

## Title: STIL 1450.4 Test Flow Extension Working Group Issues and Resolutions Document

### History:

- 28 June 2006 – DD added issues and resolutions from the subgroup and WG meetings to bring this document up to date. Reviewed open issues in WG call and resolved issues as shown.
- 14 August 2007 – JO Updated issues and resolutions (color-coding the issue number, as described below). Added new issues.

### Issue color-coding

Green	Issue updated with new proposed resolution as of Aug. 14, 2007
Tan	No change since issues/resolutions list last issued.
Yellow	Issue not yet resolved. Still needs study, with proposed resolution
No color	Issue resolved – does not need attention.

### Working Group Participants List:

1. (PB),Paulo Bernardi,bernardi@EVANCON.POLITO.IT
2. (BC),Bill Chown,billchown@YAHOO.COM
3. (CD),Carol Dowding,2dowdings@LPBROADBAND.NET
4. (D2),Dave Dowding,dkdowding@LPBROADBAND.NET
5. (DF),Daniel Fan,daniel\_fan@CREDENCE.COM
6. (MG),Michelangelo Grosso,michelangelo.grosso@POLITO.IT
7. (K),Kirit,kirit@SIMUTEST.COM
8. (HL),Hau Lam,hau@TESSI.COM
9. (SL),Sam Lee,Sam\_Leet@ASEGLOBAL.COM
10. (SSL),Steve Lill,steve.s.lill@INTEL.COM
11. (YM),Yuhai Ma,Y.Ma@ADVANTEST.COM
12. (GM),Greg Maston,g.a.maston@IEEE.ORG
13. (TM),Tom Micek,Tom.Micek@FREESCALE.COM
14. (LM),Larry Moran,larry.moran@TERADYNE.COM
15. (JM),Jim Mosley,jim\_r\_mosley@CREDENCE.COM
16. (CN),Chris Nelson,chris.j.nelson@INTEL.COM
17. (JO),Jim O'Reilly,jim\_oreilly@AGILENT.COM
18. (BP),Bruce Parnas,B.Parnas@ADVANTEST.COM
19. (SP),Sundar Pathy,sundar.v.pathy@INTEL.COM
20. (BR),Bob Roberts,bob\_roberts@CREDENCE.COM
21. (OR),Oscar Rodriques,oscar@ROOS.COM
22. (JS),Jose Santiago,jose.santiago@PHILIPS.COM
23. (DVS),Diwakar Singh,diwakar.v.singh@INTEL.COM

- 24. (DS),Douglas Sprague,dsprague@US.IBM.COM
- 25. (VT),Vincenzo Tancorre,vincenzo.tancorre@ST.COM
- 26. (TT),Tony Taylor,t.taylor@IEEE.ORG
- 27. (SB),Siew Beng Thum,siew.beng.thum@INTEL.COM
- 28. (EW),Ernst Wahl,ejwahl@ATT.NET
- 29. (GW),Gregg Wilder,gwilder@DAL.ASP.TI.COM
- 30. (WG) Working Group

**Table 1: Summary of Issues and Resolutions from the P1450.4 Working Group**

Ident	Issue	Resolution
WG01	Weekly conference calls have been initiated since the later part of 2002. They are held Tuesday mornings at 9:30 AM Pacific USA time.	<a href="#"><i>Common Agreement by working group Prior to 3 June 2003.</i></a>
WG02	Guiding approaches to developing the dot4 Flow extension: There will be at least four sets of documents the working group will have as the extension is developed: <ul style="list-style-type: none"> <li>▪ Meeting Minutes (web-site posted)</li> <li>▪ Working Draft document (updated regularly, and web-site posted)</li> <li>▪ Issues and Resolutions (updated and web-site posted)</li> <li>▪ Various proposed syntax solutions and supporting documents (tables, block diagrams, etc.) provided by working group participants and ATE vendors</li> </ul>	<a href="#"><i>Common Agreement by working group Prior to 3 June 2003.</i></a>
WG03	Guiding approach to developing the dot4 Flow extension: Use Cases – Descriptions of scenarios of test flows used to verify, demonstrate, and illustrate the usage of the proposed STIL Test Program Flow extension syntax.	<a href="#"><i>Common Agreement by working group Prior to 3 June 2003.</i></a>
WG04	Guiding approach to development of dot4 Flow extension: Syntax examples – During development of the extension syntax, syntax examples should be used as part of the decision process to determine the best syntax/structure for the Flow information. Once the syntax has been determined by a majority vote of the working group, a syntax example section for each block of syntax can have one or more example segments to illustrate the usage of that syntax block in the Working Draft.	<a href="#"><i>Common Agreement by working group on 3 June 2003.</i></a>
WG05	Guiding approach to developing the dot4 Flow extension: Conceptual descriptions (informative rationalization) should be in the working draft for the proposed syntax to help clarify and illustrate the intent of said syntax.	<a href="#"><i>Common Agreement by working group on 3 June 2003.</i></a>
WG06	Guiding approaches to developing the dot4 Flow extension: This document is intended to be a living entity that the working group can use	<a href="#"><i>Common Agreement by working group on 17 June 2003.</i></a>

