

1

Revision History

2

Date	Rev	Init	Change
02/29/08	D0.18	jo	Modified <i>bypass_stmt</i> - removed <i>portindex</i> as option in and updated semantics if <i>VAR_NAME</i> is type integer. Modified <i>ExitPort</i> statement to remove <i>portindex</i> (used as a bypass option). Ordinal index order (starting with 0) is now used instead. <i>PORTLABEL</i> is now required, not optional.
3/24/08	D0.18	jo	Modified <i>TestType</i> , <i>FlowType</i> syntax. Changed “UseDefaults” to “Inherit <i>TestBase</i> ”. Clarified semantics if “Inherit” statement is absent.
3/27/08	D0.18	jo	Added revision table to keep track of changes.
5/15/08	D0.19	jo	Added <i>softbin</i> attributes and <i>bin</i> property access syntax (per Ernie’s recent writeups).
5/22/08	D0.20	jo	Added (Const) attribute to <i>var_type</i> ; Removed <i>Operator</i> attribute from <i>var_elements_stmt</i> (may be added back if/when we determine it’s needed). Updated <i>TestBase</i> definition per discussions with Doug and Ernie. Removed <i>ReturnState</i> keyword as option in <i>exit_port_stmt</i> . Modified <i>Return</i> keyword to remove optional (<i>integer_expr</i>) token. <i>ExecResult</i> from <i>TestBase</i> (or derivatives) is used instead.
5/23/08	D0.21	jo	Removed <i>Return</i> keyword from <i>exit_port_stmt</i> . Moved <i>Stop</i> keyword from <i>exit_port_stmt</i> to <i>action_stmt</i> . In <i>PassActions/FailActions</i> of <i>test_elements_stmt</i> and <i>flow_elements_stmt</i> , and in <i>TestBase</i> , <i>TestType</i> , and <i>FlowType</i> syntax, replaced <i>exit_port_stmt</i> with <i>action_stmt</i> .
7/02/08	D0.22	jo	Added optional <i>TestExec</i> statement to <i>TestBase</i> syntax. <i>STIL .4</i> definition definition of <i>TestBase</i> will not include <i>TestExec</i> . Added <i>SetBinStop</i> alternative to <i>action</i> statement definition. Changed initial value of <i>TestBase Bin</i> parameter from undefined to <i>NoBin</i> . For component values of <i>NoBin</i> , see section 3.2.5 of Ernie’s bin document. In “Default Flow Node” definition, replaced <i><exec_object_name></i> with <i>EXEC_OBJECT_NAME</i> to adhere to the <i>STIL BNF</i> notation rules. Still need to agree on the alternatives for binning syntax shown in Fig. 7 of that same doc. What does <i>Stop</i> or <i>SetBinStop</i> mean from the actions of a <i>Test</i> , as opposed to the actions of a <i>FlowNode</i> ?
8/14/08	D0.23	jo	<ul style="list-style-type: none"> • In <i>initial_value_stmt</i>, updated syntax so that <i>test_element_stmt</i> and <i>flow_element_stmt</i> occur after <i>InitialValue</i> keyword, to bring <i>TestType</i> and <i>FlowType</i> initial value statements in line with those of all other types. Note the inclusion of open/close braces ({ }) to enclose those statements. • Added keyword Port preceding <i>PORTLABEL</i> in <i>ExitPorts</i> statement; removed colon (:) after <i>PORTLABEL</i> in <i>exit_port_stmt</i>. Addition of <i>Port</i> keyword makes the colon unnecessary.

