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

Clauses 4.5.7 – 4.5.8

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).
4.5.7	(1/3)		General								
	(2/3)		Syntax								
	(3/3)		Syntax								
	(4/3)		Syntax								
	(5/3)		Syntax								
	(6/3)		Syntax								
	(7/3)		Syntax		This is a syntax rule, but we test it because it's in English and it is so weird.	***B457006***	Part	4	Check that parentheses can be omitted around an if expression if it is in a context in which it is already surrounded by parentheses.	C-Test. Test singleton parameters, qualified expressions, type conversions, singleton indexing, pragma Assert, expression functions, and a singleton generic instance. Also possible for attribute parameters, which is too weird.	
								3	Check that parentheses can be omitted around a case expression if it is in a context in which it is already surrounded by parentheses.	C-Test. Test singleton parameters, qualified expressions, type conversions, singleton indexing, pragma Assert, expression functions, and a singleton generic instance. Also possible for attribute parameters, which is too weird.	
				Negative		***B457006***	All		Check that an if expression has to be surrounded in parentheses if it is not already surrounded by them.	B-Test. Try in parameter lists with multiple parameters, indexing with multiple indexes, and in generic instances with multiple parameters.	Base on B457006?? Not very likely to be different.
								6	Check that a case expression has to be surrounded in parentheses if it is not already surrounded by them.		
	(8/3)		NameRes			***C457006*** (enum literals)	Part	7	Check that overloaded functions can be resolved when they appear as dependent expressions in an if expression.	C-Tests. Still need to try functions and operators, see C457006 for the pattern.	
						C457006 (enum literals)	Part	7	Check that overloaded functions can be resolved when they appear as dependent expressions in a case expression.	C-Tests. Still need to try functions and operators, see C457006 for the pattern.	
						C457007	All		Check that literals can be resolved when they appear as dependent expressions in an if expression.		
						C457007	All		Check that literals can be resolved when they appear as dependent expressions in a case expression.		
	(9/3)		NameRes	Portion	Lead-in for following rules						
	(10/3)		NameRes					8	Check that an if expression used as the operand of a type conversion effectively distributes the conversion to each dependent expression.	C-Test. Try cases where the dependent expressions have different but convertible types.	
								7	Check that a case expression used as the operand of a type conversion effectively distributes the conversion to each dependent expression.	C-Test. Try cases where the dependent expressions have different but convertible types.	
	(11/3)		NameRes		This does not appear testable separately; the paragraph 8/3 tests check the interesting cases.						
	(12/3)		NameRes		This does not appear testable separately; the paragraph 8/3 tests check the interesting cases.						

(13/3)		NameRes		This does not appear testable separately; the paragraph 8/3 tests check the interesting cases.					
			Negative	If none of the above cases apply, the conditional expression does not resolve.	***B457007***	All		Check that if the type of an if expression is not determined by the resolution rules, it is illegal.	
			Negative		***B457007***	All		Check that if the type of a case expression is not determined by the resolution rules, it is illegal.	
(14/3)		NameRes		This rule was moved from 5.3, it is an Ada 83 rule.	C87B42A		2	Check that a condition resolves if the overloading includes only one solution involving boolean types.	C-Test. The existing test only tries while, if, and exit. Try other cases: entry barrier, guard, if expression (both if and elsif), elsif in an if statement, for Ada 202x, loop filter. Low priority: unlikely to find a problem here.
			Negative				3	Check that a condition does not resolve if there is not exactly one solution involving boolean types.	B-Test. Try cases where there are no solutions, and cases where there are multiple solutions. Try in all condition contexts (if statement, if expression, exit statement, while loop, guard, entry barrier, for Ada 202x, iteration filter).
(15/3)		NameRes			***C457004***	All		Check that the selecting_expression of a case statement can be resolved if it is an overloaded function call, of which exactly one has a discrete type.	
							4	Check that the choices of a case expression have the type of the selecting_expression.	C-Test. Borrow from the case statement tests for this objective?
			Negative	Based on the decision of AI12-0040-1; could check in 8.6, but it really relates here.	***B860001***	All		Check that the selecting_expression of a case expression cannot be resolved if information from the choices is required to resolve it.	
(16/3)		Legality	Widely-used	Any C-Test of a conditional expression will test.					
			Negative				7	Check that an if expression is illegal if any of the dependent expressions cannot be converted to the type of the expression.	B-Test. Try various legality rules related to conversion, like access-to-constant vs. access-to-variable, and accessibility. All 4.6 rules are available when the conditional expression is nested in a type conversion.
			Negative				6	Check that a case expression is illegal if any of the dependent expressions cannot be converted to the type of the expression.	B-Test. Try various legality rules related to conversion, like access-to-constant vs. access-to-variable, and accessibility. All 4.6 rules are available when the conditional expression is nested in a type conversion.
(17/3)	1	Legality	Subpart	Positive cases are listed under line 2.					
			Negative		***B457005***	All		Check that if the expected type of a if expression is a specific tagged type, then if some (but not all) of the dependent expressions are dynamically tagged, the expression is illegal.	
			Negative		***B457005***	All		Check that if the expected type of a case expression is a specific tagged type, then if some (but not all) of the dependent expressions are dynamically tagged, the expression is illegal.	
	2						4	Check that the dependent expressions of an if expression can be dynamically tagged, and that the expression can be used in a context that requires a dynamically tagged expression.	C-Test.