	to capture issues, questions and actions and their resolutions. It can contain a list of “Pros” and “Cons” on an issue. Each “Issue” item should have its status in the “Resolution” column. Status can show a decision of acceptance or rejection to the working draft, or that the item has been discussed but resolution determination has been “tabled” for later action. Each item’s decision(s) should be noted in the Resolution cell indicating reasons for the decision and the date of that decision. This or a similar document can be used as a vehicle for ballot issues and resolutions.	
WG07	Guiding approaches to developing the dot4 Flow extension: The Working Draft Document is the proposed document that is taken to ballot and ultimately turned over to the IEEE for inclusion in the IEEE 1450 Std document. Items shall be included in the Working Draft Document by majority vote of the participating members of the working group.	<i>Common Agreement by working group on 17 June 2003.</i>
EW01	Should we use an object model as guide to help define and conceptualize the syntax of various blocks of Test Program Flow? Pros: <ul style="list-style-type: none"> <li>▪ The model helps the Test Case example builder to understand how to use a syntax proposal.</li> <li>▪ The model is used in Agere’s ATPG supporting Advantest's programming environment, the Viewpoint Operating System and TDL (Test Development Language.)</li> <li>▪ The model provides for a well documented development methodology for organizing data and related functions, including information (object attributes and relationships), state (state-event-transition), and process (data flow) models</li> <li>▪ Some hold that this model lends itself to graphical representation that can be used to promote comprehension</li> </ul> Cons: <ul style="list-style-type: none"> <li>▪ If we hold strictly to the model to define the syntax some complexities may be introduced that could be simplified.</li> <li>▪ Object oriented approach may not be the best model for this proposed extension as a static model</li> </ul>	<p><i>Concern is expressed that “object model” may imply strict adherence to a given technology or implementation. Perhaps a “graphical representation” such as a block diagram can help in the development (and perhaps something like this in the informative portion of the Working Draft Document.)</i></p> <p><i>Consideration should be applied to the definition of the Flow extensions to serve both Object Oriented implementations as well as other methodologies.</i></p> <p><i>Ernie desires a “guaranteed” object-oriented nature (default characteristics for a test flow node and a flow block being set.) Perhaps moving from an “object model” to a model that has blocks that have attributes (common denominators that carry forward through the flow). This follows an analogy with a base class methodology. The consumer of the flow extensions needs to have enough information to determine the intent of the data without dictating</i></p> <p><i>Removing the term “object model” but pick out the characteristics of the model that we determine as important to have in the language.</i></p> <p><i>New discussion: Focus on specific parts of Object Model such as Inheritance – Relationships between some constructs are NOT similar by accident, but by design.</i></p> <p><i>On July 22, 2003 the conference call attendees determined that the</i></p>

		<i>language of this issue was not resolvable in the current context. Generally, we agreed to NOT use any "model" vehicles, but to use the notion of "Attributes" on blocks and constructs to describe those entities and their relationships to other entities.</i>
EW02	<p>It has been proposed that the scope of working group be organized into two phases of syntax definition:</p> <ul style="list-style-type: none"> <li>▪ Generally, the working group will maintain two lists of flow entities for the Test Flow extension work. The first is the "Phase 1 List" that contains a prioritized set of items/issues to be addressed with this current P1450.4 working group. The second list is the "Phase 2 List" containing items/issues that are deferred to a later specification effort.</li> <li>▪ As items/issues are brought to discussion, a determination can be made whether this item falls within the scope of Phase 1 or could be better dealt with in Phase two. With a majority vote the item would be moved to the Phase 2 list.</li> <li>▪ Phase 2 list items may be revisited with some reasonable justification, and can be moved to the Phase 1 list with the majority vote of the working group.</li> <li>▪ Management of the Phase 1 and Phase 2 items will likely be noted in this document, and will be maintained on their associated list.</li> <li>▪ The Phase 1 List will show items, their priority for definition and any other information that will help the working group develop the Draft Proposal document.</li> </ul>	<p><i>On July 22, 2003 the conference call attendees determined that this proposal should be changed to clarify a method for the working group to use to determine which flow constructs should be defined in this working group effort and which to defer to a later specification version.</i></p> <p><i>This issue/resolution should be reviewed and agreed upon in the next working group meeting. Dkd 22 July 2003.</i></p>
WG08	<p>The working group determined the list of items needing to be discussed and extension constructs defined. These items were then judged as to whether they are a Phase1 or a Phase2 category item. Priorities were set on the items regarding which should be worked through first, second and so on. The list is found in the Excel document: <a href="http://grouper.ieee.org/groups/1450/dot4/archives/Phase1Phase2InitialList.xls">http://grouper.ieee.org/groups/1450/dot4/archives/Phase1Phase2InitialList.xls</a>. This effort spanned several weekly conference calls and concluded on 8/26/2003.</p>	<i>Common Agreement by working group on 17 September 2003.</i>
WG09	<p>The working group identified unresolved issues to be discussed:</p> <ul style="list-style-type: none"> <li>▪ Binning and BinMap</li> <li>▪ Run-Time Variables: scalar or multi-dimensional (different from 1450.0 Spec/Category types)</li> <li>▪ Distributing constructs within the PatternExec block among the P1450.4 constructs.</li> </ul>	<i>Captured from the meeting notes of 25 February 2004.</i>
EW03	<p>Bypass/Skip of PreActions block: Not only bypass the Body, (common) PostActions, and Arbiter, but also the port-specific</p>	<i>Captured from WG call of Feb 08, 2006: This issue was tabled as unresolved.</i>