		<ul style="list-style-type: none"> • Modified InitialValue syntax (for variables) to include array initialization (using a comma-separated list). • Added keyword Return <i>integer_expr</i> to <i>exit_port_actions</i>. This causes an immediate return to the caller (either a Test or a Flow). The integer return value sets the FlowNode variable Result. Result is a predefined keyword which can be used only in the FlowNode PostActions or ExitPort clauses. • Updated semantics of Stop statement. Stop now returns immediately to EntryPoint initiator, with appropriate pass/fail result. See text for more details. • Added the option (<i>flownode_stmt</i>)* to <i>test_elements_stmt</i>. This will allow tests to specify a sequence of flownodes for each instance created, as well as in the type definition. • Allow a TestType to derive from a FlowType. This is to allow Flows to be turned into Tests (and thus, presumably, hiding the internals of that flow from graphical Flow Editor tools). • Removed TestDefaults block. Definitions of TestBase and the default flow type stil_dot_4_default need no container; the STIL.4 default flownode definition must occur outside the scope of a Flow or Test. If more than one default definition occurs, the last one listed is used. • Modified the syntax of FlowNode to include the SetResult statement, the introduction of the FlowNode variable Result, and the keyword CurrentExec, which is an alias for the actual Test or Flow named in the TestExec statement. CurrentExec can be used ONLY in the SetResult, the <i>exit_port_expr</i>, or <i>exit_port_stmt</i> statements. • Change STIL 1.0 Flow extension date from 2007 to 2008. • Added angle brackets (< >) to syntax notation for <i>softbin_definition</i>. Expanded SetBin and SetBinStop syntax to include BIN_VAR_NAME, as well as explicit bin name. Added <i>bin_expr</i> notation. • Modified FlowType syntax to allow FLOW_TYPE_NAME to be optional. This allows an unnamed FlowType block which can be used as a default FlowType. See section 6.9 of IEEE_1450_1999_0 (dot0) for additional information on domain names. • Updated STIL.4 default definitions of TestBase, FlowNode, and FlowType. • Change Test and Flow instantiation syntax from Flows { } / Tests { } to Flow FLOWTYPE FLOWNAME; / Test TEST_TYPE TEST_NAME; • Added clarifications about integer values for Pass, Fail, True, and False.