								Check that the dependent expressions of a case expression can be dynamically tagged, and that the expression can be used in a context that requires a dynamically tagged expression.	4	C-Test.
								Check that the dependent expressions of an if expression can be tag-indeterminate, and that the expression can be used in a context that requires a tag-indeterminate expression.	4	C-Test.
								Check that the dependent expressions of a case expression can be tag-indeterminate, and that the expression can be used in a context that requires a tag-indeterminate expression.	4	C-Test.
								Check that the dependent expressions of an if expression can be statically tagged, and that the expression can be used in a context that requires a statically tagged expression.	3	C-Test.
								Check that the dependent expressions of a case expression can be statically tagged, and that the expression can be used in a context that requires a statically tagged expression.	3	C-Test.
(18/3)		Legality				**C457002 (one case),** ***B457003***	All	Check that the else part can be omitted from a boolean if expression, and it has the value True.		Don't have a C-Test for a non-Boolean boolean type, but that seems hardly usage-oriented.
			Negative			***B457003***	All	Check that if the type of an if expression is not a boolean type, the else part cannot be omitted.		
(19/3)		Legality	Widely-used	Legal case expressions will of course meet these rules. We note the particular rules below.						
			Negative	5.4(5/3), sentence 1.		**B457001** (dynamic predicates), ***B457004*** (variables, subtypes)	All	Check that a case expression is illegal if any of the choices is non-static.		
			Negative	5.4(5/3), sentence 2.		***B457002***	All	Check that a case expression is illegal if others is not the last choice or does not stand alone.		
			Negative	5.4(6-9/3).		**B457001** (static predicates), ***B457004*** (non-predicate cases)	All	Check that a case expression is illegal if any of the possible values of the selecting expression are not covered by a choice.		
			Negative	5.4(10).		**B457001** (static predicates), ***B457004*** (non-predicate cases)	All	Check that a case expression is illegal if more than one choice covers the same value.		
(20/3)		Dynamic				**C457001, C457002**	All	Check that for the evaluation of an if expression, the condition specified after if, and any conditions specified after elsif, are evaluated in succession, until one evaluates to true.		
						C457001, C457002	All	Check that result of an if expression is the result of evaluating the dependent expression corresponding to the condition that evaluates to True.		
(21/3)	1	Dynamic						Check that the evaluation of a case expression starts by evaluated the selecting expression.	5	C-Test. Pretty basic stuff.
	2					**C457003,** ***C457005*** (others, ignored predicates)	All	Check that result of an case expression is the result of evaluating the dependent expression corresponding to the choice selected by the selecting expression.		
						C457003	All	Check that only the selected dependent expression is evaluated in a case expression, along with the selecting expression.		

