Coverage for ISO/IEC 8652:2012 and subsequent corrections in ACATS 3.x and 4.x

Section 10

A Key to Kinds and subkinds is found on the sheet named Key. Tests new to ACATS 3.0 are shown in **bold**; ACATS 3.1 in ***bold italic***; ACATS 4.0 in **blue bold**; ACATS 4.1 in ***blue bold italic***. ACATS 4.2 in ***green bold italic***.

Clause	Para.	Lines	Kind	Subkind	Notes	Tests	New	Objective's Priority	Objective Text	Objective notes	Submitted tests (will need work).
10											
	(1)		Redundant								
	(2)		Redundant								
	(3)		General								
10.1											
	(1)		Definitions	Widely Used	Program unit (This is marked redundant, but the definition does not exist elsewhere.)						
	(2)	1-2	Redundant								
		3	Impl-Def								
	(3)		Definitions	Widely Used	Subsystem.						
	(4)		Impl-Def		This means that multiple units in a file should be avoided when possible.						
10.1.1											
	(1)	1-2	Redundant								
		3	Definitions	Widely Used	Child unit.						
		4	Definitions	Widely Used	Root unit.						
	(2)		Syntax								
	(3)		Syntax								
	(4)		Syntax	Negative		**BA11013**	All		Check that **private** cannot be given on a library unit body.	This should be tested, because **private** starts both private with clauses and private library units, meaning that a grammar with single token lookahead cannot tell between them; an explicit check may be needed.	
	(5)		Syntax								
	(6)		Syntax								
	(7)		Syntax								
	(8)		Syntax								
	(8.1/2)		Legality					4	Check that an overriding indicator is not allowed on a library unit (subprogram, instantiation, or renaming).	B-Test.	
	(9)		Definitions	Widely Used	Library unit						
	(10)		Definitions	Widely Used	Parents and roots.						

(11)	1	Redundant	Visibility	This should be tested in 8.1(9), but it has more to do with child units than with general visibility.	CA11002 (public child), CA11003 (grandchild), CA11004 (private part of child), CA11005 (private part of grandchild), CA11008 (private child), CA11009 (private grandchild), CA11015 (generic child), CA11016 (private generic child)			Check that the visible declarations of the parent package are visible in all parts of child units.	
			Visibility		BA11005			Check that a parent body cannot declare a homograph of a child mentioned in a context clause.	
	2	Definitions	Subpart	Ancestors					
	3	Redundant							
	4	Definitions	Subpart	Descendant					
(12)		Definitions	Subpart	Private and public.					
			Visibility	This should be tested for 8.2(4), but these rules were tested here in ACATS 2.x, and we aren't going to move them.	BA11001, BA11002			Check that in the formal part and visible part of a public child, the private declarations of the parent package are not visible.	
			Visibility		CA11010, CA11011			Check that the private declarations of the parent package are visible in all parts of a private child.	
			Visibility		CA11006, CA11007			Check that the private declarations of the parent package are visible in the private part and body of a public child.	
(12.1/2)	1	Definitions	Subpart	Limited view					All limited with tests will use a limited view.
	2	Definitions	Negative	Anything not in the next two paragraphs.	**BA11014**	All		Check that entities other than types and nested packages are not present in the limited view.	B-Test.
(12.2/3)		Definitions		Modified by AI05-0129-1.	**BA11014**	Part	4	Check that nested packages are present in the limited view of a package.	C-Test (it's most likely that none of the other tests will try types declared in a nested package). But the priority is fairly low, since it is hard to imagine how this could go wrong.
(12.3/3)		Definitions		Modified by AI05-0108-1 and AI05-0129-1.	**BA11014**	Part	3	Check that all non-incomplete types are present in the limited view of a package, and all types are incomplete, and that tagged types are tagged incomplete.	It would be nice to have a C-Test to check this, but it is not particularly important, as other C-Tests will provide executable examples of some of the cases, and the B-Test covers all of the combinations.
							7	Check that types present in a limited view do not have a discriminant part, even if the full type does have such a part.	B-Test. Try access discriminants completed with null, and numeric discriminants completed with literals.
(12.4/2)		Definitions			**BA12012**	All		Check that the limited view of a private package is also private.	The only way to test this is via with clauses, and that is done by the listed test.
(12.5/2)	1	Redundant							
	2	Definitions	Not Testable						No known way to test this.
	3	Definitions			**CA11023**	All		Check that the context clause of a limited view is empty.	
(12.6/2)		Definitions			**CA11023**	Part	5	Check that types imported from a limited view appear complete when the library package is visible.	C-Test: Test subunits that inherit full with clauses. Test limited withs inherited from parents.
(13)		Legality	Widely Used	Every child unit tests this.					

			Negative		BA11003		Check that a child library unit may not have anything other than a library package or generic library package as its parent unit.	
(14)	1	Legality	Widely Used	Every child unit tests this.				
			Negative			3	Check that nested program units cannot have parent_unit_names.	B-Test. An Ada 95 rule, claims to be tested, but no test was found..
	2	Legality	Widely Used	Every child body tests this.				
			Negative			2	Check that the body of a nested unit cannot be a library unit; check that the body of a child unit cannot be a nested unit.	B-Test. An Ada 95 rule, but no test found. Seems unlikely to get wrong, however.
	3	Legality	Widely Used	Every library rename tests this.				
			Negative		BA11010, BA11011, BA11012		Check that library level renaming cannot rename anything that is not a library_item.	
(15)		Legality	Widely Used	Every child unit tests this.				
			Negative		BA12007		Check that the parent_unit_name cannot be a renamed unit.	
(16)		Legality			CA11012, CA11013, CA11014		Check that a child of an instance can be an instance.	
						2	Check that a child of an instance can be a renaming.	C-Test.
			Negative		BA11003, BA11008		Check that a child of an instance is cannot be anything other than an instance or a renaming.	
(17)	1	Legality			CA11012, CA11013, CA11014		Check that a child of a generic unit can be a generic unit.	
						2	Check that a child of a generic unit can be a renaming of some other child of the generic unit.	C-Test.
			Negative		BA11003		Check that a child of a generic unit is not something other than a generic unit or a renaming of some other child of the same generic unit.	
	2	Deleted		Deleted by AI05-0004-1 as it is redundant with 10.1.1(18).				
(18)		Legality				2	Check that a child of a parent generic package can be renamed within the declarative region of the parent generic.	C-Test. Combine this with the objective for (17), line 1.
						4	Check that a child of a parent generic package can be instantiated within the declarative region of the parent generic.	That is, an instantiation within the parent generic itself. This does not apply to the children inherited by an instance, only the original child unit. C-Test.
			Negative		BA11009, BA11011, BA11012		Check that the renaming of a child of a generic package cannot occur outside the declaration region of the generic package.	
			Negative		BA11008		Check that a child of a parent generic package cannot be instantiated outside of the declarative region of the generic package.	
(19/2)	1-2	Legality			CA11012, CA11013, CA11014		Check that an instance of a generic with a with_clause for a child inherits the child.	
						5	Check that an instance of a child generic inherits its children in the presence of appropriate with_clauses (AI95-00331).	C-Test. Possibly use the example from the AI.
			Negative		BA11008		Check that an instance of a generic does not inherit children from the generic in the absense of a with_clause for the child.	