3 Syntax summary for 1450.4

```

4   stil_block_type =
5       < Category
6         | DCLevels
7         | DCSequence
8         | DCSets
9         | Environment
10        | Flow
11        | MacroDefs
12        | Pattern
13        | PatternBurst
14        | PatternExec
15        | Procedures
16        | ScanStructures
17        | Selector
18        | SignalGroups
19        | Signals
20        | Test
21        | Timing >
22
23   real_var_type =
24       < Capacitance // F (Farads)
25         | Compound // combinations of types
26         | Current // A (Amperes)
27         | Double
28         | Frequency // Hz (Herz)
29         | Gain // db (Decibels)
30         | Inductance // H (Henries)
31         | Length // m (Meters)
32         | Power // W (Watts)
33         | Real
34         | Resistance // Ohm (Ohms)
35         | Temperature // Cel (Degrees Celsius)
36         | Time // s (Seconds)
37         | Voltage // V (Volts) >
38
39   var_type =
40       // Boolean values: True/False or Pass/Fail
41       (Const) < Boolean | Integer | SignalRef | SignalVariable | String | real_var_type | stil_block_type >
42
43   initial_value_elements =
44       < integer_expr // allow if var of type integer_expr
45         | sig_ref_expr // allow if var of type sig_ref_expr
46         | < True | False > // allow if var of type Boolean – from dot1, 0 = TRUE, 1 = FALSE.
47         | string // allow if var of type String
48         | wfc_string // allow if var of type SignalVariable
49         | real_expr // allow if var of type real_var_type
50         | BLOCK_NAME // allow if var of type stil_block_type - assign from predefined block
51         | { test_elements_stmt+ } // allow if var of type TestType
52         | { flow_elements_stmt+ } // allow if var of type FlowType
53         >
54
55
56
```

```

57 // In the second form of the statement shown below (used for array initialization), if more than one
58 // initial_val_elements statement is expression is present, a comma is used to separate the list
59 initial_value_stmt = InitialValue initial_val_elements | InitialValue initial_val_elements+ (,)
60
61 // Use either the explicit softbin name, or use a softbin variable name (variable of type Bin).
62 bin_expr = (BIN_AXIS.)SOFTBIN_NAME | BIN_VAR_NAME
63
64 action =
65     < VAR_NAME = expr; // Scalar variables
66     | VAR_NAME[INDEX] = expr; // Array variables
67     // Only one element of each bin axis can be active at any one time. For 3 separate axes, you can
68     // have 3 separate softbins (one from each axis) set – the binmap handles the various intersections
69     // when the part is binned.
70     | SetBin bin_expr;
71     // Stop – Immediately stop execution of current block (test, flow, or flow node), and jump directly to the
72     // initiating EntryPoint (On* condition) for end-of-test processing. If Stop is issued from a FlowNode, the
73     // result returned to the EntryPoint executor is the current value of the FlowNode variable Result. If Stop
74     // is issued from a Flow or Test, the result returned to the EntryPoint executor is the current value of the
75     // execResult field of the Test or Flow from which the Stop is issued (i.e., FLOW_NAME.execResult or
76     // TEST_NAME.execResult)
77     | Stop; // terminate the initiating On* condition.
78     // SetBinStop is equivalent, semantically, to:
79     // If (<bin_name> != NoBin) { SetBin <bin_name>; Stop;} Else { // No Action }
80     | SetBinStop bin_expr;
81
82 action_stmt =
83     < If boolean_expr | Else If boolean_expr | Else { // Usual rules for If/Else If/Else apply
84         ( action )*
85     }
86     | ( action )* >
87
88 bypass_actions =
89     <
90     // For FlowNodes, Flows, and Tests
91     // Skip only Test execution or FlowNode sequence execution, resume at entry to PostActions Block
92     // Normal processing continues from there. Execute PostActions, and PassActions/FailActions (as
93     // determined by FailFlag of Test or Flow), or ExitPort selection Actions (as determined by flownode
94     // exit port selector). Selection of Pass/Fail path (for Tests and Flows) is controlled by FailFlag (which
95     // can be forced to a desired state prior to Bypass statement). Selection of ExitPort path (for flownodes)
96     // is also controlled by boolean expressions – typically, the return status of the test executed by the
97     // FlowNode. This return state (or value of any other boolean expression) can be forced to a desired state
98     // prior to the Bypass statement.
99     Bypass;
100
101     // For Flows and Tests only.
102     // Skip Test and PostActions, resume execution at entry to PassActions/FailActions.
103     // If Pass or Fail specified, follow that path. Otherwise, VAR_NAME is a string variable which specifies
104     // either PASS or FAIL. If using VAR_NAME, and it's an empty string, no bypass action occurs
105     // If SkipActions specified, don't execute PassActions/FailActions (equivalent to a return,
106     // since there's no branching from a Test or Flow)
107     | Bypass (GoTo <Pass | Fail | VAR_NAME > ( SkipActions ) );
108
109     // For FlowNodes only.
110     // Skip Test and PostActions, resume execution at entry to specified ExitPort.
111     // If SkipActions specified, don't execute don't execute the exit port actions.
112     // PORTLABEL is a string specifying a port label. VAR_NAME is either a string variable or an integer variable.

