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

Clauses 8.3.1-8.5.3

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).
8.3.1	(1/2)		General								
	(2/2)		Syntax								
	(3/3)		Legality			**B831001**, ***C831001***	Part	2	Check that an overriding indicator can be given on an abstract subprogram declaration, a null procedure declaration, and an ordinary (non-protected) subprogram declaration.	C-Test. Still need to try abstract operation in a C-Test, but it isn't very likely to get wrong. Possibly do this in an interface test.	
						B831001	Part	5	Check that an overriding indicator can be given on an subprogram body, subprogram body stub, and a subprogram renaming declaration.	C-Test. Only tested error cases.	
						B831001	Part	4	Check that an overriding indicator can be given on a generic instantiation of a subprogram.	C-Test. Only tested error cases.	
					Added by Ada 2012, AI05-0177-1.	***C831001***	All		Check that an overriding indicator can be given on an expression function.		
	(4/2)		Legality	Subpart	Any overriding indicator C-Test will test this.						
				Negative		**B831001**	All		Check that an operation with an overriding indicator is illegal if it is not a primitive operation for some type.		
	(5/2)		Legality	Subpart	Any overriding indicator C-Test will test this.						
				Negative		**B831002**	All		Check that an operation with an indicator of **overriding** is illegal if it does not override a homograph at the place of the declaration or body.		
				Negative		**B831003**	All		Check that an operation with an indicator of **overriding** is illegal if it does not override a homograph at the place of the declaration or body even though the operation is overridden later.		
	(6/2)		Legality	Subpart	Any overriding indicator C-Test will test this.						
				Negative		**B831002**	All		Check that an operation with an indicator of **not overriding** is illegal if it overrides a homograph at the place of the declaration or body.	B-Test.	
						B831003	All		Check that an operation with an indicator of **not overriding** is illegal if it overrides a homograph even if the operation is overridden later.	B-Test. Try types where operations are revealed at multiple places.	
	(7/2)		Legality			***B831004***	Part	6	Check that overriding indicators can be used on operations primitive for a type derived from a generic formal type.	Still need a C-Test.	
				Negative	Instances are not relevant for this objective.	***B831004*** (specifications), ***B831005*** (bodies)	All		Check that an operation with the indicator of **overriding** is illegal if it is primitive for a type derived from a generic formal type and the operation does not inherit a homograph.		

			Negative	Instances are relevant for this objective, checks on instantiation are needed.	***B831004*** (specifications), ***B831005*** (bodies)	All		Check that an operation with the indicator of **not overriding** is illegal if it is primitive for a type derived from a generic formal type and the operation inherits a homograph in either the generic or the instance.	Note that this cannot be checked in private parts, as 12.3(18) says that such operations are not overriding in an instance even though they would normally be overriding.
	(8/2)	NonNormative		A note					
	(9/2)	NonNormative		Start of examples...					
	(10/2)	NonNormative							
	(11/2)	NonNormative							
	(12/2)	NonNormative							
	(13/2)	NonNormative							
	(14/2)	NonNormative							
	(15/2)	NonNormative							
	(16/2)	NonNormative		End of examples.					
8.4	(1)	Redundant							
	(2)	Syntax							
	(3)	Syntax							
	(4/3)	Syntax		**All** added by Ada 2012, AI05-0150-1.					
			Negative	Subtype_Mark needs an explicit check for subtypedness.	B840001			Check that the name in a use type clause cannot denote anything other than a subtype.	
	(5/2)	Legality	Widely Used	Any legal use clause.					
			Negative		B84001A (task decl, subp decl), ***B840002*** (record type, protected type, record object)	All		Check that the name in a use package clause cannot denote anything other than a package.	
			Negative		BC1012A (nested in subprogram), ***B840002*** (context clause)	All		Check that the name in a use package clause cannot denote a generic package.	
			Negative		***B840002***	All		Check that the name in a use package clause cannot denote the limited view of a package.	
				After AARM 5.a. This really ought to be tested in 12.7, but as we don't have objectives for that yet, we'll put it here to ensure it doesn't get missed.			4	Check that the name in a use package clause can denote a formal package.	C-Test.
	(6)	StaticSem		Context clause visibility is tested in 10.1.6, we don't test that use clauses don't apply in a context clause here.	CA1108A, CA13001		2	Check that a use clause given in a context clause of a specification applies to the body and subunits as well as the specification.	C-Test. Try a use type clause and a use all type clause.