		3	Redundant						
	(20)		Legality	Widely used	Any child library subprogram.				
				Negative		BA11007		Check that a child library subprogram may not override a user-defined peimitive subprogram.	
	(21)		Legality	Widely Used	Any library subprogram.				
				Negative		BA1001A			
	(22)		Definitions	Subpart	Any library renaming of a subprogram.				
				Negative			2	Check that a library renaming of a subprogram cannot act as a completion (of a library subprogram declaration).	B-Test. Untested in ACATS 2.x. Not certain that this is testable; since the renaming has to conform, it would be hard to tell the difference between acting as a completion and acting as a replacement declaration. (Jean-Pierre Rosen suggests using different default parameters; as such, it could be a replacement but a completion would be illegal. Not sure of the worth of such a test.)
	(23)		Redundant						
	(24)		Redundant						
	(25)		Redundant						
	(26/2)	1-6,8	Definitions	Subpart	Semantic Dependence - This should be tested as part of 10.1.4(5)				
		7	Definitions	Not Testable	This rule is so goofy, we better test it here.			Check that a unit contain the Address attribute semantically depends on System.	But this is not testable (thankfully), as System is now Pure, which means that it will elaborate before anything else anyway and can be allowed in Pure units. It could ony cause trouble in a Pure unit that the body of System depended on -- a situation that user couldn't construct.
	(26.1/2)		Dynamic	Not Testable					In order to test violation of this rule, we'd have to guess what inappropriate effect that elaboration has. But such effects on only limited by the imagination of the implementer.
	(27)		NonNormative		A note				
	(28)		NonNormative		A note				
	(29)		NonNormative		An example...				
	(30)		NonNormative						
	(31)		NonNormative						
	(32)		NonNormative						
	(33)		NonNormative						
	(34)		NonNormative						
	(35)		NonNormative		...end of example				
10.1.2									
	(1)		Redundant						
	(2)		Syntax						
	(3)		Syntax						
	(4/2)		Syntax						

(4.1/2)	Syntax							
(4.2/2)	Syntax							
(5)	NameRes	Widely used	The normal case is tested by any normal with clause.					
		Visibility	We need to test that with clauses "inherit" to bodies, children and subunits.	CA1108A, CA1108B		2	Check that entities can be used in the body or subunits of a unit if mentioned in a (normal) with clause on the specification.	The existing tests only check packages; there would be some value to checking subprograms as well. But this is unlikely to be wrong, as it is so fundamental.
		Visibility				6	Check that entities can be used in the child of a unit if mentioned in a (normal) with clause on the specification of the (parent) unit.	C-Test. I can't find a test with this objective, although it might happen in some other test.
		Visibility				2	Check that entities can be used in the body or subunits of a unit if mentioned in a limited with clause on the specification.	A C-Test. This probably can't be user-oriented.
		Visibility				5	Check that entities can be used in a child of a unit if mentioned in a limited with clause on the specification of the (parent) unit.	C-Test. This probably can be user-oriented.
		Visibility		**CA12002**	All		Check that entities can be used in the body and subunits of a unit if mentioned in a private with clause on the specification of the unit.	
		Visibility				6	Check that entities can be used in a child of a unit if mentioned in a private with clause on the specification of the (parent) unit.	C-Test. Try private and public children, but not the illegal cases!
		Negative				6	Check that entities cannot be used if not mentioned (or declared in an entity mentioned) in a with clause.	B-Tests. You'd think this was tested, but I can't find any in ACATS 2.x. This is an extremely common error in practice, so it should be tested. Make sure that limited with and private with are covered here.
(6/2)	Definitions	Subpart	Named, mentioned					
(7/2)	Redundant							
(8/2)	Legality	Portion	Lead-in for bullets below.					
		Negative		**CA12002** (private with clause, root library subprogram)	Part	3	Check that the restrictions on the use of private child units in a with clause do not apply to a public child.	C-Test(s). This is pretty basic stuff.
(9/2)	Legality					4	Check that the declaration, body, or subunit of a private descendant of unit L can mention a private child of L in a with_clause.	C-Test. Try declarations (renames, packages, subprograms), bodies, and subunits.
(10/2)	Legality					8	Check that a body or subunit of a public descendant of a unit L can mention a private child of L in a with_clause.	Simple legal case in BA12005. But no tests of renames or subunits, no C-Test.
		Negative		**BA12011**	All		Check that a subprogram body acting as a declaration of a public descendant of a unit L cannot mention a private child of L in a non-private with_clause.	
(11/2)	Legality			**CA12002** (packages, subprograms)	Part	4	Check that the declaration of a public descendant of a unit L can mention a private child of L in a private with_clause.	C-Tests. Try all kinds of declarations (renames, generics?) for the private unit, and for the use of the with clause.
		Negative		BA11012 (private renamings), BA12001, BA12002, BA12003			Check that the declaration of a public descendant of a unit L cannot mention a private child of L in a non-private with_clause.	
		Negative		BA11012 (private renamings), BA12004, BA12005			Check that a unit that is not a descendant of a unit L cannot mention a private child of L in a non-private with_clause.	
				BA12012	All		Check that the declaration of a public descendant of a unit L cannot mention a private child of L in a non-private limited with_clause.	

				BA12012	All		Check that a unit that is not a descendant of a unit L cannot mention a private child of L in a limited with_clause.	
		Negative		**BA12013**	All		Check that a unit that is not a descendant of a unit L cannot mention a private child of L in a private with_clause.	
(12/3)	Legality		Lead-in for bullets below.					
		Negative	(Combined for cases where none of the bullets apply)	**BA12014**, ***BA12018***	All		Check that a name visible due to a private with clause is not allowed in a context-clause use_clause.	
				BA12014, ***BA12018***	All		Check that a name visible due to a private with clause is not allowed in the package visible part or subprogram specification of the unit that has the with clause.	
				BA12014, ***BA12018***	All		Check that a name visible due to a private with clause is not allowed in the package visible part or subprogram specification of a public descendant of the unit that has the with clause.	
				BA12016	All		Check that a name visible due to a limited private with clause is not allowed in the package visible part or subprogram specification of the unit that has the with clause.	
				BA12016	All		Check that a name visible due to a limited private with clause is not allowed in the package visible part or subprogram specification of a public descendant of the unit that has the with clause.	
				BA12015	All		Check that a private with clause does not make entities in the private part of a package visible.	
			Part added by AI05-0122-1. We don't need a limited private with test here, as generics aren't made visible by those clauses anyway.			7	Check that the name of a generic child made visible by a private with clause is not made visible in the package visible part of the unit that has the with clause, or in a public descendant of that unit.	"Sprouting"; create a B-Test, see AI05-0122-1 for an example.
(13/2)	Legality			**CA12002**	All		Check that a name mentioned in a private with clause can be used in a private part, including those of nested and descendant packages.	
(14/2)	Legality			**CA12002** (subprogram, package bodies, subprogram subunits)	Part	2	Check that a name mentioned in a private with clause can be used in a body, including in a subunit.	We could check more kinds of bodies (package, protected, & task subunits, generic units), but it doesn't seem likely to fail.
(15/2)	Legality			**CA12002**	All		Check that a name mentioned in a private with clause can be used in the visible part of a private descendant.	
(16/2)	Legality			**BA12015**	All		Check that a name mentioned in a private with clause can be used in a pragma in the same context clause.	
(17/2)	Redundant		This should be tested by 10.1.6(2/2)					
(18/2)	Legality			**BA12009**	All		Check that a limited with clause cannot appear on a body, subunit, or renaming.	
(19/2)	Legality	Portion	Lead-in for the bullets below.					
(20/3)	Legality			**BA12009**	All		Check that a limited with clause for package L cannot be given on the declaration of L.	
			Added by AI05-0040.	***BA12017***	All		Check that a limited with clause for package L cannot be given on a descendant of L.	
				CA12001 (limited private with)	Part	5	Check that a limited with clause for a child of a package L can be given on the declaration of L.	C-Test. Try a limited with of a public child to declare mutually dependent types between the child and parent.
(21/3)	Legality		Reworded by AI05-0077-1, doesn't change testing.	**BA12010**	All		Check that a limited with clause for package L cannot be given in the scope of a nonlimited with clause that mentions L.	
				BA12010	All		Check that a limited with clause for package L cannot be given in the same context clause as a nonlimited with clause that mentions L.	