```

```

113 // If a string variable, it specifies the bypass exit port by label. If using VAR_NAME, and it's an empty string,
114 // no bypass action occurs. If an integer variable, it specifies the bypass exit port by index (based on the
115 // ordinal order of the exit ports, from top of list to bottom, with the first index being 0.
116 | Bypass (GoTo < PORTLABEL | VAR_NAME > ( SkipActions ) );
117
118 bypass_stmt =
119 < bypass_actions | If boolean_expr { bypass_actions } >
120
121 exit_port_stmt =
122 < action_stmt
123 | Next (FLOW_NODE_NAME);
124 | Return < Result | integer_expr >; // terminate the Test or Flow. Return value of
125 // Result or integer_expr to calling Flow or Test.
126 // Value of Result is set by FlowNode SetResult, and
127 // statement, and can be used only in FlowNode
128 // PostAction or ExitPort clauses.
129 >
130
131 func_stmt =
132 < FUNC_NAME ; // name of a Function from FunctionDefs block
133 | FUNC_NAME { // name of a Function from FunctionDefs block
134 (VAR_NAME = expr);*
135 (VAR_NAME = [ expr (expr)+];)* // For arrays (?)
136 } // end func_stmt
137 >
138
139 test_elements_stmt =
140 < VAR_NAME = expr;
141 | VAR_NAME = [ expr (expr)+]; // For arrays (?)
142 | PreActions { ( < action_stmt | bypass_stmt )* } )
143 | FuncExec func_stmt | FuncExec { ( func_stmt )* } | ( flownode_stmt )*
144 | PostActions { ( action_stmt )* }
145 | PassActions { ( action_stmt )* }
146 | FailActions { ( action_stmt )* }
147 >
148
149 test_instance_stmt =
150 // create a named Test from a TestType, using the default elements for the TestType
151 TEST_TYPE TEST_INSTANCE_NAME;
152
153 // create a named Test from a TestType, overriding some or all of the default elements of the TestType.
154 // Any element specified in the instantiation (i.e., PreActions, FailActions) completely replaces that
155 // element as specified in the type definition.
156 | TEST_TYPE TEST_INSTANCE_NAME { ( test_elements_stmt )* }
157
158 run_result_item = integer | integer:integer
159
160 run_result_list = run_result_item \ run_result_list, run_result_item
161
162 // i.e., "Result 0;" or "Result 0,1,5,7;" or "Result -3:-1,1,4:6;"
163 // Note that the boolean_expr can use the variable Result (i.e., Result == 0 or Result > 2
164 exit_port_expr = Result run_result_list; | boolean_expr
165
166
167
168

```

```

169 flownode_stmt =
170     FlowNode (NODE_NAME) {
171         ( PreActions { ( < action_stmt | bypass_stmt )* } )
172         ( TestExec execute_stmt | TestExec { ( execute_stmt )* } )
173         // If the SetResult clause is not specified, the value of FlowNode variable Result defaults
174         // to the execResult of the most recently executed test or flow (i.e. TEST_NAME.execResult
175         // or FLOW_NAME.execResult). Within the SetResult statement or the exit_port_expr, the
176         // keyword CurrentExecResult can be used as a synonym for the execResult of the actual
177         // test or flow name (i.e., CurrentExec refers to TEST_NAME or FLOW_NAME)
178         ( SetResult < integer_expr | CurrentExec.execResult>;)
179         ( PostActions { ( action_stmt)* } )
180         ( ExitPorts {
181             ( Port PORTLABEL exit_port_expr { ( exit_port_stmt)* } ) *
182         } ) // end ExitPorts
183     } // end FlowNode
184
185
186 flow_elements_stmt =
187     < VAR_NAME = expr;
188     | VAR_NAME = [ expr (expr)+]; // For arrays (?)
189     | PreActions { ( < action_stmt | bypass_stmt )* } )
190     | Title STRING ;
191     | ( flownode_stmt )*
192     | PostActions { ( action_stmt )* } )
193     | PassActions { ( action_stmt )* } // end PassActions
194     | FailActions { ( action_stmt )* } // end FailActions
195     >
196
197 flow_instance_stmt =
198     // create a named Flow from a FlowType, using the default elements for the FlowType
199     < (FLOW_TYPE) FLOW_INSTANCE_NAME ; // if FLOW_TYPE is not specified, then the
200     // STIL.4 default FLOW_TYPE is used
201
202     // create a named Flow from a FlowType, overriding some or all of the default elements of the FlowType.
203     // Any element specified in the instantiation (i.e., PreActions, flownodes, PassActions) completely
204     // replaces that element as specified in the type definition.
205     | (FLOW_TYPE) FLOW_INSTANCE_NAME { ( flow_elements_stmt )* } > // if FLOW_TYPE is not specified, then the
206     // STIL.4 default FlowType (the unnamed
207     // FlowType) is used.
208
209 test_execute_stmt =
210     < TEST_NAME ; // execute a named Test
211     | TEST_TYPE ; // create and execute a temporary inline Test (named _INLINE_TEST)
212     // from TestType, using the type's default element value.
213
214     | TEST_TYPE { ( test_elements_stmt )* } > // Create and execute a temporary inline Test
215     // (named _INLINE_TEST) from TestType,
216     // overriding the type's default element values
217
218 flow_execute_stmt =
219     < FLOW_NAME ; // execute a named Flow
220     | FLOW_TYPE ; // create and execute a temporary inline Flow (named _INLINE_FLOW)
221     // from FlowType, using the type's default element values
222     | FLOW_TYPE { ( flow_elements_stmt )* } > // Create and execute an a temporary inline Flow
223     // (named _INLINE_FLOW) from FlowType,
224     // overriding the type's default element values