	<p>PostActions all the way to the branch point. Why this is needed: the post-actions (common or port-specific) may take certain actions if the test passed or failed, such as incrementing a counter, setting a bin. If the test wasn't executed, then these actions should NOT be done. (Noted as a production running of the test program.)</p> <p>Further discussion on this topic was:</p> <ul style="list-style-type: none"> <li>• Jim O – there are two different uses – one is to skip the test, the other is for debugging branching logic usage of the flow. In the latter case, it makes sense to include the Port Actions. (In the Credence ASAP tester, you could branch to a specific port with its associated bin block assignment action). (Noted as an off-line mode)</li> <li>• Tony and Greg indicated that the if the Skip statement had a integer, it could be used to skip to the arbiter to the ExitActions Port with an assigned an integer.</li> </ul> <p>Further discussion in the Syntax-CM subgroup meeting of 06Jun2006: For the Test and a FlowNode: set to a label or a label and a test result. Could use label or integer (label being preferred) mechanism. Both approaches could be addressed with the use of the optional label or if the optional label was left off, and then the skip could jump directly to the exit branch point of the node. Another is to have to labels a multiple points such that a label can be at the beginning and another at another point such as at the end of a block so that the skip would skip the body of the block or would skip to the start of a block and then process the body of that block only.</p>	<p><i>Further discussion on 06 June 2006 provided the suggestion that:</i></p> <ol style="list-style-type: none"> <li><i>1. The syntax "integer" is changed to "integer or label".</i></li> <li><i>2. The location of the Bypass integer or label is allowed in multiple places throughout the Test or FlowNode such that the user could place the label after a block to indicate that the block is processed before the label's action of exiting the test or FlowNode.</i></li> <li><i>3. Further semantic explanation be placed in the syntax to clarify if the integer/label is not used what the behavior will be for FlowNode multiple exit ports situations... i.e. exit is through the "first" exit port, which is ...</i></li> </ol> <p><i>This further suggestion needs to be ratified by the WG and is left for such action.</i></p>
WG11	<p>TestBase: Should this capability be provided by the Inheritance statement or using a Defaults mechanism?</p> <ul style="list-style-type: none"> <li>- Ernie feels that both are needed.</li> <li>- Bruce indicated that if Inheritance is the mechanism then you don't need Default.</li> </ul> <p>For information on results of voting in this call on this issue see resolution cell to the right.</p> <ul style="list-style-type: none"> <li>• JimO: Both Tony and Greg were suggesting the Defaults approach that Bruce suggested. They were thinking of a Defaults approach, but this description would probably be okay as Bruce described.)</li> </ul> <p>NOTE: The Defaults topic is only described in the Conceptual Model document and has yet to be brought out in the Syntax. There is an inherent default behavior in STIL. The Defaults spoken of here and as noted in the CM is a different type of defaults from the inherent STIL default behavior.</p>	<p><i>Captured from WG call of Feb 08, 2006: Votes were taken on this issue as follows:</i></p> <ul style="list-style-type: none"> <li><i>- Bruce voted to use a required inheritance... or an explicit definition in the TestMethod. In favor of Defaults mechanism, he thinks that this Inherit can be clearly defined explicitly. Later, Bruce indicated that if you Inherit, then you can get the behavior, or you do not use Inherit you must supply all information required by the TestMethod. The tool that reads this must do the checking to ensure that all information is provided and appropriately given. (See Doug Sprague's vote below.) He doesn't want to say that you MUST use Inherit.</i></li> <li><i>- Ernie voted for the inheritance behavior and didn't care if the syntax were to state it. He feels that this is a functional requirement not syntactical requirement. Difference whether you place Inherit or not use but always have it</i></li> <li><i>- Doug voted for inheritance as an explicit element of the construct. If you don't want to use Inheritance then you fill in all the blanks.</i></li> </ul>

