

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

Subclause 13.1.1

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***.

						Objective's					Submitted tests
Clause	Para.	Lines	Kind	Subkind	Notes	Tests	New	Priority	Objective Text	Objective notes	(will need work).
13.1.1	(1/3)		Definitions		Associated declaration						
	(2/3)		Syntax								
	(3/3)		Syntax								
	(4/3)		Syntax								
	(5/3)	1	NameRes	Subpart	This is descriptive text.						
		2		Portion	This is a lead-in for the following						
	(6/3)		NameRes						For an aspect that represents an object, the aspect_declaration can be a name denoting an object of the correct type.	C-Test. Try with Storage_Pool. Low priority since any use of an appropriate aspect will test.	
				Negative					For an aspect that represents an object, the aspect_declaration cannot be an expression of correct type nor a name of an object of the wrong type.	B-Test. Try with Storage_Pool. Medium priority because this is fairly normal resolution.	
	(7/3)		NameRes						For an aspect that represents a value, the aspect_declaration can be an expression of the correct type.	C-Test. Try with Size, Alignment, others?. Low priority since any use of an appropriate aspect will test.	
				Negative					For an aspect that represents an expression, the aspect_declaration can be an expression of the correct type.	C-Test. Try with Pre, Static_Predicate, others?. Low priority since any use of an appropriate aspect will test.	
									For an aspect that represents a value, the aspect_declaration cannot be an expression of the wrong type.	B-Test. Try with Size, Alignment, others? Medium priority because this is fairly normal resolution.	
				Negative					For an aspect that represents an expression, the aspect_declaration cannot be an expression of the wrong type.	B-Test. Try with Pre, Static_Predicate, others? Medium priority because this is fairly normal resolution.	
	(8/3)		NameRes						For an aspect that represents a subprogram, the aspect_declaration can be a name denoting an subprogram with the correct profile.	C-Test. Try with Read, Input. Low priority since any use of an appropriate aspect will test.	
				Negative					For an aspect that represents a subprogram, the aspect_declaration cannot be an expression that is not a subprogram nor the name of a subprogram with the wrong profile.	B-Test. Try with Read, Input. Medium priority because this is fairly normal resolution.	
	(9/3)		NameRes						For an aspect that represents something other than an object, value, expression, or subprogram, the aspect_declaration can be a name denoting an entity of the correct kind.	C-Test. Try with ????. Low priority since any use of an appropriate aspect will test.	
				Negative					For an aspect that represents something other than an object, value, expression, or subprogram, the aspect_declaration cannot be an expression or name denoting the wrong kind of entity.	B-Test. Try with ????. Medium priority because this is fairly normal resolution.	
	(10/3)		NameRes						For an aspect that is given by an identifier specific to the aspect, the aspect_declaration can one of the identifiers specific to the aspect.	C-Test. Try with Synchronized. Low priority since any use of an appropriate aspect will test.	