```

```

225
226 execute_stmt =< test_execute_stmt | flow_execute_stmt >
227
228 var_elements_stmt =
229     // Do we need an Operator attribute? Intended to identify variables whose values are set by a
230     // query to an operator and the operator's response (i.e., lot ID)
231     < var_type VAR_NAME ;
232     | var_type VAR_NAME {
233         ( Length integer; ) // array of var_type HOW TO INDEX????
234         ( initial_value_stmt )
235     }
236
237 =====
238 STIL 1.0 {
239     ( Flow 2008; )+
240 }
241
242 =====
243 Variables ( VAR_DOMAIN ) { // extensions to 1450.1
244     var_elements_stmt*
245 }
246
247 =====
248 FunctionsDefs {
249     ( FUNCTION_NAME {
250         ( Parameters {
251             ( var_type VAR_NAME (< In | Out | InOut > ; )*)
252             ( var_type VAR_NAME (< In | Out | InOut > ) { initial_value_stmt } )*)
253         } ) // end Parameters
254     })* // end function_name
255 } // end FunctionDefs
256
257 =====
258 softbin_attribute =
259 <
260     Color <String>; | // Hex, RGB, or name
261     Number <Unsigned Integer>; |
262     Retest <Unsigned Integer>; |
263     Terse <String>; |
264     Verbose <String>; |
265     WafermapChar <simple_character>; // From P1450.1999 BNF
266 >
267
268 softbin_definition =
269 Bin < SOFTBIN_NAME ;
270     | SOFTBIN_NAME { (softbin_attribute)* }
271 >
272
273 BinDefs (BIN_DEF_NAME) { // soft bin definitions
274     Pass { // Increment Pass-bin if ReturnState is Pass
275         ( Color <String>; ) // Contains default color for all Pass bins, "green" if unspecified
276         (softbin_definition)* |
277         ( Axis BIN_AXIS_NAME {
278             (softbin_definition)*
279         } )*
280     } // end Pass

```

```

281     Fail {           // Increment Fail-bin if ReturnState is Fail
282         (Color <String>); // Contains default color for all Fail bins, "red" if unspecified
283         (softbin_definition)* |
284         ( Axis BIN_AXIS_NAME {
285             (softbin_definition)*
286         })*
287     } // end Fail
288 }
289
290 // In the second form of the statement shown below, if more than one BIN_AXIS_NAME.SOFTBIN_NAME
291 // expression is present, a comma is used to separate the list
292 bin_map_stmt =
293     Map SOFTBIN_NAME integer; |
294     Map [(BIN_AXIS_NAME.SOFTBIN_NAME)+ (, )] integer;
295
296 =====
297 BinMap BIN_MAP_NAME { // soft to hard bin mapping
298     (bin_map_stmt)*
299 }
300


---


301 Bin Property access syntax
302 counter_reset_event =
303 <
304     OnLoad |
305     OnLotStart |
306     OnRetest |
307     OnSiteStart | // What's the difference between OnSiteStart
308     OnStart | // and OnStart ?
309     OnWaferStart
310 >
311
312 bin_property =
313 <
314     Color | // String
315     ContinueOnFail | // Boolean
316     counter.counter_reset_event | // Unsigned
317     Enabled | // Boolean
318     Index | // Unsigned
319     isSet.counter_reset_event | // Boolean
320     Name | // String
321     Number | // Integer
322     retest.(current|Original) | // Unsigned
323     Terse | // String
324     Verbose | // String
325     WafermapChar // Character
326 >
327
328
329 group = < Pass|Fail >
330
331 (BINDEFS_NAME.) group[Unsigned| SOFTBIN_NAME ].bin_property |
332 (BINDEFS_NAME.) group[Unsigned| AXIS_NAME ][Unsigned| SOFTBIN_NAME ].bin_property
333
334
335
336