	<p>Addition discussion that are follow on from use Case #4 tha need to be added.</p>	<p><i>Resolution:</i>  <i>Proposal: Inherit is required and must be explicit. Voted and carries.</i></p> <p><b>Further Resolution as of 08/14/2007:</b>  <i>TestTypes or FlowTypes which do NOT explicitly use the Inherit statement implicitly inherit from TestBase. If the Inherit statement is used, then the FlowType or TestType being defined explicitly inherits from the type named in the Inherit statement. FlowTypes can only inherit from other FlowTypes; TestTypes can only inherit from other TestTypes.</i></p>
WG12	<p>EntryPoint or EntryPoints – Decided on plural.</p>	<p><i>Captured from WG call of Feb 08: Proposed and voted on to use the plural use of "EntryPoints".</i></p>
WG13	<p>User extensible EntryPoints: Discussed mechanisms. The Syntax subgroup has taken the action item to investigate the mechanism to be used further. Useful to note that Chris Nelson wanted this.</p>	<p><i>Captured from WG call of Feb 08 and tabled for Syntax subgroup discussions to decide on UserDefined Keyword or other mechanism.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>No action has been taken on this issue. Still needs exploration. Note that LTX Envision has user-defined entry points (OnUsr0-OnUsr9, which are invoked via the external interface API or can be mapped to keyboard function keys). Perhaps we could adopt a similar solution. Note that including such user-defined entry points can reduce portability when going from one vendor's system to another vendor's system.</i></p>
WG14	<p>TestModule to be changed to TestMethod? (module does not refer to a type)</p>	<p><i>Captured from WG call of Feb 08 2006: Proposed and voted to change TestModule to TestMethod.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>The keyword is now (and has been since late 2006) TestType.</i></p>
WG15	<p>TestModule (called TestMethod in syntax D16-01March2006): Should this construct include an explicit TestExec statement, or simply execute an unstated (in the syntax) execution method?</p> <ol style="list-style-type: none"> <li>1. If the former, Jim O. believed that there were some WG members that would have issues with a TestModule being able to call something other than the appropriate execution function (i.e., another TestModule or a full TestFlow)</li> <li>2. If the latter, then we need a way of defining how a user can create their own or extend those provided by an ATE vendor, (and of course, how a ATE vendor would actually create such a construct/object, and make it available to the STIL test program)</li> </ol>	<p><i>Captured from WG call of Feb 08 2006 but table and left unresolved.</i></p> <p><i>Doug suggests that the explicit TestExec would be better. Not have things look different from one target tester to another. If there is something explicit be the same for various testers and that the implementation on each tester would be under the hood. It is understood that we need portability and standardization.</i></p> <p><i>Ernie said that in the syntax line 125 and 125 take away the execute_stmt. If create a TestMethod VOH and when called use the TestExec statement to show where it is happening. He thinks that</i></p>

In an effort to clarify this in WG call of 07June2006:  
in Figure 3 of RevK of the CM there was a need to do the function call on a specific tester you would call the function for that tester with the arguments needed. There needed to be another place to specify the tester specific required arguments, etc.  
Bruce remembers doing this for a specific tester would not be in the invocation of the function call.

*this notion of having a tester-specific approach was struck down. That assumption was affirmed as correct.*

*Doug if the execute\_stmt is a function call to a standard template library item. How the standard template library is actually implemented can be different. The only issue whether the arguments and parameters are the same number and order. What you put there is the interface mechanism.*

*Dave: STIL has various mechanisms for providing additional information via Annotations or UserKeywords.*

*It is proposed to standardize on using the "TestExec;" to show where the test "action" should happen.*

*Secondly it is proposed to use the "TestExec (name);" instead of the execute\_stmt. (this eliminates the execute\_stmt {param-list} redundant elements). This second proposal will be added to the syntax on a trial basis to be revisited.*

*These two were voted and carried to move these into the syntax document (vote was conducted on 07June 2006).*

### **Resolution as of 08/14/2007:**

*As agreed, the TestType (formerly called TestMethod and TestModule at various times), has an explicit tokenless TestExec statement to indicate where the actual execution of the desired test takes place. The tokenless TestExec; statement in a (named) TestType implies that a TestType with that name exists in a library which is linked with the STIL test program.*

*Further, there are three other forms of "execution statement" available in a TestType:*

```
FuncExec func_stmt  
FuncExec { ( func_stmt )* }  
flownode_stmt+
```

*The first two forms allow the execution of a vendor- or user-provided function within a TestType. The third form allows the creation of a TestType which actually executes a sequence of flow nodes (creating a TestType which in effect, executes a flow; but that flow is*