			Negative				For an aspect that is given by an identifier specific to the aspect, the aspect_declaration cannot be an expression nor some identifier other than the ones specific to the aspect.	B-Test. Try with Synchronized. Perhaps this is better tested for each individual aspect?
(11/3)		NameRes			BD11002, BDD2005	Part		
			Negative		BD11002	All	Check that the usage names in an aspect_declaration given in the visible part of a package are resolved at the end of the the visible part; in particular, names declared in the private part cannot be used.	C-Test. Make sure that items not declared at the point of the aspect_specification can be referenced. The B-Tests try some such cases but (of course) does not attempt to execute them. Possibly tests for Pre/Post will try this? Any stream attribute tests would necessarily do so.
(12/3)	1	NameRes			CD11001	All	For an associated declaration that is a subprogram, check that the names of parameters are directly visible in each aspect_declaration.	
			Negative				For an associated declaration that is a subprogram, check that the name of the subprogram is not visible in each aspect_declaration.	Would be a B-Test, but it's not testable as the name of the subprogram surely will be visible at the point of resolution (the end of the declaration list).
	2				CD11001	All	For an associated declaration that is a type declaration, check that the current instance of the type is directly visible in each aspect_declaration.	
					CD11001	All	For an associated declaration that is a type declaration, check that the names of components are directly visible in each aspect_declaration.	
	3				CD11001	All	For an associated declaration that is a subtype declaration, check that the current instance of the subtype is directly visible in each aspect_declaration.	
(13/3)		Legality	Negative	Widely Used			For an associated declaration that is an object, check that the name of the object is not visible in each aspect_declaration.	Would be a B-Test, but it's not testable as the name of the object surely will be visible at the point of resolution (the end of the declaration list).
				Any correct aspect will test.				
(14/3)	1	Legality	Negative	Widely Used	BD11001	All	An aspect_declaration is illegal if any usage name resolves differently at the first freezing point of the associated entity and at the end of the immediately enclosing declaration list.	
			Negative	Widely Used	BD11001	All	Multiple occurrences of an aspect cannot occur in a single aspect_specification.	
	2			Any correct aspect will test.				
			Negative		BD11001	All	The aspect_mark is illegal if it doesn't identify an aspect of the associated entity.	We just try a few simple cases; each aspect should test this more thoroughly.
(15/3)		Legality	Widely Used	Commonly used for boolean aspects like Pack and Pure.				

		Negative	BD11001	All	The aspect_definition cannot be omitted for a non-boolean aspect	
(16/3)	Legality	Widely Used	Any correct class-wide aspect will test.			
		Negative	BD11001	All	An aspect cannot include 'Class unless it applies to a tagged type or primitive subprogram of a tagged type.	
(17/3)	Legality		BD11001	All	A language-defined aspect cannot be specified on a renames.	We just try a few simple cases where the aspect would have been allowed on the original declaration.
			BD11001	All	A language-defined aspect cannot be specified on a generic formal parameter.	We just try a few simple cases where the aspect would have been allowed on the original declaration.
(18/4)	Legality	Wording modified by AI12-0105-1, intent of rule is unchanged.	BD11001	Part	A language-defined aspect cannot be specified on the completion of a subprogram.	Still need to try on body_stubs acting as a completion.
(18.1/4)	StaticSem	All boolean aspects of types are listed at right. Rule moved here by AI12-0138-1.			Check that if a derived type inherits Pack as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Volatile as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Atomic as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Independent as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Volatile_Components as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Atomic_Components as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Independent_Components as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Discard_Names as True from an ancestor, specifying it as False is illegal. Check that if a derived type inherits Unchecked_Union as True from an ancestor, specifying it as False is illegal.	B-Test: should be in C.6, as this is annex specific. B-Test: should be in C.6, as this is annex specific. B-Test: should be in C.6, as this is annex specific. B-Test: should be in C.6, as this is annex specific. B-Test: should be in C.6, as this is annex specific. B-Test: should be in C.6, as this is annex specific.
(18.2/4)	Definitions	“nonoverridable” Added by AI12-0138-1.				Note: Import and Export are boolean, but are never inherited so this rule doesn't apply to them. Default_Value and Default_Component_Value can be Boolean, but they explicitly disclaim this rule.
(18.3/4)	Legality	Added by AI12-0138-1.			Check that a descendant of a type with Implicit_Dereference specified can specify a confirming value for the aspect. Check that a descendant of a type with Constant_Indexing specified can specify a confirming value for the aspect.	C-Test. Not very important, it won't happen in usual use of the aspect. C-Test. Not very important, it won't happen in usual use of the aspect.