```

```

337 =====
338 TestBase {
339     (Parameters {
340         ((<In | Out | InOut>) var_type VAR_NAME;)*
341         ((<In | Out | InOut>) var_type VAR_NAME;){ initial_value_stmt })*
342     }) // end Parameters
343     (Variables { var_elements_stmt* } | Variables VAR_DOMAIN)*
344     (PreActions { (< action_stmt | bypass_stmt )* })
345     // Allows implementers to include a dummy TestExec in TestBase. The dummy TestExec can,
346     // for instance, display a warning or error that an actual Test did not include its own TestExec or
347     // FuncExec, or that an actual Flow did not include at least one FlowNode (which can be empty).
348     (TestExec;)
349     (PostActions { ( action_stmt)* })
350     (PassActions { ( action_stmt)* }) // end PassActions
351     (FailActions { ( action_stmt)* }) // end FailActions
352 } // end TestBase
353 =====
354 TestType TEST_TYPE_NAME {
355     // If not inheriting from a previously-defined TEST_TYPE_NAME (either vendor-defined or user-defined),
356     // a TestType will inherit from TestBase. The “Inherit TestBase” statement is not required, but is
357     // recommended. If no Inherit statement is present, the behavior is equivalent to “Inherit TestBase”
358     (Inherit TestBase; | Inherit TEST_TYPE_NAME ; | Inherit FLOW_TYPE_NAME ;)
359     (Parameters {
360         ((<In | Out | InOut>) var_type VAR_NAME;)*
361         ((<In | Out | InOut>) var_type VAR_NAME;){ initial_value_stmt })*
362     }) // end Parameters
363     (Variables { var_elements_stmt* } | Variables VAR_DOMAIN)*
364     (PreActions { (< action_stmt | bypass_stmt )* })
365     (TestExec; | FuncExec func_stmt | FuncExec { ( func_stmt )* }
366     | flownode_stmt+ ) // Allows inline instantiation of a flow in a Test. That flow can't be reused.
367     (PostActions { ( action_stmt)* })
368     (PassActions { ( action_stmt)* }) // end PassActions
369     (FailActions { ( action_stmt)* }) // end FailActions
370 } // end TestType
371 =====
372 (Test test_instance_stmt )* // Instantiate named Test from TestType
373 =====
374 // A FlowType can be named or unnamed. Only one unnamed FlowType is allowed. If an unnamed
375 // FlowType is not provided by the user, then a default unnamed FlowType must be provided by the
376 // environment or toolset. The contents of this default unnamed FlowType are specified below.
377 FlowType (FLOW_TYPE_NAME) {
378     // If not inheriting from a previously-defined FLOW_TYPE_NAME (either vendor-defined or user-defined),
379     // a FlowType will inherit from TestBase. The “Inherit TestBase” statement is not required, but is
380     // recommended. If no Inherit statement is present, the behavior is equivalent to “Inherit TestBase”
381     (Inherit TestBase; | Inherit FLOW_TYPE_NAME ; )
382     (Parameters {
383         ((<In | Out | InOut>) var_type VAR_NAME;)*
384         ((<In | Out | InOut>) var_type VAR_NAME;){ initial_value_stmt })*
385     }) // end Parameters
386     (Variables { var_elements_stmt* } | Variables VAR_DOMAIN)*
387     (PreActions { (< action_stmt | bypass_stmt )* })
388     (Title STRING ;)
389     (flownode_stmt)*