		<p><i>"atomic" (i.e., from graphical tools which operate on the Test, you can't see that it's a flow).</i></p> <p><b>See issues WG19 and WG24.</b></p>
EW04	<p>SkipPath vs. Bypass (or both with different means)  - Use Bypass keyword, but want different syntax options to allow for bypassing only the test, or all the post-actions (both the Body or Test and result) or branch-specific (only resolved to use Bypass.)</p> <p>From further discussions from 06June2006 sub-group call:  Bypass/Skip of PreActions block – Debug mode without editing the labels in or out.</p> <p>Further clarifications from Ernie via e-mail sent on 7 June 2006:  We had discussed two uses of the Bypass statement, one which I'll call debug submitted by Jim O'Reilly, the other which I'll call flow-control originally submitted by me, Ernie Wahl. Adhering to our assumption that the single keyword Bypass could be used to serve both purposes, I'd proposed we use three forms:</p> <ol style="list-style-type: none"> <li>1. Bypass() [FlowControl mode],</li> <li>2. Bypass(exit_port_label) [FlowControl mode], and</li> <li>3. Bypass (assignment_statement) [Debug mode],,</li> </ol> <p>thinking that these would be most easily mapped to existing tester languages while not compromising the capabilities of a tester running STIL.</p> <p>A matrix can be created where each of the three operate in two contexts; FlowNode and TestInstance as follows:</p> <ol style="list-style-type: none"> <li>1. Bypass()[Flow Control mode] in TestInstance: Go directly to the single anonymous exit-port bypassing subsequent pre-actions, the arbiter, and post-actions. And in FlowNode: 1. If there is only one exit-port, go directly to that exit-port bypassing subsequent pre-actions, the arbiter, and post-actions, or 2. If there are multiple exit-ports, this statement is illegal.</li> <li>2. Bypass(exit_port_label) [Flow Control mode] in TestInstance: Illegal: test instance exit-port is anonymous. And in a FlowNode: Go directly to the named exit-port bypassing subsequent pre-actions, the arbiter, and post-actions.</li> <li>3. Bypass(assignment statement) [Debug mode] in TestInstance Assign</li> </ol>	<p><i>Captured from WG call of Feb 08: Table to determine different syntax options.</i></p> <p><i>Captured from SubGroup call of June 06 2006 but was tabled and left for a later discussion.</i></p> <p><i>Captured from Working Group call of June 21 2006: Bruce suggested that the debug portion of the proposal not be included within the syntax. Doug agreed that this be left out. It was proposed that we accept as useful to include #1 and 2 and leave out #3 as an implementation approach not part of the language. This was voted upon and the vote carried the proposal to drop #3 in the language.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>Syntax and semantics of proposed solution covering the different forms of bypass need to be worked out.</i></p>

	value(s), presumably to this test instance's data member(s), do not execute test <sup>1</sup> but execute all pre-actions, arbiter, and all post-actions. And FlowNode: Assign value(s), presumably to this FlowNode's test instance's <sup>2</sup> data member(s), do not execute test but execute all pre-actions, arbiter, and all post-actions.	
WG16	TestMethodDefs - Provides a container for one or more definitions of function Definitions - prototypes – essentially a header file. - Proposed and voted to change this to TestFunctionDefs.	<i>Captured from WG call of Feb 08: Proposed and voted to change TestMethodDef to TestFunctionDefs (This vote effects WG14 TestModule issue above.)</i>
WG17	TestObject (CM) vs. TestInstances (syntax): both are intended to refer to same thing. It was proposed and voted to use in both CM and Syntax to use TestInstance.  Additional change was requested to make TestInstances singular.	<i>Captured from WG call of Feb 08: Proposed and voted to change TestObject to TestInstance</i>  <b>Resolution as of 08/14/2007:</b> <i>The keyword TestInstances has been replaced with two keywords: Test and Flow. Both Test and Flow are objects; Test is an object of type TestType, Flow is an object of type FlowType.</i>
WG18	CM shows a TaskNode and a DecisionNode. It was discussed and agreed upon that these are no longer needed, since they can be synthesized from existing elements. It was proposed and voted upon to be removed from the CM.	<i>Captured from WG call of Feb 08: Proposed and voted to remove these two sub-clauses from the CM document.</i>
WG19	ObjectRef (in CM rev.K see figure 3) and TestExec (in syntax D16-01Mar2006): It was discussed and suggested that in the FlowNode use TestExec in both documents. 1. Should the TestMethod (at type) include a TestExec statement, or rely on an execute function whose existence is established by TestBase for that TestMethod? 2. Alternatively, it seems acceptable that another similar keyword (i.e. TestFunctionExec) could be defined and used instead, with the constraint that a TestFunctionExec statement could ONLY call a function defined by TestFunctionDefs.  With further discussion on 13 June Subgroup call: The subgroup proposes that the second (or #2) part not be addressed, but that number one stand as the proposal for this issue.	<i>Captured from WG call of Feb 08: Proposed and voted that the FlowNode will use the TestExec construct in both documents.</i>  <i>The question of whether the TestModule (now to be called TestMethod should include a TestExec was not decided upon and was tabled and left for further discussion. Agreed to using TestExec is okay but whether to allow the parameter section. Dave indicated that the parameter section was redundant in the example UseCase#4. The other issue was to implicit to the language (Flow Extension) and not allow tester specific.</i>  <i>In the working group meeting of 21 June 2006, it was proposed to have the TestExec statement (unnamed and parameter-less) entity. (The function that is called with the TestExec is implicit in nature</i>

<sup>1</sup> In document dot4-ConceptualModelK.pdf, figure 3, that refers to box labeled *Test VOH* inside the box labeled *TestObject* (should now be labeled *TestInstance*).