(18.4/4)	Legality	Added by AI12-0138-1.			4	Check that a descendant of a type with Variable_Indexing specified can specify a confirming value for the aspect.	C-Test. Not very important, it won't happen in usual use of the aspect.	
					4	Check that a descendant of a type with Default_Iterator specified can specify a confirming value for the aspect.	C-Test. Not very important, it won't happen in usual use of the aspect.	
					4	Check that a descendant of a type with Iterator_Element specified can specify a confirming value for the aspect.	C-Test. Not very important, it won't happen in usual use of the aspect.	
			Negative	B415001	All		Check that a descendant of a type with Implicit_Dereference specified cannot specify a nonconfirming value for that aspect.	
			Negative	B416001 (case F)	Part	5	Check that a descendant of a type with Constant_Indexing specified cannot specify a nonconfirming value for that aspect.	B-Test. Still need to try in the visible and private parts of an instance, a renames that renames the entity in question.
			Negative	B416001 (case F)	Part	5	Check that a descendant of a type with Variable_Indexing specified cannot specify a nonconfirming value for that aspect.	B-Test. Still need to try in the visible and private parts of an instance, a renames that renames the entity in question.
			Negative			6	Check that a descendant of a type with Default_Iterator specified cannot specify a nonconfirming value for that aspect.	B-Test. Try a renames that renames the entity in question.
			Negative			6	Check that a descendant of a type with Iterator_Element specified cannot specify a nonconfirming value for that aspect.	B-Test. Try a renames that renames the entity in question.
					4	Check that Implicit_Dereference can be specified for the full view of a private type if the partial view does not have discriminants.	C-Test. Not very important, it won't happen in usual use of the aspect.	
					4	Check that Constant_Indexing can be specified for the full view of a private type if the partial view is untagged.	C-Test. Not very important, it won't happen in usual use of the aspect.	
					4	Check that Variable_Indexing can be specified for the full view of a private type if the partial view is untagged.	C-Test. Not very important, it won't happen in usual use of the aspect.	
					4	Check that Default_Iterator can be specified for the full view of a private type if the partial view is untagged.	C-Test. Not very important, it won't happen in usual use of the aspect. Note: We can't test the case where the partial view is non-indexable but tagged, because the full view would either be illegal by this rule or it too would not be indexable.	
		4	Check that Iterator_Element can be specified for the full view of a private type if the partial view is untagged.	C-Test. Not very important, it won't happen in usual use of the aspect. Note: We can't test the case where the partial view is non-indexable but tagged, because the full view would either be illegal by this rule or it too would not be indexable.				
		4	Check that Implicit_Dereference cannot be specified for the full view of a private type or private extension if the partial view has known discriminants.	B-Test. Probably in 4.1.5. Still need to check inside a generic.				
		5	Check that Constant_Indexing cannot be specified for the full view of a private type if the partial view is tagged.	B-Test. Probably belongs in 4.1.6. Check when the partial view has specified the aspect, and check inside a generic.				