							3	Check that a use clause given in a context clause of a body applies to the any subunits as well as the body.	C-Test. Try both use and use [all] type.
							4	Check that a use clause given in a context clause of a library package specification applies to child units.	C-Test. Try both use and use [all] type. Don't forget the child unit body.
			Negative				5	Check that a use clause given in a context clause of a library package specification P does not apply in any units that mention P in a context clause..	B-Test: Try both use package and use type, also use all type. Try withing P in specifications, bodies, and stubs, also check in bodies where P is given on the spec, and in stubs where P is given on the body.
(7)		StaticSem			C84008A		2	Check that a use clause in the visible part of a package specification applies to the body and any subunits as well.	C-Test. Try a use type clause (also use all type).
							3	Check that a use clause in the private part of a package specification applies to the body and any subunits as well.	C-Test. Try both a use package and use [all] type clause.
							4	Check that a use clause in the visible part of a package specification applies to any child units.	C-Test. Try both a use package and use [all] type clause. Don't forget the child unit body.
							2	Check that a use clause in a body applies to any subunits as well.	C-Test.
					B840001 (use type only, no bodies)		3	Check that a use clause in the private part of a package specification applies to all of private child units, and the private part and body of public child units.	C-Test. Try both a use package and use [all] type clause. Don't forget the child unit body.
			Negative		B84007A		2	Check that a use clause does not apply before its declaration.	B-Test. Try a use type clause (also use all type).
			Negative		B840001 (use type)		4	Check that a use clause given in the private part of a package does not include the public part of a public child unit.	B-Test. Try a use package clause.
			Negative		B84008B			Check that a use package clause for package P does not make items visible that were visible in P due to a use clause in P's visible part.	
(7.1/2)		Definitions		named					
(8/3)		Definitions		potentially use visible					
	1						4	Check that a use package clause for a library package makes any withed child units directly visible.	C-Test.
			Negative				3	Check that a use package clause does not make entities declared in nested packages directly visible.	B-Test.
	2						6	Check that a use type clause on a class-wide type T'Class makes the primitive operators of type T directly visible.	C-Test. Important for "=" (other operators are less likely).
			Negative		B840001		2	Check that a use type clause does not make primitive subprograms of the appropriate type that are named with identifiers directly visible.	B-Test. Check that enumeration literals are not made visible, as well as functions with arguments.
			Negative		B840001			Check that a use type clause does not make primitive operators for other types visible.	
			Negative		C840001			Check that a use type clause does not make non-primitive operators declared in the package where the named subtype is declared directly visible.	
	3		Lead-in	Added by Ada 2012, AI05-0150-1.					
(8.1/3)				Added by Ada 2012, AI05-0150-1.	**C840002**	All		Check that a use all type clause makes primitive subprograms of the appropriate type directly visible.	
			Negative		***B840003***	All		Check that a use all type clause does not make non-primitive subprograms declared in the package where the named subtype is declared directly visible.	