<sup>2</sup> In document dot4-ConceptualModelK.pdf, figure 3, that refers to the box labeled *ObjectRef (InstanceRef?)* inside the box labeled *FlowNode*.

		<p><i>rather than named.) The proposal was voted upon and carried.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>In addition to the parameterless TestExec statement, three (3) other forms are allowed:</i></p> <pre> FuncExec func_stmt FuncExec { ( func_stmt )+ } flownode_stmt+ </pre> <p>The first form executes a single function, as defined in the FunctionDefs block. The second form executes a sequence of one or more functions - again, as defined in the FunctionDefs block.  <b>(Are both forms needed? See issue WG24).</b></p>
WG20	<p>Arbiter (in CM) and ExitPorts (in syntax):</p> <ol style="list-style-type: none"> <li>1. In syntax, the full set of Boolean expressions in the ExitPorts block of the FlowNode (and, currently, elsewhere) constitute the arbiter.</li> <li>2. It was suggested to use the ExitPort keyword in a FlowNode where the Arbiter is the collection of the Boolean expressions. Also associate with the ExitPort a descriptive tag.</li> <li>3. It was suggested that for the TestMethod and TestFlow a different (new) keyword be used for the Arbiter that directs whether to access PassActions or FailActions.</li> </ol>	<p><i>Captured from WG call of Feb 08: Proposed and voted to use the keyword "ExitPort" in the FlowNode construct in lieu of the CM term of "Arbiter".</i></p> <p><i>The suggestion to use a new keyword for the "Arbiter" function was tabled for further discussion in the syntax sub-group.</i></p> <p><i>In the working group meeting of 21 June 2006, it is proposed to use the keywords PassActions or FailActions in the TestMethod. (PreActions, the test, then PostActions, then the Arbiter and the PassActions and FailActions blocks.) Vote carries for the proposal.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>Within the TestType and FlowType, there are two sets of conditional actions: PassActions and FailActions. There is an implied arbiter – the state of the TestType variable FailFlag (which is actually established in TestBase). The implied arbiter action is:</i></p> <pre> if (FailFlag == FALSE)     execute pass actions else     execute fail actions </pre> <p>Within the FlowNode, many sets of actions (each set corresponding to one of N exit ports) are allowed, with an explicit arbiter – a set of boolean expressions. The first boolean expression satisfied (i.e.,</p>

		<p>TRUE) will select the exit port. The syntax is as follows:</p> <pre> ExitPorts {     ( (PORTLABEL:) boolean_expr { (exit_port_stmt)* } ) * } </pre> <p>The PORTLABEL is optional, to be used if you wish to bypass, in the PreActions, to a specific exit port. If bypassing to a specific exit port label, NONE of the exit port actions are executed.</p>
WG21	ExitPorts (in syntax) is a misnomer and so this should be limited to PassActions – FailActions for TestMethods. Another opinion was to have additional action types for FlowNodes. (Should add to CM figure 3... add the port designation reflect the Port Label.)	<i>Captured from WG call of Feb 08: Proposed and voted to modify the syntax reference of "ExitPorts" be limited to "PassActions" and "FailActions" (with ExitPorts other action types to be allowed for FlowNodes) (This is WG20#3).</i>
WG22	"ExitAction" (in CM) be changed to "ExitPort". It was agreed to use ExitPort within the FlowNode.	<i>Captured from WG call of Feb 08: Proposed and voted to use the term "ExitPort" within the FlowNode in the CM document.</i>
DS01	<p>TestMethod TestBase should be required in the STIL data stream (in a header file) as opposed to being STIL reader knowledge. This should be require as explicit not implied.</p> <p>Further discussion on 14 June 14, 2006 working group call: the interest here is to emphasize the explicit nature of things.</p> <p>E-mail communications from 15June2006 clarification was given on what the issue was intended. The communications were between Ernie and Doug and excerpts are as follows.</p> <p>Ernie to Doug: "in addition to requiring an explicit Inherit statement, (I think) you had also wanted to require an explicit description of base method TestBase as shown on lines 9 - 24 in UseCase4-D16-Syntax... I had thought the TestBase description would, under normal circumstances, be in an "include" file but could alternatively be inline. "</p> <p>Doug's response to Ernie: "Yes, I would expect that to be in the STIL stream of data, be it an include file or straight inline."</p> <p>In Working Group call on the 28<sup>th</sup> of June 2006: Bruce expressed concern that this could be misused. If the "Inheritance" is or the C++ model then TestBase cannot be added to or taken away from but can be used to make a "derived" entity the conforms to the user's requirements including all of TestBase.</p>	<p><i>Captured from WG call of April the 12<sup>th</sup> or 19<sup>th</sup> 2006. The issue was tabled for further discussion by a larger representation of the WG.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>TestBase (whether STIL.4 default or user-specified) must be included in the input stream. If the STIL.4 default TestBase is used, it is up to the tool implementer whether the default TestBase contents are hard-coded in the tool code to appear in the input stream, or read from a tools default file or other include file.</i></p>
JS01	Abstract TestMethod TestBase should be unnamed	<i>Captured from WG call of April the 12<sup>th</sup> or 19<sup>th</sup> 2006. The issue was tabled for further discussion by a larger representation of the WG.</i>