	(22/3)		Legality		Reworded by AI05-0077-1, doesn't change testing.	**BA12010**	All		Check that a limited with clause for package L cannot be given in the scope of a use clause that names an entity declared in package L.	
						BA12010	All		Check that a limited with clause for package L cannot be given in the same context clause as a use clause that names an entity declared in package L.	
	(23/2)		NonNormative		A note					
	(24/2)		NonNormative		An example...					
	(25/2)		NonNormative							
	(26/2)		NonNormative							
	(27/2)		NonNormative							
	(28/2)		NonNormative							
	(29/2)		NonNormative							
	(30/2)		NonNormative							
	(31/2)		NonNormative		...end of example.					
10.1.3										
	(1)		Redundant							
	(2)		Syntax							
	(3/3)		Syntax		Aspect_Clauses are added by Ada 2012.					
	(4)		Syntax							
	(5)		Syntax							
	(6)		Syntax							
	(7)		Syntax							
	(8/2)		Definitions	Subpart	Parent body, subunit					
	(9)		Legality	Widely used	Any subunit.					
				Negative		BA1020A (parent is recompiled as a non-body)		2	Check that a subunit is illegal if the the parent body is not present in the environment.	B-Tests. Straightforward case of a non-existent parent is not tested.
						BA1020B (parent has no stubs after recompilation)		2	Check that a subinit is illegal if the parent body does not contain an appropriate stub..	B-Tests. Straightforward cases of parent that never had stubs is not tested.
	(10/2)		Legality	Widely used	Any stub.					
				Negative		BA2001B (non-existent spec only)		3	Check that a package stub is illegal if it doesn't complete a package specification.	B-Test: need to test cases where the wrong kind of entity is the spec.
						BA2001B (non-existent spec only)		3	Check that a task stub is illegal if it doesn't complete a task declaration.	B-Test: need to test cases where the wrong kind of entity is the spec.
								4	Check that a protected stub is illegal if it doesn't complete a protected declaration.	B-Test. (Coverage is claimed in ACATS 2.5, but by a C-Test, which is impossible.)
	(11/2)	1	Legality					4	Check that a subprogram stub does not need to complete a declaration.	C-Test. This was an Ada 83 objective (10.2 T12), but there is no test.
		2		Widely used	Any subprogram stub.					
				Negative				3	Check that a subprogram stub is illegal if it completes something other than a subprogram or generic subprogram declaration.	B-Test. (Marked as covered, but tests don't test this objective.)
		3		Widely used	Any subprogram stub.					

				Negative		BA2011A		Check that a subprogram stub must fully conform with its specification (if any).	
	(12)	1	Legality	Widely used	Any subunit.				
				Negative			4	Check that a subunit must be the same kind of entity as its stub.	B-Test. (Marked as covered, but tests don't test this objective.)
		2		Widely used	Any subprogram subunit.				
				Negative		BA2011A		Check that a subprogram subunit must fully conform with its stub.	
	(13)	1	Legality	Widely used	Any stub.				
				Negative		BA2001A (subprograms only)	2	Check that a stub must be immediately in the parent unit.	B-Test (Try other kinds of stubs).
		2					3	Check that a generic unit containing a stub can be instantiated at a nested level.	C-Test.
	(14)		Legality			CA2002A		Check that the identifiers of stubs in different units can be the same.	
				Negative		BA2001C		Check that the identifiers of stubs cannot be overloaded.	(Other sorts of conflicts are illegal homographs anyway.)
	(15)		PostComp			LA5007D, LA5007E, LA5007F, LA5007G, LA5008D, LA5008E, LA5008F, LA5008G	4	Check that a partition without a subunit for some stub cannot be created.	L-Test (protected stubs not tested).
	(16)		NonNormative		A note.				
	(17)		NonNormative						
	(18)		NonNormative		...end of note.				
	(19)		NonNormative		An example...				
	(20)		NonNormative						
	(21)		NonNormative						
	(22)		NonNormative						
	(23)		NonNormative						
	(24)		NonNormative		...end of example.				
10.1.4									
	(1)		Definitions	Widely used	Environment is used in every compilation.				
	(2)	1	Definitions	Subpart	Order of items in an environment.				
		2	Definitions	Visibility	A subunit acts like it is at the place of the stub.	CA2003A (procedure parent), CA2004A (procedure and package subunit parents)	2	Check that declarations from the parent body before the stub can be used in a subprogram subunit.	C-Test. Need to try a library package parent; it would be easy to make a test like CA2004A to accomplish that.
				Visibility			3	Check that declarations from the parent body before the stub can be used in a package subunit.	C-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
				Visibility			3	Check that declarations from the parent body before the stub can be used in a task subunit.	C-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
				Visibility			3	Check that declarations from the parent body before the stub can be used in a protected subunit.	C-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
				Visibility		BA2003A (procedure parent)	2	Check that declarations from the parent body declared after the stub cannot be used in a subprogram subunit.	B-Tests. Package and subunit parent units should be tried.

			Visibility			3	Check that declarations from the parent body declared after the stub cannot be used in a package subunit.	B-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
			Visibility			3	Check that declarations from the parent body declared after the stub cannot be used in a task subunit.	B-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
			Visibility			3	Check that declarations from the parent body declared after the stub cannot be used in a protected subunit.	B-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
			Visibility			5	Check that declarations made accessible by a with clause on a subunit are not visible in the parent body after the stub of the subunit.	B-Tests. This checks that the subunit isn't a purely syntax insertion.
			Visibility		CA13A01, CA13A02 (package parent)	4	Check that declarations from ancestors other than the parent body can be used in a subprogram subunit.	C-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
			Visibility			5	Check that declarations from ancestors other than the parent body can be used in a package subunit.	C-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
			Visibility			5	Check that declarations from ancestors other than the parent body can be used in a task subunit.	C-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
			Visibility		CA13001 (package parent)	4	Check that declarations from ancestors other than the parent body can be used in a protected subunit.	C-Tests. All kinds of parent bodies (subprogram, package, subunit) should be tried.
					CA2007A (packages only)	2	Check that the elaboration of a stub elaborates the subunit body.	C-Tests for subprograms, tasks, and protected types are needed. These can only test that Program_Error isn't raised, thus the low priority.
	3	Definitions	Widely used	Any reference to a library unit tests this.				
(3/2)		Impl-Def	Widely used	Methods of compilation are implicitly tested by running the ACATS.				
(4/1)		NameRes			CA14028		Check that a library subprogram body replaces an instance of a generic subprogram with the same name.	
						3	Check that a library subprogram body completes a subprogram declaration with the same name.	C-Test. Untested in ACATS 2.x. The only thing that can be tested is that recompilation of a subprogram body (replacement with a different body) does not require semantic dependents to be recompiled. This is only interesting for "traditional model" compilers, although it simulates normal editing of a body and thus has to work for all compilers. Note that the error cases (not conforming, inconsistent) are tested elsewhere.
					CA1012A		Check that a library subprogram body completes a generic subprogram declaration with the same name.	
						5	Check that a library subprogram body replaces a library package or library renames with the same name and acts as a definition.	C-Test. I would have expected an Ada 83 test for this, but I cannot find one.
					BA1010A, BA1010B, ..., BA1010Q, BA1011B, BA1011C		Check that a library subprogram body that completes a subprogram (or generic subprogram) declaration with the same name is illegal if it is not type conformant.	This is sort of a combination test, but it's too important to not test.
					CA1011A		Check that a library subprogram body can replace a non-conformant library subprogram body with the same name if that body does not have a separate specification.	Another combination test that's important.