	(8.2/3)				Added by Ada 2012, AI05-0150-1.	**C840002**	All		Check that a use all type clause of a specific tagged type makes appropriate class-wide operations directly visible.	
				Negative		***B840003***	All		Check that a use all type clause of a specific tagged type T does not make operations of T'Class directly visible unless they are declared in the same package as T or an ancestor of T.	
	(8.3/3)		Definitions		Others kinds of “use-visible”; tested in 12.6, added by AI05-0131-1.					
	(9)		StaticSem	Portion	Lead-in for following bullets.					
	(10)		StaticSem	Subpart	Any legal use clause.					
				Negative		C84002A (proc declared later)		2	Check that a use package clause does not make an entity visible within the immediate scope of a homograph.	C-Test: try operations declared before the use clause and operators.
				Negative				3	Check that a use type clause does not make an operator visible within the immediate scope of a homograph.	C-Test: it's necessary to check which operator is executed.
	(11)		StaticSem	Subpart	Any legal use clause.					
						C84005A			Check that a use package clause can make overloaded subprograms with the same identifier visible, and that they can be resolved.	
						C84009A (use package)		1	Check that a use clause can make overloaded operators visible.	C-Test: use type clauses. (But unlikely to be wrong)
				Negative		B84004A, B84006A			Check that multiple declarations with the same identifier that are not overloadable are not made directly visible by one or more use clauses.	This can happen only for use package; operators are always overloadable.
	(12)		Dynamic	Not Testable	Can't tell "no effect" from forgetting to execute it; can't guess random wrong effects.					
	(13)		NonNormative		Start of examples...					
	(14)		NonNormative							
	(15)		NonNormative							
	(16)		NonNormative		...end of examples.					
8.5	(1)		Redundant							
	(2)		Syntax							
	(3)	1	Dynamic		Can't test this for exceptions, packages, or generics, because their names have no dynamic component.			3	Check that the name in an object renaming is evaluated each time it is elaborated.	C-Test.
								4	Check that the name in an object renaming is evaluated and needed index and access checks are performed.	C-Test.
								3	Check that the name in a subprogram renaming is evaluated and needed index and access checks are performed.	C-Test: try renaming access-to-subprogram objects stored in arrays or heap-allocated objects.
		2	Redundant							
	(4)		NonNormative		Start of examples...					

	(5)		NonNormative								
	(6)		NonNormative								
	(7)		NonNormative		...end of examples.						
8.5.1	(1)		Redundant								
	(2/3)		Syntax		Aspect_clauses added by Ada 2012.						
					This is likely to be a common mistake, so it is tested.	B85001I, B85001J, B85001K, B85001M			Check that the subtype_mark in a renaming declaration cannot be replaced by a subtype_indication.		
	(3/2)	1	NameRes					7	For an object renaming with a subtype_mark, check that the name is resolved if there is only one interpretation with the correct type, even if other interpretations exist.	C-Test.	
				Negative		B85001H		7	For an object renaming with a subtype_mark, check that the name is illegal if it does not resolve to the appropriate type.	B-Test. Make sure that X : T; Y : T'Class renames X is tested.	BY30001 (contains named access cases)
		2			Rule confirmed by AI05-0105-1.	***C851002***	All		For an object renaming with an anonymous access-to-object type, check that the name is resolved if there is only one interpretation with a correct anonymous access type, even if other interpretations exist.		
				Negative		***B851002***	All		For an object renaming with an anonymous access-to-object type, check that the name is illegal if it does not resolve to an anonymous access type with the appropriate designated type.		
		3			Rule confirmed by AI05-0105-1.	***C851002***	All		For an object renaming with an anonymous access-to-subprogram, check that the name is resolved if there is only one interpretation with a correct anonymous access type, even if other interpretations exist.		
				Negative		***B851003***	All		For an object renaming with an anonymous access-to-subprogram type, check that the name is illegal if it does not resolve to an anonymous access type with the appropriate designated profile.		
	(4)			Widely Used							
				Negative		B85001A, B85001B, B85001C, B85001D, B85001E			Check that an object renaming cannot rename a literal or aggregate.		
						B85001G (attrib)		4	Check that an object renaming cannot rename a value.	B-Test. Check named numbers, other attributes. (Enumeration literals tested by B85001F.)	
						B85001F			Check that an object renaming cannot rename something that is not an object.		
	(4.1/2)			Portion	This is the lead-in for the following rules.						
	(4.2/2)							5	Check that an object renaming with an anonymous access-to-object type can rename an object with the same kind of anonymous access-to-object.	C-Test. Two cases in the next objective's test (C851001).	CY30001 (three cases), CY30002 (two cases).