Negative	B416001 (case B)	Part	5 Check that Variable_Indexing cannot be specified for the full view of a private type if the partial view is tagged.	B-Test. Probably belongs in 4.1.6. Check when the partial view has specified the aspect, and check inside a generic.
Negative			6 Check that Default_Iterator cannot be specified for the full view of a private type if the partial view is indexable.	B-Test. Probably belongs in 5.5.1. Check when there is no aspect on the partial view, as well as when the partial view has specified the aspect. Note: Tagged but not indexable is illegal for the indexing aspects.
Negative			6 Check that Iterator_Element cannot be specified for the full view of a private type if the partial view is indexable.	B-Test. Probably belongs in 5.5.1 (its specific to this aspect). Check when there is no aspect on the partial view, as well as when the partial view has specified the aspect. Note: Tagged but not indexable is illegal for the indexing aspects.
			4 Check that Implicit_Dereference can be inherited for the full view of a private type if the partial view inherits or specifies the same value.	C-Test. Not very likely to get wrong.
			4 Check that Constant_Indexing can be inherited for the full view of a private type if the partial view inherits or specifies the same value.	C-Test. Not very likely to get wrong.
			4 Check that Variable_Indexing can be inherited for the full view of a private type if the partial view inherits or specifies the same value.	C-Test. Not very likely to get wrong.
			4 Check that Default_Iterator can be inherited for the full view of a private type if the partial view inherits or specifies the same value.	C-Test. Not very likely to get wrong.
			4 Check that Iterator_Element can be inherited for the full view of a private type if the partial view inherits or specifies the same value.	C-Test. Not very likely to get wrong.
			Check that Implicit_Dereference cannot be inherited for the full view of a private type if the partial view has known discriminants and does not inherit or specify the same value of the same aspect.	
Negative	B415001	All	6 Check that Constant_Indexing cannot be inherited for the full view of a private type if the partial view is tagged and does not inherit or specify the same value of the same aspect.	B-Test. Probably belongs in 4.1.6 (it's aspect-specific).
Negative			6 Check that Variable_Indexing cannot be inherited for the full view of a private type if the partial view is tagged and does not inherit or specify the same value of the same aspect.	B-Test. Probably belongs in 4.1.6 (it's aspect-specific).
Negative			6 Check that Default_Iterator cannot be inherited for the full view of a private type if the partial view does not inherit or specify the same value of the same aspect.	B-Test. Probably belongs in 5.5.1 (it's aspect-specific).
Negative			6 Check that Iterator_Element cannot be inherited for the full view of a private type if the partial view does not inherit or specify the same value of the same aspect.	B-Test. Probably belongs in 5.5.1 (it's aspect-specific).
Negative			6 Check that Iterator_Element cannot be inherited for the full view of a private type if the partial view does not inherit or specify the same value of the same aspect.	B-Test. Probably belongs in 5.5.1 (it's aspect-specific).

(18.5/4)	Legality	Negative	Added by AI12-0138-1.	6 Check that an instance is illegal if an actual type has Implicit_Dereference specified, and it is specified for a derived type that inherits from the corresponding formal type.	B-Test. Probably belongs in 4.1.5 (it's aspect-specific).
		Negative		6 Check that an instance is illegal if an actual type has Constant_Indexing specified, and it is specified for a derived type that inherits from the corresponding formal type.	B-Test. Probably belongs in 4.1.6 (it's aspect-specific).
		Negative		6 Check that an instance is illegal if an actual type has Variable_Indexing specified, and it is specified for a derived type that inherits from the corresponding formal type.	B-Test. Probably belongs in 4.1.6 (it's aspect-specific).
		Negative		6 Check that an instance is illegal if an actual type has Default_Iterator specified, and it is specified for a derived type that inherits from the corresponding formal type.	B-Test. Probably belongs in 5.5.1 (it's aspect-specific).
		Negative		6 Check that an instance is illegal if an actual type has Iterator_Element specified, and it is specified for a derived type that inherits from the corresponding formal type.	B-Test. Probably belongs in 5.5.1 (it's aspect-specific).
(18.6/4)	Redundant		Added by AI12-0138-1.		
(19/3)	StaticSem	Portion	Lead-in for the following.		
(20/3)	StaticSem	Subpart	This just says that the aspect_definition is interpreted and evaluated as a name for some aspects. Test for the individual aspects.		
(21/3)	StaticSem	Subpart	This just says that the aspect_definition is interpreted and evaluated as an expression for some aspects. Test for the individual aspects.		
(22/3)	StaticSem	Subpart	This just says that the aspect_definition is interpreted as an identifier specific to the aspect for some aspects. Test for the individual aspects.		
(23/3)	StaticSem	Portion	Lead-in for the following.		
(24/3)	StaticSem			5 An aspect specified on an object_declaration is view-specific.	C-Test. In particular, an object passed by reference may have other values for the aspects (Size, Alignment). Not particular critical to test.
(25/3)	StaticSem			3 An aspect specified on a subprogram_declaration is view-specific.	C-Test. But is this testable (as there are no aspects on bodies)? This seems to imply that aspects can be different on a renames (but that also doesn't allow any aspects).
(26/3)	StaticSem			1 An aspect specified on a renaming_declaration is view-specific.	Would be a C-Test, but as there are no aspects that can be used on a renaming declaration, this is useless to test.