	(5)	1A	Legality	Widely Used	Any unit that compiles and depends on another. "1A" here refers to the part before the semicolon.					
				Negative		BA1101A, BA1101B, BA3001A, BA3001B, BA3001C, BA3001E, BA3001F, BA3001G		6	Check that a compilation unit cannot be compiled if some unit that it semantically depends on has not been compiled.	As B-Tests, we can only test cases where the unit in question never exists. Everything else is an L-Test because of various permissions. We only have Ada 83 tests, so we need tests for child units and library renames (low priority), and for limited views (via limited withs, higher priority).
		1B				BA3006A, BA3006B, ***BA14003***	Part	3	Check that a compilation unit is illegal if it semantically depends on two versions of the same unit.	This is difficult to test, because implementation permissions allow automatic recompilation, and this is a Legality Rule, not a Post-Compilation Rule. It can only be tested if automatic recompilation will fail. Do not confuse this rule with 10.2(27) [which is a Post-Compilation Rule], which is extensively tested with L-Tests.
						BA14001, BA14002	All		Check that a compilation unit is illegal if it semantically depends on an earlier version of itself.	10.1.2(20/3) is this rule for limited withs; the limited with cases are tested there.
	(6/2)		Impl-Def		Untestable: this is not required.					We cannot insist that only legal units be entered (withable); this permission allows illegal units in the environment. 10.2(27) does not allow them in a final program, however.
	(7/2)	1	Impl-Def		Might not be removed.					Really also covered by 10.2(27) tests.
		2	Impl-Def		Might not be removed.					
		3	Impl-Def		Might not be removed.					Covered by 10.2(19) tests.
		4	Impl-Def		Might not be removed.					
		5	Impl-Def		Might not be removed.					
				Negative	These are cases where the unit must not be removed.	CA13002, CA13003			Check that two child units and/or subunits may have the same simple name as long as they don't have the same full expanded name.	
	(8)		NonNormative		A note					
	(9)		NonNormative		A note					
	(10)		NonNormative		A note					
10.1.5	(1)		Redundant							
	(2)	1	Definitions		Defines Program Unit Pragma					
		2	NameRes					5	Check that the name in a program unit pragma cannot be declared in an outer declarative region.	B-Test. Pragma Inline. Only program unit pragmas that are not library unit pragmas make sense here (otherwise, they'd be illegal in inner scopes). No tests in Ada 95 ACATS; this was mistakenly marked as "nothing new" in it's coverage document. But this whole concept is new!
		3	NameRes	Subpart	Test with each pragma.					
	(3)		Legality	Not Testable	Lead-in for following bullets.					
	(4)		Legality	Subpart	Test with each pragma.					

			Negative				4	Check that a program unit pragma cannot be given first in a compilation.	B-Test. Pragmas Inline, Pure, Preelaborate, Elaborate_Body. (Annex E pragmas should be tested there).	
			Negative				4	Check that a program unit pragma cannot follow a unit that is not a subprogram, generic subprogram, or instantiation.	B-Test. Pragmas Inline, Pure, Preelaborate, Elaborate_Body. Try packages, renames, and subunits.	
			Negative		BA15002 (Preelaborate, Pure)		2	Check that a program unit pragma following a subprogram, generic subprogram, or instantiation, cannot name some other library unit in the same compilation (or none at all).	B-Test. Pragmas Inline, Elaborate_Body. For inline, see the objectives for 10.1.6(5).	
(5/1)		Legality	Subpart	Test with each pragma.						
			Negative		BA15002 (Preelaborate, Pure), ***BA15003 (Elaborate_Body)***	Part	2	Check that a program unit pragma given in the visible part of a program unit cannot name any unit other than the one it is in.	B-Test. Try other pragma Inline. (Preelaborate error is tested twice in BA15002, likely bug, probably Elaborate_Body was intended in file BA150023.A).	
					BA15002 (Inline, Elaborate_Body, Preelaborate, Pure)			Check that a program unit pragma cannot be given in the formal part of a generic unit.		
					BA15002 (Elaborate_Body, Preelaborate, Pure)			Check that a program unit pragma cannot be given in the private part of a unit.		
					BA15002 (Elaborate_Body, Preelaborate, Pure)		2	Check that a program unit pragma given in the visible part of a program unit cannot follow any nested declaration.	B-Test. Try other pragma Inline.	
(6)		Legality	Subpart	Test with each pragma.						
			Negative	Does not apply to library unit pragmas.			4	Check that a program unit pragma given in a declarative_part and after the first declaration cannot name a unit declared somewhere other than this declarative_part.	B-Test. Pragmas Inline. Be sure to check naming the enclosing unit.	
							4	Check that a program unit pragma given in a declarative_part and after the first declaration cannot omit the name or name something other than a program unit.	B-Test. Pragma Inline.	
(7/3)		Legality	Subpart	Test with each pragma.						
			Negative		***BA15003 (Elaborate_Body, Preelaborate, Pure)***	All		Check that a library unit pragma cannot name a nested unit from within that unit.		
				Modified by AI05-0132-1.	***BA15003 (Elaborate_Body, Preelaborate, Pure)***	All		Check that a library unit pragma cannot be given in a nested package without a name.		
					BA15002 (Elaborate_Body, Preelaborate, Pure)			Check that a library unit pragma cannot be given other than as the first item in the visible part.		
(7.1/1)		StaticSem					3	Check that a library unit pragma applied to a generic does not apply to its instances.	B-Test and C-Test possible, for Pure/Preelaborate, and various Annex E pragmas. Test Annex E in Annex E.	BA1507A, CA1507A test this for Pure and Prelaborate
(8)	1	PostComp	Negative		BA15001 (Suppress)		2	Check that a configuration pragma cannot appear after the first compilation unit of a compilation.	B-Test. Try other pragmas: Assertion_Policy, Restrictions, Unsuppress. Test Annex D and H pragmas with those annexes.	
	2	Redundant								
	3	PostComp	Subpart	Test with the individual pragmas.						

	(9/2)		Impl-Def					4	Check that configuration pragmas confirming initially selected partition or system-wide options are accepted.	C-Test(s). Try pragmas Assertion_Policy, Restrictions, Suppress, Unsuppress. Provide an individual pragma in a compilation by itself, then compilations with confirming pragmas.	
	(10/1)		Impl-Adv		Not testable, would depend on the individual pragmas.						
10.1.6											
	(1)		Redundant								
	(2/2)	1	StaticSem	Widely Used	Any legal **with** clause will test.						
				Negative		BA11003			Check that a child unit's parent cannot be a nested package.		
				Negative		BA12008 (normal with), **BA16001 (limited with), BA16002 (private with)**	All		Check that a child unit cannot be named in a with_clause by its simple name or any abbreviated form of its full name.		
				Negative		**BA16002**	All		Check that a unit nested in a library package cannot be mentioned in a nonlimited_with_clause.		
		2	StaticSem	Widely Used	Any legal limited with clause will test (C-test for 10.1.2(20) will check child units).						
						BA16001	All		Check that a package nested in a library package cannot be mentioned in a limited_with_clause.		
						BA16001	All		Check that library subprograms, generic units, and library renames of them cannot be mentioned in a limited_with_clause.		
	(3)		StaticSem	Widely Used	Any legal use of Elaborate or Elaborate_All will test.						
				Negative				6	Check that a pragma in a context clause cannot name units not given in previous with_clauses.	B-Test. Pragma Elaborate and Elaborate_All. Marked as untested in ACATS 2.x; Ada 83 10.5 T9 also untested.	
								4	Check that a use clause given in a context clause can name units mentioned in previous with_clauses, and declarations in those units.	C-Test. Try both package use and use type. Marked as untested in ACATS 2.x. May happen in other tests, but not all cases.	
				Negative		BA1101G (previously use visible)		6	Check that a use clause given in a context clause cannot name entities that are neither units mentioned in previous with_clauses or declarations in those units.	B-Test. Try both package use and use type. Try using packages that are in the environment, but not withed. Marked as untested in ACATS 2.x. Ada 83 10.1 T9 also untested.	BA1101G (Dan added a child unit subtest)
	(4)		StaticSem	Widely Used	Any legal subunit will test basic cases.						
						CA2004A, CA13003			Check that the parent_unit_name of a subunit can name a stub in a subunit.		
								3	Check that the parent_unit_name of a subunit can name a stub in a child unit.	C-Test. Untested in ACATS 2.x.	
				Negative		BA2001F			Check that the parent_unit_name of a subunit cannot directly name a stub (without naming the parent unit).		
				Negative				3	Check that the parent_unit_name of a subunit cannot name a child unit without naming the parent of the child unit.	B-Test. Untested in ACATS 2.x. Assuming that we have a stub S in a parent unit P.C, we mean to test that C.S is not a legal parent_unit_name.	