				C851001	All		Check that an object renaming with an anonymous access-to-object type with no null exclusion can rename an object with an anonymous access-to-object with a matching designated subtype and a null exclusion.	
		Negative		***B851002***	All		For an object renaming with an anonymous access-to-object type, check that the renaming is illegal if the designated subtypes don't statically match.	
				B851002	All		For an object renaming with an anonymous access-to-object type, check that the renaming is illegal if one of the types is access-to-constant and the other is access-to-variable.	
(4.3/2)						5	Check that an object renaming with an anonymous access-to-subprogram type can rename an object with the same kind of anonymous access-to-subprogram.	C-Test. A single case in C851001.
				B851003	All		For an object renaming with an anonymous access-to-object type, check that the renaming is illegal if the designated profiles are not subtype conformant.	
(4.4/2)		Portion	This is the lead-in for the following rules.					
(4.5/2)		Subpart	Any renaming of a formal object in a generic body.					
		Negative		***B851004*** (simple cases)	Part	6	For an object renaming with a null_exclusion given in a generic body that names a formal object of the generic or a parent unit of the generic, check that the renaming is illegal if the formal object does not have a null_exclusion.	B-Test. Be sure to check bodies of nested and child generics as well the body of the generic. Especially try cases that would otherwise be legal (the formal object having a null excluding subtype).
				B851004 (simple cases)	Part	6	For an object renaming with an access_definition with a null_exclusion given in a generic body that names a formal object of the generic or a parent unit of the generic, check that the renaming is illegal if the formal object does not have a null_exclusion.	B-Test. Be sure to check bodies of nested and child generics as well the body of the generic. Especially try cases that would otherwise be legal (the formal object having a null excluding subtype).
(4.6/2)	1	Subpart	Any renaming with a null_exclusion.					
		Negative		***B851004***	All		For an object renaming with a null_exclusion, check that the renaming is illegal if the subtype of the renamed object does not exclude null.	
				B851004	All		For an object renaming with an access_definition with a null_exclusion, check that the renaming is illegal if the subtype of the renamed object does not exclude null.	
	2			***B851004***	All		For an object renaming with a null_exclusion that renames a formal object in a generic package specification, check that an instance is illegal if the subtype of the actual object does not exclude null.	
				B851004	All		For an object renaming with an access_definition with a null_exclusion that renames a formal object in a generic package specification, check that an instance is illegal if the subtype of the actual object does not exclude null.	

(5/3)	1		This rule was revised by AI05-0008.	B3A2015 (definite, deref of general access), B851001 (generic body rule for formal derived), B85002A (definite, variant), B85003A (definite, variant, formal in out), B85003B (definite, formal in out)		9	Check that a renamed object is not a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained unless the object is known to be constrained.	B-Test. Check that the object being a constant (via a dereference of an access-to-constant) does not necessarily make it known-to-be-constrained. Check that pool-specific types are excluded if they have a constrained partial view. Check the special rules in generic bodies (formal access, formal private).
		Negative				9	Check that a renamed object can be a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained and which is known to be constrained.	C-Test. Be sure to check all of the cases for known-to-be-constrained. Check immutably limited types, indefinite types, parts of constants other than deref of access-to-constants.
	2					7	Check that a slice is not renamed if it is a slice of a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained and which is not known to be constrained.	B-Test. Be sure to check all of the cases that aren't known-to-be-constrained (definite, deref of general access inc. access-to-constant, deref of pool-specific with constrained partial view). Check special formal body cases.
		Negative				7	Check that a slice can be renamed if it is a slice of a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained and which is known to be constrained.	C-Test. Be sure to check all of the cases for known-to-be-constrained. Check immutably limited types, indefinite types, parts of constants other than deref of access-to-constants.
	3			B851001 (definite)		7	For a renamed object in a generic unit that is a subcomponent that depends on discriminants of an object, check that an instance is illegal if the object's nominal subtype is unconstrained and the object is not known to be constrained.	B-Test. Check deref of access-to-formal; check formal private types.
(6/2)				C85005G (range constraints), C85006G (index constraints)		6	Check that the constraints of a renamed object are those of the renamed object, not those given in the renaming declaration.	C-Test. Try discriminant constraints.
				C851001	All		Check that when renaming an object that excludes null, the renamed object still excludes null even if the renaming_declaration does not include a null_exclusion.	
				B85004A		6	Check that a renamed constant still is treated as a constant.	B-Test. Check function results, dereferences of access-to-constant types (named and anonymous), constant extended return statements, selected and indexed components of a constant.