(27/3)	StaticSem				<p>7 An aspect specified on a type applies to all views of the type.</p> <p>An aspect specified on a subtype applies to all views of the subtype.</p> <p>An aspect specified on a package applies to all views of the package.</p>	<p>C-Test. Check that attributes Size, Alignment are the same for any view, including (package) renames and private types. Try various kinds of types, too. This may exist somewhere.</p> <p>C-Test, but is there a way to get another view of a subtype? A subtype declaration makes a new subtype.</p> <p>B-Test: check that a library-level renaming of a Pure package is still a pure package (can't be withed by a normal package).</p>
(28/4)	StaticSem	Portion		Lead-in for the following. Modified by AI12-0106-1 to define the term “class-wide aspect” and to make it clear that the following rules can be overridden. This requires no additional testing.		
(29/3)	StaticSem	Subpart		Test as part of specific aspects (Type_Invariant'Class, Input'Class?).		
(30/3)	StaticSem	Subpart		Test as part of specific aspects (Pre'Class, Post'Class).		
(31/3)	StaticSem	Subpart		Test as part of specific aspects (like Size, Alignment, Address, etc.)		
(32/4)	1	StaticSem	Subpart	All such pragmas as now defined as aspects, so tests for the individual aspects will test this.		
			Subpart	Test for each individual aspect.		
			Subpart	Test for each individual aspect.		
			Subpart	Added by AI12-0154-1. Test with each individual aspect.		
(33/3)	StaticSem	Subpart		Just a statement that there are additional kinds of aspects.		
(34/4)	Deleted			Moved to 13.1.1(18.1/4) as this is a Legality Rule, it should be under that heading.		
(35/3)	StaticSem				<p>Check that if Variable_Indexing is specified in the private part, index notation is not supported on objects whose nominal subtype is the (untagged) partial view.</p> <p>Check that if Constant_Indexing is specified in the private part, index notation is not supported on objects whose nominal subtype is the (untagged) partial view.</p> <p>Check that if Implicit_Dereference is specified in the private part, generalized references are not supported on objects whose nominal subtype is the (undiscriminated) partial view.</p> <p>Check that if Default_Iterator and Iterator_Element is specified in the private part, component element iterators are not supported on objects whose nominal subtype is the (untagged) partial view.</p>	<p>B-Test. Note that most forms of hiding these are illegal, we only care about the legal ones.</p> <p>B-Test. Note that most forms of hiding these are illegal, we only care about the legal ones.</p> <p>B-Test. Note that most forms of hiding these are illegal, we only care about the legal ones.</p> <p>B-Test. Note that most forms of hiding these are illegal, we only care about the legal ones.</p>

(36/3)	StaticSem	Not Testable	A permission to override these rules; test for any specific aspects that do so.		
(37/3)	Dynamic			Check that aspect_definitions are evaluated at the freezing point of the associated entity, not at the point of the aspect_specification.	C-Test. Try aspects that can have dynamic values, like Storage_Pool and Storage_Size.
(38/3)	Impl-Def	Not Testable	This is a permission to support other sorts of aspects, even with different syntax.	8	

Paragraphs:		Objectives with tests:	Objectives to test:	Total objectives:	Objectives with submitted tests:
1	44	21	69	83	0
Must be tested		Objectives with Priority 10	0		
		Objectives with Priority 9	0		
Important to test		Objectives with Priority 8	1		
		Objectives with Priority 7	1		
Valuable to test		Objectives with Priority 6	26		
		Objectives with Priority 5	13		
Ought to be tested		Objectives with Priority 4	17		
		Objectives with Priority 3	7		
Worth testing		Objectives with Priority 2	1		
Not worth testing		Objectives with Priority 1	3		
		Total:	69		
		Objectives covered by new tests since ACATS 2.6	21		
		Completely:	14		