	(5)		StaticSem		Any pragma given as a compilation unit will test, but these are rare (so we test this here).			3	Check that a pragma Inline given after a library subprogram declaration or library instance can name the declaration.	C-Test. Be sure to try child subprograms. We're only trying Inline here for simplicity, others will be tested in place. CA21001 does test this for Preelaborate.
				Negative				4	Check that a pragma Inline following a subprogram, generic subprogram, or instantiation, cannot name some other library unit (or none at all).	B-Test. Be sure to try naming some other unit in the same compilation. We're only trying Inline for simplicity. This should have been tested for Ada 83 (6.3.2 T2) but was not.
	(6/2)		StaticSem			CA11012, CA11013, CA11014			Check that the generic child of a generic library unit can be mentioned in a with clause.	
								3	Check that the generic child of a generic library unit can be mentioned in an Elaborate or Elaborate_All pragma.	C-Test. There are no tests for Elaborate_All and no new tests for Elaborate in ACATS 2.x, so this must be untested.
								2	Check that the generic subprogram child of a generic library package can be followed with a pragma Inline naming the child.	C-Test. But the pragma will be ignored on most implementations, so this is very low priority.
10.2	(1)		Definitions		Partition					Note: We only test active partitions with main subprograms in this clause; other types of partitions are tested by Annex E.
	(2)	1	Definitions		Partition					
		2	Redundant							
		3	Definitions		Explicitly assigned units					
		4	Impl-Def		To some extent, this is tested by running the ACATS.					
		5-6	Definitions		Needed. Note we will test the definition of "needed" here when possible, it really isn't used in other rules.					
	(3)		StaticSem		This rule is really redundant with the one that says any explicitly assigned units are included in a partition.					
	(4)		StaticSem	Widely Used	Every legal program tests this.					
	(5)		StaticSem					2	Check that a needed package body is included in a partition.	Try a package body that has non-trivial elaboration, and is only needed because of pragma Elaborate_Body.
	(6)		StaticSem	Widely Used	Every legal program with subunits tests this.					
	(6.1/2)		StaticSem			**CA20003**	All		Check that a package that is needed only because it is referenced in a limited with clause in included in a partition.	
				Negative	(For the whole set.)			2	Check that a unit that is compiled (and in the environment) but not needed is not included in a partition.	C-Test. Use a package and instance with non-trivial elaboration to test. Not very likely to be wrong.
	(7)	1	Impl-Def		Main subprogram. Every ACATS test implicitly tests this.					

	2	Legality	Negative				Check that a main subprogram cannot be a generic subprogram, package, generic package, or package renames.	Would need a B-Test, but not testable. A partition does not require a main program, and an implementation could use the same command that designates a main subprogram to "explicitly assign" a (single) package to a partition. Such a partition would be a legal "mainless" ones. There's no benefit to requiring different commands for "mainless" partitions (nor is there any RM support for that). Indeed, such a B-Test was deleted during the development of ACATS 2.0.
(8)		Definitions		Environment task - every legal test tests this.				
(9)		Definitions	Subpart	Elaboration dependence				Test as part of Elaborate pragma tests.
(10)		Definitions		Structure of the environment task.				
(11)		Definitions						
(12/2)		Definitions						
(13)		Dynamic	Portion	Lead-in for the following bullets.				
(14)		Dynamic			CA5003A, CA5003B	3	Check that the elaboration order of units is such that there are no forward elaboration dependencies.	C-Tests are needed for Ada 95 cases such as child units (and parent units), and library renames.
(15/3)		Dynamic		Changed to aspect by AI05-0229-1.		6	Check that a unit to for which aspect Elaborate_Body is True is elaborated immediately after its specification.	C-Test. This marked as untested in ACATS 2.x.
(16)		Dynamic	Not Testable	Pure units have no interesting elaboration to check.				The only possible test is to check that Program_Error is not raised by calls on a Pure unit from an impure unit. But the checks are likely to be omitted even if the unit wasn't elaborated properly, so a test would be useless.
(17)		Dynamic				2	Check that preelaborable units are elaborated before any non-preelaborable units.	C-Test. Check cases where some other order might make sense in the absence of the Preelaborate pragma. Not tested in ACATS 2.x.
(18)	1	PostComp			LA5001A		Check that a partition which contains a unit A which withs and mentions in an Elaborate pragma a unit B whose body withs A cannot be created.	
						4	Check that a partition which contains a unit A which withs and mentions in an Elaborate_All pragma a unit B whose body depends on units that with A cannot be created.	L-Test. Marked as not testable in ACATS 2.x. The AARM disagrees.
						4	Check that a partition which contains two units which contain pragma Elaborate_Body and whose bodies **with** the other unit cannot be created.	L-Test. Marked as not testable in ACATS 2.x. The AARM disagrees.
	2	Impl-Def		Elaboration order beyond rules.				
(19)					CA20002		Check that a partition can be created even of the environment contains more than one unit with the same expanded name.	
		Negative			LA20001		Check that two units or subunits with the same expanded name cannot be included in the same partition.	
(20)		Dynamic	Portion	Lead-in for the following bullets.				
(21)	1	Dynamic	Widely Used	Any test checks this rule.				
	2-3	Impl-Def		Many compilers don't even support parameters or results.				
(22)		General		"or"				

(23)		Dynamic	Not Testable	It doesn't make sense to try to test for extra effects in this case, as they could be anything.					
(24)		Impl-Def		These are tested by running the ACATS tests.					
(25)	1	Dynamic	Not Testable	Tested by running ACATS tests.					
	2	Impl-Def							
	3-5	Dynamic			C761001			Check that controlled objects declared immediately within a library package are finalized following the completion of the environment task (and prior to termination of the program).	
					CXC7004			Check that the environment task waits for the termination of library-level tasks, and that Is_Callable is properly False while waiting.	This also checks that the environment task is the master of such library-level tasks.
							4	Check that controlled objects are finalized even if the environment task is aborted.	C-Test. This will require using the Task_Id library to abort the environment task (it doesn't have a name), so this will have to be a CXC test. Careful: avoid tasks, see 10.2(30).
(26)	1	BoundedErr					2	Check that a task that is created and activated after the environment task starts finalization either works normally (but possibly not waiting for termination) or raises Program_Error.	C-Test. This mainly checks that the program doesn't crash. This can happen in sort-of reasonable code, so it probably should be checked. This would have to happen in a finalization handler; take care that the task does its action before the finalization handler is allowed to return (otherwise an incorrect result might appear to happen from allowed early termination).
	2	BoundedErr	Not Testable	Unspecified behavior					
(27)		PostComp		This covers all consistency checks.	LA5007A, LA5007B, LA5007C, LA5008A, LA5008B, LA5008C		2	Check that a partition cannot be created if a needed library unit is missing.	L-Tests (child unit bodies not tested).
					LA5007D, LA5007E, LA5007F, LA5007G, LA5008D, LA5008E, LA5008F, LA5008G		3	Check that a partition without a subunit for some stub cannot be created.	L-Tests (protected subunits not tested). This objective is the same as 10.1.3(15).
					LA14001 – LA14027, ***LA20004***		5	Check that an inconsistent partition cannot be created.	L-Tests (protected subunits and child units not tested; private withs not tested). These tests are all attributed to 10.1.4(5), but that can only be tested at link-time, and the post-compilation rule is here, not in 10.1.4.
					LA20002 (subprogram body), **LA20003** (package spec)	Part	5	Check that a partition inconsistent because of the use of limited withs cannot be created.	L-Tests. Check that significantly changing a unit (for instance, deleting a type) referenced through a **limited with** makes the partition inconsistent. This really is part of the previous objective. Test this in child units, subunits, etc.; check both limited with only and that a limited with and a regular with see the same version of a unit.
(28)	1	Definitions	Widely used	Active partition -- applies to virtually all tests.					
	2	Impl-Def							