```

```

393     ( PostActions { ( post_action_stmt)* } )
394     ( PassActions { ( action_stmt)* } ) // end PassActions
395     ( FailActions { ( action_stmt)* } ) // end FailActions
396 } // end FlowType
397
398 =====
399 ( Flow flow_instance_stmt ) * // Instantiate named Flow from FlowType
400
401 =====
402 TestProgram TEST_PROGRAM_NAME {
403     DUTType "DUT NAME STRING";
404     SocketDef "SOCKET NAME STRING";
405     // Variables are global to test program and below; local to site (a copy for each site)
406     ( Variables { var_elements_stmt* } | Variables VAR_DOMAIN ) *
407     ( BinDefs BIN_DEF_NAME; )
408     ( BinMap BIN_MAP_NAME; )
409
410     EntryPoints {
411         ( OnException execute_stmt )
412         ( OnFinish execute_stmt )
413         ( OnLoad execute_stmt )
414         ( OnLotEnd execute_stmt )
415         ( OnLotStart execute_stmt )
416         ( OnMultiSiteDisable execute_stmt )
417         ( OnMultiSiteEnable execute_stmt )
418         ( OnPatternLoad execute_stmt )
419         ( OnPowerDown execute_stmt )
420         ( OnReset execute_stmt )
421         ( OnSiteEnd execute_stmt )
422         ( OnSiteStart execute_stmt )
423         ( OnStart execute_stmt )
424         ( OnUnload execute_stmt )
425         ( OnWaferEnd execute_stmt )
426         ( OnWaferStart execute_stmt )
427     } // End Entry Points
428 } // end TestProgram
429
430 =====

```

For Multi-Site

```

431
432
433 // PROPOSED – How do we want to specify different test programs for different sites?
434 // Is this necessary?
435 ( SiteDefs (SITE_DEF_NAME) {
436
437     // Variables are global across all sites (one copy across all sites)
438     (Variables { var_elements_stmt* } | Variables VAR_DOMAIN)*
439
440     ( NumSites integer_expr ; )
441
442     // One statement per site. Only one can be default. If there are more sites than statements, then all
443     // unspecified sites will use the default program.
444     // It has been stated that we'd like to accommodate a scenario in which the same test program is
445     // running on all sites, but a different flow is running on each site. Using the syntax proposed here,
446     // it is possible do this by specifying different test programs for each site. The test programs are
447     // identical EXCEPT that the Flow specified by OnStart is different.
448     ( (Default) SITE_NAME TestProgram TEST_PROGRAM_NAME;)+
449 })
450
451 SPEC_NAME.NumCategories returns the number of categories in a spec block (as an integer)
452
453 SPEC_NAME.Category[I].Name returns the category name corresponding to the Ith category in a spec block
454 (first category is index 0)
455
456 SPEC_NAME.Category[I].NumVariables returns the number of variables in a Ith category (first category is
457 index 0) in a spec block
458
459 SPEC_NAME.Category[I].Variables[J].Name returns the variable name corresponding to the Jth variable in
460 the Ith category within a spec block.
461