		3				**C457003**	All		Check that if the value of the selecting expression is not covered by any case alternative, Constraint_Error is raised.	This is almost untestable, since it is impossible to create an invalid value without making the program erroneous. But we can play tricks with uninitialized objects, as in this test. A better, but limited, way is in the next objective.
									If the selecting expression of a case expression is a name with a static nominal subtype and has a static predicate, the case statement does not have an others clause, and the static predicate is disabled, then Constraint_Error is raised if the value of the selecting expression does not satisfy the predicate.	
						C457005	All			
	(22/5)		NonNormative		An example added by AI12-0312-1					
	(23/5)		NonNormative		An example added by AI12-0312-1					
4.5.8	0.1/4		General		Added by AI12-0158-1.					
	(1/3)		Syntax							
	(2/3)		Syntax							
	(3/3)		Syntax							
	(4/3)		Syntax		This is a syntax rule, but we test it because it's in English and it is so weird.	***B458001***	Part	4	Check that parentheses can be omitted around a quantified expression if it is in a context in which it is already surrounded by parentheses.	C-Test. Test singleton parameters, qualified expressions, type conversions, singleton indexing, pragma Assert, and a singleton generic instance. Also possible for attribute parameters, which is too weird and the wrong type.
				Negative		***B458001***	All		Check that a quantified expression has to be surrounded in parentheses if it is not already surrounded by them.	
	(5/3)		NameRes					7	Check that the predicate of a quantified expression can be resolved if it is an overloaded function call.	C-Test. Try overloaded functions that return a boolean and non-boolean type.
				Negative				4	Check that the predicate of a quantified expression cannot have a different type than the entire quantified expression.	B-Test. The type has to be some Boolean type, so it isn't very likely – thus the test isn't important.
	(6/4)		Dynamic		Modified by AI12-0158-1, objectives unchanged.	**C458001**	All		Check that the predicate of a quantified expression is evaluated in the order specified by the loop_parameter_specification.	
						C458002	All		Check that the predicate of a quantified expression is evaluated in the order specified by an array component iterator specification.	
						C458A01	All		Check that the predicate of a quantified expression is evaluated in the order specified by a generalized iterator specification.	
						C458A02	All		Check that the predicate of a quantified expression is evaluated in the order specified by a container element iterator specification.	
	(7/3)		Dynamic	Portion	This is lead-in text.					
	(8/4)	1,2	Dynamic		Modified by AI12-0158-1, objectives unchanged.	**C458001** (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All		Check that if the quantifier is all, the result of a quantified expression is True if the predicate is True for all values and False otherwise.	

				C458003 (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All	Check that if the quantifier is all, the result of a quantified expression is True if there are no values in the domain.
	3			**C458001** (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All	Check that evaluation of predicates stops for a quantified expression with a quantifier of all when a predicate evaluates to False.
	4			**C458003** (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All	Check that any exceptions propagated by the predicate of a quantified expression with a quantifier of all is propagated by the quantified expression.
(9/4)	1, 2	Dynamic	Modified by AI12-0158-1, objectives unchanged.	**C458001** (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All	Check that if the quantifier is some, the result of a quantified expression is True if the predicate is True for some value and False otherwise.
				C458003 (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All	Check that if the quantifier is some, the result of a quantified expression is False if there are no values in the domain.
	3			**C458001** (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All	Check that evaluation of predicates stops for a quantified expression with a quantifier of some when a predicate evaluates to True.
	4			**C458003** (normal for loop), **C458002** (array component iterator), **C458A01** (generalized iterator), **C458A02** (container iterator)	All	Check that any exceptions propagated by the predicate of a quantified expression with a quantifier of some is propagated by the quantified expression.
(10/3)		NonNormative	Start of examples...			
(11/3)		NonNormative				
(12/3)		NonNormative				
(13/5)		NonNormative	End of examples.			

Paragraphs:		Objectives with tests:	Objectives to test:	Total objectives:	Objectives with submitted tests:
2	37	39	22	56	1

	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	5

Valuable to test	Objectives with Priority 6	2
	Objectives with Priority 5	1
Ought to be tested	Objectives with Priority 4	8
	Objectives with Priority 3	4
Worth testing	Objectives with Priority 2	1
Not worth testing	Objectives with Priority 1	0
	Total:	22

Objectives covered by new tests since ACATS 2.6	38
Completely:	34