	(29)	Impl-Def	Widely used	Virtually all tests have this sort of main subprogram.				
						1	Check that a subprogram generic instantiation can be a main subprogram	C-Test. This has no-test priority because the ARG voted no action rather than confirm for AI95-00172.
						1	Check that a child subprogram can be a main subprogram.	C-Test. This has no-test priority because the ARG voted no action rather than confirm AI95-00172. Not sure it is worth reopening that can of worms.
	(30)	Impl-Def		But finalization still has to happen, see 10.2(25).				
	(31)	NonNormative		A note				
	(32)	NonNormative		Another note				
	(33)	NonNormative		Another note				
	(34)	NonNormative		Last note				
10.2.1	(1)	General						
	(2)	Syntax						
	(3)	Syntax						
	(4)	Definitions		Illegal cases of library unit pragma rules are tested in 10.1.5.				
	(4.1/2)	Syntax						
	(4.2/2)	Syntax						
	(5)	Legality	Portion	Lead-in for following bullets.				
	(6)	Legality			BA21002 (Pure, non-generic), BA21A02 (Preelaborate, generic), BA21003 (Preelaborate, generic package subunit)	4	Check that the elaboration of a preelaborated unit cannot execute a non-null statement.	B-Test: Try a statement without a call (if or case with a static expression), and try in a non-generic package subunit. Try cases where the category is specified by an aspect rather than a pragma.
	(7/3)	Legality		Moved by AI12-0175-1, not changed.	BA21002 (Pure, body), BA21A01 (Preelaborate, instance), BA21A02 (Preelaborate, generic body), **BA21A03** (Preelaborate, spec), BA21003 (Preelaborate, generic package subunit)	4	Check that the elaboration of a preelaborated unit cannot call a non-static function.	B-Test: try cases in Pure bodies and Preelaborate specs; also in non-generic package subunits. Try cases where the category is specified by an aspect rather than a pragma. (Careful: avoid the conversion functions allowed by AI12-0175-1.)
					BA21002 (Pure, body), BA21A02 (Preelaborate, generic body), **BA21A03** (Preelaborate, spec)	4	Check that the elaboration of a preelaborated unit can include a call to a static function.	C-Test: Try in a package subunit. Need some executable tests with these pragmas (not just occurrences in B-Tests). Try cases where the category is specified by an aspect rather than a pragma.
				The elaboration of a generic unit does nothing, so none of these rules apply in a generic spec (bodies have their own rules).	BA21A01 (Preelaborate, formal function)	4	Check that the elaboration of a preelaborated generic specification can include a call of any function (including a formal subprogram).	C-Test: Rechecked in the instance, only could pass for a formal function. Try Pure, OK instances. Try cases where the category is specified by an aspect rather than a pragma.
	(8)	Legality			BA21002 (Pure, body), BA21A01 (Preelaborate, instance), BA21A02 (Preelaborate, generic body), **BA21A03** (Preelaborate, spec)	4	Check that the elaboration of a preelaborated unit cannot include the evaluation of a primary that is the name of an object unless it is static or a discriminant.	B-Test: try cases in Pure bodies and Preelaborate specs; also in package subunits. Try cases where the category is specified by an aspect rather than a pragma.

					BA21002 (Pure), BA21A02 (Preelaborate, generic body)	5	Check that the elaboration of a preelaborated unit can include the name of a static object.	C-Test: Try in package subunits and with Preelaborate. Try cases where the category is specified by an aspect rather than a pragma.
					BA21A02 (in default)	5	Check that the elaboration of a preelaborated unit can include the name of an enclosing type's discriminant.	C-Test: Try in package subunits and using Pure. Use the discriminant to define a discriminant dependent type and declare an object. BA21002 claims to test this, but the expression is not evaluated when the type is elaborated and the type is not otherwise used, so it is bogus.
				The elaboration of a generic unit does nothing, so none of these rules apply in a generic spec (bodies have their own rules).	BA21A01 (Preelaborate, formal object)	2	Check that the elaboration of a preelaborated generic specification can include the evaluation of a primary.	C-Test: Rechecked in the instance, only would pass for a formal object. Try Pure, OK instances. Try cases where the category is specified by an aspect rather than a pragma.
(9/3)	1	Legality		The initialization part of the rule was restored by AI05-0028.	BA21002 (Pure, body), BA21A01 (Preelaborate, instance), BA21A02 (Preelaborate, generic body), **BA21A03** (Preelaborate, spec), BA21003 (Preelaborate, generic package subunit)		Check that the elaboration of a preelaborated unit cannot include the creation of an object of a type without preelaborable initialization unless it has an initialization expression.	Here we'll test the objects; we'll try to test all of the kinds of types elsewhere.
						5	Check that a preelaborated unit can contain declarations of objects of types without preelaborable initialization inside subprograms.	C-Tests: Try a variety of types for the object, and try in subprogram subunits. Try Preelaborate only (Pure has stricter rules, tested at 10.2.1(15.2/2)).
					BA21A02 (Preelaborate, generic body)	3	Check that the elaboration of a preelaborated unit can include the creation of an object of a type with preelaborable initialization that does not have an initialization expression.	C-Tests: Try a variety of types for the object, and try in package subunits and non-generic packages (all parts). Try Preelaborate only (Pure has stricter rules). BA21002 sort of tries this, but it's intended to test Pure rather than Preelaborable_Initialization.
					BA21002 (Pure)	4	Check that the elaboration of a preelaborated unit can include the creation of an explicitly initialized object of any type (including a type without preelaborable initialization).	C-Test: Try preelaborate and complex (but allowed) initialization expressions.
				The elaboration of a generic unit does nothing, so none of these rules apply in a generic spec (bodies have their own rules).	BA21A01 (Preelaborate, formal type)	2	Check that the elaboration of a preelaborated generic specification can include the creation of an object of any type.	C-Test: Rechecked in the instance, only a formal type could pass. We'll test Pure at 10.2.1(15.2/2).
					BA21002 (Pure, body), BA21A01 (Preelaborate, instance), BA21A02 (Preelaborate, generic body), **BA21A03** (Preelaborate, spec)		Check that the elaboration of a preelaborated unit cannot evaluate an extension aggregate with an ancestor type that does not have preelaborable initialization.	
						3	Check that the elaboration of a preelaborated unit can evaluate an extension aggregate with an ancestor type that does have preelaborable initialization.	C-Test.
(10/2)		Legality	Portion	Lead-in for following bullets.				
(10.1/3)		Legality			BA21A02, BA21003		Check that the elaboration of a preelaborated generic body cannot create an object of a formal private type or extension.	The test objective for BA21A02 is too narrow, but the test is OK.

				Added by AI05-0028.			6	Check that the elaboration of a preelaborated generic body cannot create an object of a discriminated formal derived type.	B-Test. Try cases with the category specified by either pragma or aspect.
				Added by AI05-0028.			5	Check that the elaboration of a preelaborated generic body can create an object of a formal private type, private extension, or discriminanted derived type if the formal type has a pragma Preelaborable_Initialization.	C-Test. Try cases with the category specified by either pragma or aspect.
(10.2/2)		Legality					5	Check that the elaboration of a preelaborated generic body cannot evaluate a primary based on a generic formal type.	B-Test. Try attributes of a formal type in contexts that would otherwise be OK. Consider using the existing foundation (FA21A00) in a new test. Try cases where the category is specified by an aspect rather than a pragma.
(10.3/2)		Legality					6	Check that the elaboration of a preelaborated generic body cannot evaluate a primary based on a generic formal object.	B-Test. Try generic in objects in contexts that would otherwise be OK. Combine with previous objective. Try cases where the category is specified by an aspect rather than a pragma.
(10.4/2)		Legality			BA21A02			Check that the elaboration of a preelaborated generic body cannot call a formal subprogram.	
(11/3)	1	Definitions	Subpart	Tested by previous checks. AI05-0243-1 makes this an optional aspect.					We should try some cases where the category is specified by an aspect. But we'll try only a few such cases as the pragma is preferred. See above.
					CA21002	All		Check that Preelaborate can be specified by an aspect, and that the value can be specified in a different package.	
				Added by AI12-0154, to 13.1.1 (which requires individual tests).	***BA21005***	All		Check that the value of the Preelaborate aspect cannot be defined after the aspect.	
				13.1.1(32/4) requests individual tests.	***BA21005***	All		Check that the value of the Preelaborate aspect must have type Boolean.	
				13.1.1(32/4) requests individual tests.	***BA21005***	All		Check that the value of the Preelaborate aspect must be static.	
	2	Redundant							
	3	Legality			BA21003			Check that package subunits of a preelaborated package enforce the restrictions on preelaborated units.	Additional tests are called out above.
					CA21001			Check that package subunits of a preelaborated subprogram do not enforce the restrictions on preelaborated units.	
					BA21003			Check that a preelaborated package can have a non-preelaborated child unit.	
	4	Legality			BA21A01			Check that the restrictions on preelaborated units are enforced in the private part of a preelaborable instance.	
	5	Legality	Subpart	Any legal test checks this.					
				Clarification from AI05-0034.			7	Check that a preelaborated unit can have a semantic dependence on the limited view of a preelaborated unit.	C-Test. Use a limited with, of course. Try a Pure unit and a Preelaborated unit.
				Widening from AI05-0034.			5	Check that a preelaborated unit can have a semantic dependence on the limited view of a non-preelaborated unit.	C-Test. This probably can't be usage-oriented.
			Negative		BA21003, BA21004 (both try a with clause, child unit)			Check that a preelaborated unit cannot have a semantic dependence on a non-preelaborated unit.	
(11.1/2)		Legality	Portion	Lead-in for following bullets. In theory, these should be tested at 10.2.1(9/2), but there are so many cases, we'll test them here.					