JS02		<p><b>Resolution as of 08/14/2007:</b>  <i>TestBase is a separate syntactical element. The contents of TestBase set the defaults for both TestType and FlowType (assuming that the Inherit keyword is not used in the type definition). STIL.4 will define a minimum set of elements for TestBase. The user can specify their own TestBase definition, but it MUST include at least the minimum set of elements specified by the standard.</i></p> <p><i>The STIL.4 TestBase specification is as follows:</i></p> <p>// The STIL.4-mandated contents for TestBase are shown below.  // If a user provides an alternate definition, that definition MUST include all the elements shown below. The purpose of TestBase is to provide a common set of elements for all TestType and FlowType definitions.</p> <pre> <b>TestBase</b> {   <b>Parameters</b> {     Out String TestID { InitialValue ""; }     Out Boolean FailFlag { InitialValue False; }     Out Real Result { InitialValue NaN; }     Out Int ReturnVal { InitialValue 0; }     In Bin FailBin;   }   <b>PreActions</b> { } // No PreActions   <b>PostActions</b> { } // No PostActions   // PassActions or FailActions are selected by the implied arbiter:   //   if (FailFlag == True) { Do FailActions } Else { Do PassActions   <b>PassActions</b> { } // No PassActions.   <b>FailActions</b> {     <b>SetBin</b> FailBin; // Perform binning based on the current value                       // of FailBin. If value of FailBin is undefined,                       // then no binning takes place     <b>Stop</b>;          // Stop Test or Flow execution    } // end FailActions } // end TestBase </pre>
	STIL should not require TestBase (could be suggested usage)	<p><i>Captured from WG call of April the 12<sup>th</sup> or 19<sup>th</sup> 2006. The issue was tabled for further discussion by a larger representation of the WG.</i></p>

		<p><b>Resolution as of 08/14/2007:</b>  <i>STIL.4 requires TestBase to be defined.</i></p>
EW05	<p>The D16 Syntax Summary document of 01March2006 lacks giving the ability to create instances of TestMethods at the top level of STIL. In the syntax there were only two places where you could create a test_instances (TestFlow and TestProgram blocks). We want the ability to create an instance at the top level of STIL using the keyword "TestInstance" to start. Additionally it is recommended that the type of instance should be added as shown here below:</p> <pre>TestInstance TESTMETHODTYPE_NAME INSTANCE_NAME ;   TESTFLOWTYPE_NAME INSTANCE_NAME { (VAR_NAME = expr;)* (VAR_NAME = [ expr (expr)+];)* } // end TestInstance_stmt</pre> <p>Ernie later updated his preference for the syntax substitution as:</p> <pre>TestInstance TESTMETHOD_NAME;   TestInstance TESTMETHOD_NAME INSTANCE_NAME { (VAR_NAME = expr;)* (VAR_NAME = [ expr (expr)+];)* } // end TestInstance_stmt</pre>	<p><i>Captured from the Subgroup call of May 30<sup>th</sup> 2006. Both Ernie and Dave have drafted examples for Use Case #4 using this suggested syntax. This issue needs to be discussed and a resolution voted upon.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>Tests (formerly called TestMethod instances) and Flows (formerly called TestFlow instances) can now be created (instantiated from predefined types) at both the top level (outside the scope of the TestProgram block) and within any TestProgram block, TestType block, or FlowType block (as part of the Variables block).</i></p> <p><i>Both Tests and Flows can be created from their respective types, using syntax (as shown below) which is consistent for both Tests and Flows, and which allows using default elements as defined by the types, or allows overriding some or all of the elements on a per-instance basis.</i></p> <pre>TEST_TYPE TEST_INSTANCE_NAME; TEST_TYPE TEST_INSTANCE_NAME { ( test_elements_stmt )* } (FLOW_TYPE) FLOW_INSTANCE_NAME; (FLOW_TYPE) FLOW_INSTANCE_NAME { ( flow_elements_stmt )* }</pre>
EW06	<p>Another missing construct is the TestMethod TestFlow as shown in Ernie's example sent out to the reflector on 18 April 2006: UserDefined types are also TestMethod. To allow users to use it as a base class so that they can write their own TestMethod as a TestFlow.</p> <pre>26 // STIL.4 def 27 TestMethod TestFlow {           // NOTE: not represented as TestMethod in syntax summary 28 Inherit TestBase;             // NOTE: inherits properties + allows FlowNode specifications 29 PostActions { 30   If any_test.failed           // NOTE: need syntax 31   { this_test.failed = True } 32 } 33 }</pre>	<p><i>Captured from the Subgroup call of May 30<sup>th</sup> 2006. Both Ernie and Dave have drafted examples for Use Case #4 using this suggested syntax. This issue needs to be discussed and a resolution voted upon.</i></p> <p><b>Resolution as of 08/14/2007:</b>  <i>FlowType (Flow) is a separate syntax construct. It has much in common with TestType (Test) (both FlowType and TestType get some basic attributes from TestBase) – and can be used interchangeably with Tests in TestExec and EntryPoints statements.</i></p> <p><i>Users can create their own TestTypes (from which Tests are instantiated); those tests can execute either Functions or sequences of one or more FlowNodes (each of which may execute other Flows).</i></p>