					B85005A (obj dec), B85005B, B85005C (in out param), B85005D (generic in out), B85005E (allocator), B85006A (comp or slice of obj dec), B85006B, B85006C (comp or slice of in out param), B85006D (comp or slice of generic in out), B85006E (comp or slice of allocator), B85006F (slice of slice), B85007E (out param)		Check that a renamed variable can be assigned to.	
					B85004B		Check that a renamed object has the correct value.	
					B85005F (access deref)		Check that the renamed object remains the same even if the name that it renames changes to designate a different object.	C-Test. Try renaming an array item, and changing the index value.
	(7)	NonNormative		Start of examples...				
	(8)	NonNormative		...end of examples.				
8.5.2	(1)	Redundant						
	(2/3)	Syntax		Aspect_clauses added by Ada 2012.				
	(3)	Legality	Subpart	Any legal exception renaming.				
			Negative		B85008F, B85008G, B85008H	3	Check that the renamed entity of an exception renaming declaration denotes an exception.	B-Test. Check protected units, and various types: task, protected, decimal, float, fixed, integer, modular, record, array, private, interface.
	(4)	StaticSem			C85009A	4	Check that a renamed exception can be used anywhere that an exception can be used.	C-Test. Try in raise with message and as the prefix of 'Identity.
	(5)	NonNormative		Start of example...				
	(6)	NonNormative		...end of example.				
8.5.3	(1)	Redundant						
	(2/3)	Syntax		Aspect_clauses added by Ada 2012.				
	(3)	Legality	Subpart	Any legal package renaming				
			Negative		B85010A, B85010B (literals)	7	Check that the renamed entity of a package renaming declaration denotes a package.	B-Test. Try renaming a generic package, a task unit, a protected unit, an object declaration (of a task unit?), a parameter, a block label, a loop label, and various kinds of types (enum, integer, modular, fixed, decimal, float, array, record, private, interface). Note that all important cases are not currently tested.

(3.1/2)	Legality				8	Check that a limited view of a package can be renamed as a package, and that it can be used in it's immediate scope and within the scope of a with clause for the package.	C-Test. Note: AdaCore submitted test thinks this is illegal, so the priority was raised a bit.
		Negative			8	Check that the name of a renamed limited view of a package cannot be used outside of the scope of a with clause for the package or the immediate scope of the renaming.	B-Test.
(4)				B85011A		Check that a renamed package name can be used in the same ways as a normal package name.	
(4.1/2)	Redundant		This is formally defined in 8.3.				
(5)	NonNormative		Start of example...				
(6)	NonNormative		...end of example.				

	Paragraphs:		Objectives with tests:	Objectives to test:	Total objectives:	Objectives with submitted tests:
6	71		64	45	86	2

Must be tested	Objectives with Priority 10	0
	Objectives with Priority 9	2
Important to test	Objectives with Priority 8	2
	Objectives with Priority 7	6
Valuable to test	Objectives with Priority 6	6
	Objectives with Priority 5	4
Ought to be tested	Objectives with Priority 4	9
	Objectives with Priority 3	8
Worth testing	Objectives with Priority 2	7
Not worth testing	Objectives with Priority 1	1
	Total:	45

Objectives covered by new tests since ACATS 2.6	34
Completely:	28