(11.2/3)	1	Legality		AI05-0028 fixed a typo here.	**BA21A03** (private)	3	Check that the partial view of a private type or private extension does not have preelaborable initialization (without the pragma).	B-Test. Try a private extension.
						5	Check that a protected type without entries does not have preelaborable initialization (without the pragma).	B-Test. Declare an object in a preelaborated unit.
					BA21A02		Check that a generic formal private type does not have preelaborable initialization (without the pragma).	
						4	Check that a generic formal derived type does not have preelaborable initialization (without the pragma).	B-Test.
					BA21002 (Pure, anonymous)	5	Check that a protected type with entries does not have preelaborable initialization.	B-Test. Try a protected type definition and separate object, in a Preelaborated unit.
	2	Legality			BA21A02, BA21A03		Check that a task type does not have preelaborable initialization.	
(11.3/2)		Definition	Subpart	Test this with (11.5/2).				
(11.4/3)	1	Legality				4	Check that a type derived from a type that does not have preelaboration initialization does not have preelaborable initialization.	B-Test.
						4	Check that a type extension derived from a type with preelaborable initialization does not have preelaborable initialization if it has components that don't have it.	B-Test.
				As changed by AI05-0221-1.		4	Check that a type extension derived from a type with preelaborable initialization does not have preelaborable initialization if it has discriminants that don't have it.	B-Test; try untagged derivation (see AI05-0221-1).
						4	Check that a type derived from a type with preelaborable initialization (and with extension components that have preelaborable initialization) also has preelaboration initialization.	C-Test. Try both extensions with and without components and untagged derived types.
	2	Legality		As revised by AI05-0028-1.	BA21A01, BA21A02, BA21A03	3	Check that a controlled type does not have preelaborable initialization (without the pragma), unless it has an Initialize procedure that is a null procedure.	Add a test case for a known null Initialize procedure.
(11.5/2)		Legality				3	Check that an elementary type has preelaborable initialization.	C-Test.
					BA21A02 (of a formal private type), BA21A03 (of a private type)	2	Check that an array type whose component type does not have preelaborable initialization does not have preelaborable initialization itself.	B-Test. Try some other cases (controlled types, records with defaults, etc.)
						3	Check that an array type whose component type does have preelaborable initialization also has preelaborable initialization.	C-Test.
					BA21A02 (variable name, function call), BA21A03 (variable name),		Check that a record type which has a component that is initialized with a function call or variable name does not have preelaborable initialization.	
					BA21A02 (controlled component).		Check that a record type which has a component whose type does not have preelaborable initialization does not have preelaborable initialization.	
						3	Check that a record type all of whose components have types with preelaborable initialization or have default expressions that are static has preelaborable initialization.	C-Test.
						4	Check that an interface type has preelaborable initialization.	C-Test. Test this by using it as a progenitor of an extension that otherwise has Pinit.
(11.6/2)	1	Definitions	Subpart	Tested in the next paragraph.				
	2	Legality	Subpart	Legal cases are tested in the next paragraph.				
			Negative			4	Check that a pragma Preelaborable_Initialization cannot appear in a private part or body.	B-Test.

(11.7/3)	1	Legality	Negative	As revised by AI05-0028.	4	Check that a pragma Preelaborable_Initialization cannot denote a type declared other than in the package where it appears.	B-Test. Try types declared in other packages and in nested packages.
					5	Check that a pragma Preelaborable_Initialization cannot denote an elementary type.	B-Test.
					5	Check that a pragma Preelaborable_Initialization cannot denote a non-first subtype.	B-Test.
	2	Legality			6	Check that a pragma Preelaborable_Initialization can be applied to a private type or private extension if the full view has preelaborable initialization, and that the type then has preelaborable initialization.	C-Test. Don't forget to use the types to declare objects.
			Negative		5	Check that a pragma Preelaborable_Initialization cannot be applied to a private type or private extension if the full view does not have preelaborable initialization.	B-Test. Try full types declared in a generic private part (to test sentence 4).
	3	Legality		As revised by AI05-0028.	6	Check that a pragma Preelaborable_Initialization can be applied to a protected type without entries if each component has preelaborable initialization, and that the type then has preelaborable initialization.	C-Test. Don't forget to use the types to declare objects.
					5	Check that a pragma Preelaborable_Initialization cannot be applied to a protected type without entries if any component does not have preelaborable initialization.	B-Test.
					6	Check that a pragma Preelaboration_Initialization cannot be applied to a protected type with entries.	B-Test.
	4	Legality		As added by AI05-0028.	6	Check that a pragma Preelaborable_Initialization can be applied to a controlled type if the parent type and all components have preelaborable initialization and Initialize is a null procedure, and that the type then has preelaborable initialization.	C-Test. Don't forget to use the types to declare objects.
			Negative		6	Check that a pragma Preelaborable_Initialization cannot be applied to a controlled type if the parent type or any component does not have preelaborable initialization or Initialize is not a null procedure.	B-Test. Try Initialize routines defined in a generic private part.
			Negative		5	Check that a pragma Preelaborable_Initialization cannot be applied to a task type.	B-Test.
			Negative		4	Check that a pragma Preelaborable_Initialization cannot be applied to a record or array type which has a component that does not have preelaborable initialization.	B-Test.
	5	Legality	Subpart	Tested in previous objectives.			
(11.8/2)	1	Legality	Subpart	Tested with next sentence.			
			Negative		4	Check that a pragma Preelaborable_Initialization given in a formal part cannot be applied to any type not declared in the formal part.	B-Test.
					4	Check that a pragma Preelaborable_Initialization given in a formal part cannot be applied to any formal type other than a formal derived or private type..	B-Test.
	2	Legality			5	Check that if a formal type has pragma Preelaborable_Initialization, the generic can be instantiated with actual types that have preelaborable initialization.	C-Test.
					6	Check that if a formal type has pragma Preelaborable_Initialization, an attempt to instantiate the generic with an actual type that does not have preelaborable initialization is rejected.	B-Test.
(12)		Impl-Adv	Not Testable	...even if it wasn't advice.			
(13)		Syntax					