*STIL.4 defines a standard FlowType; when instantiating a Flow, if a FlowType is not specified, then the standard FlowType is used. Of course, a user can define their own FlowTypes, from which Flows can be instantiated. When instantiating Flows from user-defined FlowTypes, the FlowType MUST be explicitly mentioned.*

*// The flowtype name **stil\_dot\_4\_default** is arbitrary – but it MUST be defined by the standard, just as we’ve defined what TestBase will contain.*

```
FlowType stil_dot_4_default {  
    // 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  
    // PostActions – no PostActions  
    // PassActions – no PassActions  
    // FailActions – use FailActions as defined in TestBase  
}
```

In effect (when taking into account the elements of TestBase), the default FlowType is as follows:

```
FlowType stil_dot_4_default {  
    Parameters {  
        Out String TestID { InitialValue “”; }  
        Out Boolean FailFlag { InitialValue False; }  
        Out Real Result { InitialValue NaN; }  
        Out Int ReturnVal { InitialValue 0; }  
        In Bin FailBin;  
    }  
    PreActions { } // No PreActions  
    PostActions { } // No PostActions  
    PassActions { } // No PassActions.  
    FailActions { // Fail Actions are executed if FailFlag == True  
        SetBin FailBin; // Bin based on current value of FailBin  
        // If value of FailBin is undefined,
```

		<pre> // then no binning takes place  <b>Stop;</b> // Stop Flow or Test execution } // end FailActions } </pre>
WG23	<p>Need to consider support for VMeas/IMeas. These were intentionally left out of .2, but the issue was raised again during the balloting for .3. The .3 response was to defer this to .4, so we need to consider if and how we might deal with this.</p>	
WG24	<p>(See issue WG11). Do we want to allow “Inherit TestBase” as a way of explicitly stating the implicit inheritance from TestBase.</p>	
WG24	<p>(See issue WG15 and WG19). There are two forms of the FuncExec allowed in a TestType or test_elements_stmt:</p> <pre> FuncExec <i>func_stmt</i> (for execution of a single function) FuncExec { <i>func_stmt</i>+ } (for execution of one or more functions) </pre> <p>Since the second form can execute either a single function, or more than one, is there a need for the first form (which can ONLY execute one function)?</p>	
WG25	<p>Variables instantiation within a type definition. Currently, STIL.4 syntax allows for variant A: (which builds on the capabilities introduced in .1). It has been proposed to allow variant B as well. Do we want to allow both forms, or only one? If only one form, which form?</p> <p><b>Variant A:</b></p> <pre> // Define variables block myTestVars. Does NOT actually create // instances of the variables specified in the block definition Variables myTestVars {   Int A;   String myString { InitialValue = "myString"; } }  TestType MyTestType {   Parameters {   }   // Adds the variables block definition to the TestType definition.   // Does not create the variables block until the TestType block   // itself is created (instantiated)   Variables myTestVars;   ... </pre>	

	<pre>     ... }  // Creates the Test block MyTest, based on the TestType MyTestType. // The Test block MyTest includes the variables in the variables block // myTestVars. These variables are local in scope to the Test block // MyTest, and cannot be accessed outside the scope of that block. Test MyTestType MyTest;  <b>Variant B:</b> TestType MyTestType {     Parameters {         }     // Define an unnamed variables block, and add that variables     // block definition to the TestType definition. Does NOT     // actually create instances of the variables specified in     // the block definition.     Variables {         Int A;         String myString { InitialValue = "myString"; }         }     ...     ... }  // Creates the Test block MyTest, based on the TestType MyTestType. // The Test block MyTest includes the variables in the unnamed // variables block. These variables are local in scope to the Test block // MyTest, and cannot be accessed outside the scope of that block. Test MyTestType MyTest; </pre>	
WG26	<p>Usage of all STIL block types as variables.  By allowing usage of all STIL block types as variable types, the syntax of the Variables block definition will have to be reworked so that all valid STIL block types can be adequately defined. Note that the Variables block from .1 includes ONLY IntegerConstant, Integer, SignalVariable, and WFC constant.</p>	<p><b>Resolution as of 08/14/2007:</b>  I (JO) believe that the current syntax draft (to be distributed shortly) addresses this issue in a consistent way.</p>
WG27	<p>Creation (instantiation) of Flows (or Tests) local to a TestType or FlowType block. If we want to allow this (which seems likely), need to work out syntax</p>	<p><b>Resolution as of 08/14/2007:</b>  I (JO) believe that the current syntax draft (to be distributed shortly)</p>



