"
element="chaseDC.Dog.Woof" uuid="" />

      </script>

    </task>

  </process>

  <process machine="127.0.0.1" id="remoteTask">

    <task name="SYS_TASK_ID_MAIN" port="5150" >

      <script>
```

```

        <point break="true" trace="false" category="service" type="invoke"
element="chaseDC.Init" uuid="" />

        <point break="true" trace="false" category="association" type="link"
element="chaseDC.A1" uuid="" />

        <point break="true" trace="false" category="association" type="unlink"
element="chaseDC.A1" uuid="" />

    </script>

</task>

<task name="SYS_TASK_ID_TREE" port="5151" >

    <script>

        <!-- Action language lines -->

        <point break="true" trace="false" category="pal" type="statement"
filename="[action_file]" line="[stmt.lineNumber]" />

    </script>

</task>

</process>

<!--

Breakpoints can also be used within the script section to enable and disable
breakpoints on the fly

-->

<script>

    <cmd cmd="setTask" process="localTask" task="SYS_TASK_ID_MAIN" />
    <cmd cmd="go" />
    <cmd cmd="pause" />
    <cmd cmd="delay" msec="1000" />
    <cmd cmd="openfile" filename="chaseDCLog.log" />
    <cmd cmd="writelog" message="starting test" />
    <cmd cmd="closefile" filename="chaseDCLog" />
    <cmd cmd="reset" />

    <!-- open under LogicalView -->
    <cmd cmd="openSeqChart" name="seqChartName" />
    <cmd cmd="closeSeqChart" />

    <cmd cmd="invoke" domain="EV" operation="Init" uuid="" >
        <property name="arg1" value="val" uuid="" />
    </cmd>

    <!-- not supported yet -->
    <cmd cmd="initialize" scope="domain | system" domain=" EV" uuid="" />

    <!-- not supported yet without a way to identify a destination -->
    <cmd cmd="generate" event="chaseDC.EV.Dog.Woof" destination="" >

```

```
<property name="arg1" value="val" uuid="" />
</cmd>

<cmd cmd="loadPopulation" filename="chaseDCPopulation.xml" />
<cmd cmd="dumpPopulation" scope="system | domain | class | events"
filename="chaseDCPopulation.xml" />

<!-- wait for idle condition before proceeding. No indication here the source of
the breakpoint -->
<onbreak >
  <cmd />
  <point break="false" trace="false" category="class" type="transition"
element="chaseDC.Dog" uuid="" />
</onbreak>

<!-- wait for idle condition before proceeding -->
<onidle >
  <cmd />
  <cmd cmd="dumpPopulation" scope="system | domain | class | events"
filename="chaseDCPopulation.xml" />
</onidle>
</script>
</spotlightDriver>
```

This example shows how to initialize, test, and collect results for a single process, single task deployment.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Sample Instance Population

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Instance report generated by PathMATE Spotlight
(http://www.pathmate.com)-->
<system xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.PathfinderMDA/idp.xsd"
name="chaseDC_SYS_TASK_ID_MAIN">
  <population>
    <package name="Events">
      <object class="Cat" id="00940F40" state="Safe" >
        <property name="Name" value="Fluffy" />
        <link name="A1" role="" to="00940E48"/>
      </object>
      <object class="Dog" id="00940E48" state="Resting" >
        <property name="CatsChased" value="1" />
        <property name="Name" value="Fido" />
        <link name="A1" role="" to="00940F40"/>
      </object>
    </package>
  </population>
</system>
```