(14)	Syntax						
(15)	Definitions		Illegal cases of library unit pragma rules are tested in 10.1.5.				
(15.1/5)	StaticSem	Portion	Lead-in for following bullets; using the fixes of AI05-0035. Also modified by AI12-0232-1.				
(15.2/2)	StaticSem		We'll test these here where it is more obvious that they're covered.	BA21002		Check that the elaboration of a pure unit cannot elaborate a variable declaration.	
				BA21002	3	Check that variables can appear in a pure unit in subprogram, task, and protected bodies.	C-Test. Try subunits, too.
(15.3/2)	StaticSem				5	Check that the elaboration of a pure unit cannot evaluate an allocator of an access-to-variable type.	B-Test. This has to be for an access discriminant in an discriminant constraint of a constant. Try cases with the category specified by either pragma or aspect.
					8	Check that the elaboration of a pure unit cannot evaluate a constant declaration for a private type or private extension, even if that type has preelaborable initialization.	B-Test. Try cases with the category specified by either pragma or aspect.
(15.4/3)	StaticSem		"Defined by the language" can only occur in a Remote_Type package (Annex E), so we don't test that here. Uses change of AI05-0035.		7	Check that the elaboration of a pure unit cannot elaborate a non-derived named access-to-variable type whose storage_size is not specified to be zero.	B-Test. Careful: derived access types are always OK. Try cases with the category specified by either pragma or aspect.
					6	Check that the elaboration of a pure unit can elaborate a named access-to-variable type whose storage_size is specified to be zero.	C-Test. Try cases with the category specified by either pragma or aspect.
(15.5/3)	StaticSem		Uses change of AI05-0035.		7	Check that the elaboration of a pure unit cannot elaborate a non-derived named access-to-constant type whose storage_size is specified to be nonzero	B-Test. Careful: derived access types are always OK. Try cases with the category specified by either pragma or aspect.
					7	Check that the elaboration of a pure unit can elaborate a named access-to-constant type whose storage_size is specified to be zero or is not specified at all.	C-Test. Try cases with the category specified by either pragma or aspect.
(15.6/3)	Legality		Rule added by AI05-0035-1.		5	Check that the elaboration of any pure generic body cannot elaborate a variable declaration or allocator for an access-to-variable type.	B-Test. Check bodies and subunits of a generic unit. Separate test as it comes from an AI. Try cases with the category specified by either pragma or aspect.
					5	Check that the elaboration of any pure generic unit cannot elaborate a named access-to-object type with a specified nonzero storage size, or an access-to-variable without a specified storage size.	B-Test. Try cases with the category specified by either pragma or aspect.
					5	Check that the elaboration of any pure generic body cannot evaluate a constant declaration for a formal private type or private extension, even if that type has preelaborable initialization.	B-Test. Try cases with the category specified by either pragma or aspect.
					5	Check that the Storage_Size of an anonymous access-to-variable type declared at library-level of a generic pure body is zero.	B-Test. Be sure to check uses through an (impure) instance. Try cases with the category specified by either pragma or aspect.
(15.9/5)	StaticSem		Number changed by AI05-0035; originally was (15.6/2); again changed by AI12-0232-1, was (15.7/3).		6	Check that the Storage_Size of an anonymous access-to-variable type declared at library-level of a pure unit is zero.	B-Test. This cannot be tested directly; check that an allocator in a subprogram is illegal for a library-level record type with an anon access component. Check uses in other impure units as well as the pure unit.
(16)	Deleted						

(17/3)	1	Definitions	Subpart	All other pure unit tests check this. AI05-0243-1 makes this optionally an aspect.					We should try some cases where the category is specified by an aspect. But we'll try only a few such cases as the pragma is preferred. See above.
					CA21002	All		Check that Pure can be specified by an aspect, and that the value can be specified in a different package.	
				Added by AI12-0154, to 13.1.1 (which requires individual tests).	***BA21005***	All		Check that the value of the Pure aspect cannot be defined after the aspect.	B-Test.
				13.1.1(32/4) requests individual tests.	***BA21005***	All		Check that the value of the Pure aspect must have type Boolean.	
				13.1.1(32/4) requests individual tests.	***BA21005***	All		Check that the value of the Pure aspect must be static.	
	2		Subpart	Added from AI05-0034. Tested below.					
	3						4	Check that package subunits of a pure package enforce the restrictions on pure units.	B-Test.
				AI05-0035 makes this consistent with preelaborate.			4	Check that package subunits of a preelaborated subprogram do not enforce the restrictions on preelaborated units.	C-Test.
							4	Check that a pure package can have a impure child unit.	C-Test.
	4			A consequence of AI05-0034-1.			7	Check that a pure unit can have a semantic dependence on the limited view of a pure unit.	C-Test. Use a limited with, of course.
							5	Check that a pure unit can have a semantic dependence on the limited view of a non-pure unit.	C-Test. Try a Preelaborated package and a non-categorized package. This probably can't be usage-oriented.
			Negative		BA21003		3	Check that a pure unit cannot have a semantic dependence on a non-pure unit.	B-Test (Try on subunits, package spec).
	5		Subpart	Generic boilerplate; tested as part of other rules.					
	6						8	Check that the full view of any nonlimited partial view declared in the visible part of a pure package is illegal if it does not support external streaming.	B-Test. Check when the type has named access components without attributes, anonymous access components, adds such components via an extension, etc.
							6	Check that the full view of any limited partial view declared in the visible part of a pure package which is extended from a type with available stream attributes is illegal if it does not support external streaming.	B-Test. The full type should add an access component, and not redefine the attributes.
				AI05-0035 requires rechecking of the entire instance spec.			4	Check that a pure package instance cannot contain a variable or named access type with a non-zero storage_size.	B-Test. This checks that rechecking of the instance is performed. Separate test as it comes from an AI.
(17.1/4)		Erroneous	Not Testable	Erroneous execution is never testable. AI12-0076-1 restored this just in the case of Pure packages; AI05-0054-2 removed this erroneousness in some cases, but that caused problems for distribution (Annex E).					

(18/3)		Impl-Perm	Not Testable	We could try to see whether side-effects occur in such cases, but as either possibility is allowed, that has no value. We could try to test cases where this permission doesn't apply to ensure that side-effects happen, but that's not of much value, as it would be hard to guess when a compiler would do this wrong and there are many possibilities. AI05-0219-1 clarifies the wording, but has no effect on testability.				
(19)		Syntax						
(20)		Syntax						
(21)		Syntax						
(22)		Syntax						
(23)		Legality				4	Check that a pragma Elaborate or Elaborate_All cannot be given outside of a context clause.	B-Test. Marked as untested in ACATS 2.x. Try placing the pragmas inside a package spec, in a generic formal part, and as a compilation unit.
(24)		Definitions		Illegal cases of library unit pragma rules are tested in 10.1.5.				
(25/3)		Legality			B720001	5	Check that if a pragma Elaborate_Body applies to a library package, a body must be given.	L-Test. The B-Test checks that it can be given, we also need to check that a program cannot link if it is omitted.
				Aspect Elaborate_Body is added by AI05-0229-1.		5	Check that if aspect Elaborate_Body is True for a library package, a body must be given.	L-Test. We need to check that a partition cannot link if there is no body in this case.
(25.1/2)		Legality				7	Check that the unit in a pragma Elaborate or Elaborate_Body cannot denote a limited view.	B-Test. Check various names only mentioned in limited with clauses. Note that this can't happen for aspect Elaborate_Body.
(26/3)		Redundant		AI05-0229-1 puts the last sentence into the next paragraph.				
(26.1/3)	1	Definitions		Paragraph added by AI05-0229-1. Not testable by itself, but implicitly tested by any other pragma Elaborate_Body test.				
	2	Redundant						
(27)		NonNormative		A note.				
(28)		NonNormative		Another note.				

	Paragraphs:		Objectives with tests:	Objectives to test:	Total objectives:	Objectives with submitted tests:
10	221		145	165	262	2

Must be tested	Objectives with Priority 10	0
	Objectives with Priority 9	0
Important to test	Objectives with Priority 8	3
	Objectives with Priority 7	8
Valuable to test	Objectives with Priority 6	18
	Objectives with Priority 5	33
Ought to be tested	Objectives with Priority 4	43
	Objectives with Priority 3	31
Worth testing	Objectives with Priority 2	27
Not worth testing	Objectives with Priority 1	2
	Total:	165

Objectives covered by new tests since ACATS 2.6	51
Completely:	41