```

```

462
463 STIL.4 mandatory requirements for TestBase
464 // The STIL.4-mandated contents for TestBase are shown below. If a user provides an alternate
465 // definition, that definition MUST include all the elements shown below. The purpose of
466 // TestBase is to provide a common set of elements for all TestType and FlowType definitions.
467 TestBase {
468     Parameters {
469         Out Const String TestID { InitialValue ""; } // Empty string – set when test or flow is instantiated.
470         Out Integer execResult { InitialValue 0; } // Return value from execution
471         // execResult == 0 -> Pass, execResult != 0 -> Fail
472         // 0 = Pass, non-zero = Fail
473         // Define Pass = 0; Define Fail = !Pass
474         // For the normal case of only one fail mode, execResult = 1 on fail.
475         In Bin failBin { InitialValue NoBin; } // For component values, see section 3.2.5 of Ernie's bin doc.
476     }
477     PreActions { } // No PreActions
478     // TestExec;
479     PostActions { } // No PostActions
480     // Semantics: PassActions or FailActions are selected by the implied arbiter:
481     // if (execResult == Pass) { Do PassActions } Else { Do FailActions }
482     PassActions { } // No PassActions. Pass Actions are executed if execResult == Pass
483     FailActions { // Fail Actions are executed if execResult == Fail
484         SetBin failBin; // Perform binning based on the current value of failBin.
485         // If value of failBin is undefined, then no binning takes place
486         // This is the equivalent to:
487         // If (failBin != Undefined) { SetBin failBin; }
488         // Vet this against industrial practice (ASAP, SmarTest, OTPL, Envision, Stylus)
489     } // end FailActions
490 } // end TestBase
491

```

492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546

STIL.4 definition of default FlowNode

```
// When an unnamed (global) FlowNode definition occurs OUTSIDE of any other scope (i.e., the scope of
// Tests or Flows, which is the only other place where it's allowed to define a FlowNode), that FlowNode
// definition becomes the default FlowNode definition. If the user does not provide a global FlowNode
// definition, the STIL .4 standard dictates that the following definition be used.
FlowNode {
  PreActions { }
  TestExec EXEC_OBJECT_NAME;
  SetResult CurrentExec.execResult; // i.e., CurrentExec is an alias for EXEC_OBJECT_NAME
  PostActions { }
  ExitPorts {
    Port FAIL Result == Fail {
      SetBin CurrentExec.failBin;
      Stop; // Jumps to EntryPoint initiator with test result Result
            // (i.e., EXEC_OBJECT_NAME.execResult)
            // Could also use Return Result; which will return to
            // default FlowType, which will Stop.
    }
    Port PASS Result == Pass { Next; }
  } // end ExitPorts
} // end FlowNode
```

STIL.4 definition of default FlowType

```
// If not provided by the user, the STIL.4 environment or toolset MUST provide for an unnamed default
// FlowType as defined below.
FlowType {
  // No Inherit keyword – so this FlowType includes all elements defined by TestBase above
  // Parameters – use TestBase Parameters
  // Variables – no local variables
  // PreActions – no PreActions
  // Flownodes – no flownodes specified in the type – will be provided at instantiation
  // SetResult CurrentExec.execResult;
  // PostActions – no PostActions
  // PassActions – no PassActions
  FailActions { // Fail Actions are executed if execResult == FAIL
    Stop; // execResult may be set by Return statement in FlowNode sequence,
          // or any FlowNode in the FlowNode sequence may issue a Stop itself,
          // which causes an immediate jump back to the EntryPoint initiator. In
          // that case, the Flow actions (Post, Pass or Fail) are skipped.
  } // end FailActions
}
```

547

548

STIL.4 definition of TestType NoOpType and Test NoOp

549

550 // In order to help demonstrate concepts in STIL.4, it's necessary to define a TestType, and show how a

551 // Test can be instantiated using this type. Therefore, we define an example TestType called NoOpType,

552 // and instantiate a Test called NoOp from it. This test can be used to illustrate any of the flow concepts

553 // without getting bogged down in the definition of the many actual TestTypes that might be needed on any

554 // specific tester.

555

556 **TestType** NoOpType {

557 // No Inherit keyword – so this FlowType includes all elements defined by TestBase above

558 // Parameters – use TestBase Parameters

559 // Variables – no local variables

560 // PreActions – no PreActions

561 // No TestExec. No operation performed. Or – use TestExec; (with no tokens).

562 // Test library must contain a Test named NoOp, which sets the value of

563 // NoOp.execResult to the appropriate value (PASS (0) or FAIL (non-zero))

564 // PostActions – no PostActions

565 // PassActions – no PassActions

566 // FailActions – use FailActions as defined in TestBase

567 } // end TestType NoOp

568

569 **Test** NoOpType NoOp