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

Clauses 5.4 – 5.5.2

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).
5.4	(1)		Redundant		Case expression rules are tested in 4.5.7; we don't need to test them here.						
	(2/3)		Syntax								
	(3)		Syntax								
	(4/3)	1	NameRes			C540001 (integer, formal integer, formal derived integer, formal derived enum), C54A03A (ints, enums), C54A04A (type with partial views), **C540002** (modular, formal modular, int, formal int, enum)	All		Check that the selecting_expression can be of any discrete type	C-Test. Formal discrete would work, but no static choices can be written, which makes it pointless to test.	
					This objective should have been tested in a legacy test named C87B43x, (based on the Implementors Guide including it as one of 4 objectives under 8.7(B), T43) but no such test exists. (C87B43A tests the 4th of the 4 objectives).	**C540002**	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.		
				Negative		B54A05A (String, private), B54A05B (Fixed)		4	Check that the selecting_expression cannot be of a non-discrete type	B-Test. Check record types, array types, task types, protected types, access types, and float types. Also generic private types (even if actual is discrete).	
						B54A60A (Ints, Enums), B54A60B (Character literal)		2	Check that the selecting_expression has to resolve to a single solution (it cannot be two overloaded discrete functions).	B-Test. Try cases involving modular types.	
	2					C87B43A (int operators)		3	Check that the choices of a case statement can be resolved if it involves an overloaded function call.	C-Test. Need to try enumeration literals, and modular operators. If we ever add user-defined static functions, then we need to revisit the priority of this objective and include such cases (especially for enumerations).	
				Negative		B54A10A (Different integer types), B54A20A (Boolean vs. Integer)		3	Check that the discrete_choices must be of the type of the selecting_expression.	B-Test. Need test for overloaded enumerations, and for modular types.	
	(5/3)	1	Legality		Any legal case statement will test this.						
				Negative		B54A21A (Integer, Boolean)		3	Check that non-static discrete choices are illegal.	B-Test. Need test for user-defined enumerations, modular types, and in generics.	

	2	Redundant		This normatively stated in 3.8.1(8/3). But we test it here for case statements.	B54A01L			Check that others must appear alone and last in a case statement.		B-Test.
(6/3)		Legality	Subpart	Lead-in for following.						
(7/4)		Legality	Widely Used	The majority of case expressions will test this.						
			Negative		B54A12A (Integer), B54A25A (Enumerations), B540002 (Modular)	All		If the selecting expression of a case statement is a name with a static nominal subtype, then no discrete_choice can cover any value outside of the range of the subtype.		
			Negative	AI12-0071-1 slightly changes the wording but makes no material change to the semantics.	**B540001**	All		If the selecting expression of a case statement is a name with a static nominal subtype with a static predicate, then no discrete_choice can cover any value that does not satisfy its predicates.		
			Negative		B54B01C (Formal In, Integer), B54B04A (Objects, Integer, Boolean), B54B06A (Enum literal, Enums), **B540002** (Modular)	All		If the selecting expression of a case statement is a name with a static nominal subtype and no predicates, and there is no others choice, then every value of the range of the subtype must be covered by some discrete_choice.		
			Negative	AI12-0071-1 slightly changes the wording but makes no material change to the semantics.	**B540001**	All		If the selecting expression of a case statement is a name with a static nominal subtype and has a static predicate, and there is no others choice, then every value that satisfies the predicates of the subtype must be covered by some discrete_choice.		
(8/3)		Legality			C540001 (formal integer, formal derived integer, formal derived enum)		3	Check that a case statement with an others clause and a selecting_expression with type selecting_expression is root_integer, universal_integer, or a descendant of a formal scalar type works as expected.		C-Test. Need test for root integer and universal integer.
			Negative				5	Check that a case statement with a selecting_expression with type selecting_expression is root_integer, universal_integer, or a descendant of a formal scalar type is illegal if there is no others choice.		B-Test. The old Ada 95 coverage document lists this as Nothing New, but this rule was not in Ada 83. There are a couple of cases like this in the ancient tests, but they don't try this directly. Try to cover the entire range from System.Min_Int to System.Max_Int, others still required.
(9/3)		Legality	Widely Used	Many case statements will try this.	C54A13D (Ints, Enums)		3	If the selecting_expression of a case statement is an expression, then any value of the base range of the type may appear as a choice.		C-Test. Try cases involving modular types.
					C54A13A (Vars, Ints), C54A13B (Generic in or in out, Ints), C54A13C (Qualified, Type Conversion, Ints)		3	If the selecting_expression of a case statement is a name with a non-static subtype, then any value of the base range of the type may appear as a choice.		C-Test. Try cases involving modular types.
			Negative		B54B02B (Integer, Enums), B54B05A (Ints), **B540002** (Modular)	All		If the selecting_expression of a case statement is an expression, and there is no others choice, then every value of the base range of the type must be covered by some discrete choice.		

					B54B02B (Objects, Integer, Enums), B54B02C (Generic in), B54B02D (Generic in out), B54B04B (Objects, Integer, Enums), **B540002** (Modular)	All	If the selecting_expression of a case statement is a name with a non-static subtype, and there is no others choice, then every value of the base range of the type must be covered by some discrete choice.		
	(10)	Legality			B54A20A (Integer, Character), **B540002** (Modular)	All	Check that two discrete_choices of a case_statement may not cover the same value.		
	(11/3)	Dynamic	Widely Used	Any case statement will test this.					
	(12/3)	Dynamic	Widely Used	Any case statement will test this.	C54A42A, C54A42B (compact range), C54A42C (wide range), C54A42D (wide range), C54A42E (biased small range), C54A42F (wide range, others), C54A42G (ranges, others), ***C540003*** (ignored predicates, others)		3 Check that the correct case alternative is selected.	These tests try to ensure that both jump table and repeated if implementations are needed and tested. C-Test: Try case statements with modular types.	
					C54A24A (Ints), C54A24B (Chars)		Check that null choices can occur in case statements.	This follows from 3.8.1 rules. It's weird enough that the existing test objectives need to be preserved.	
	(13)	Dynamic	Other than the case of static predicates, this should only happen for invalid values, which we can't generate on demand, so other cases aren't testable.		***C540003***	All	If the selecting expression of a case statement 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.		
	(14)	NonNormative	A note.						
	(15)	NonNormative	Examples						
	(16)	NonNormative							
	(17)	NonNormative							
	(18)	NonNormative							
5.5	(1)	Redundant							
	(2)	Syntax							
	(3/3)	Syntax							
	(4)	Syntax							
	(5)	Syntax	This is really a Legality Rule.		C57004A (Normal for loops), C57004B (Normal for loops)		3 If a loop_statement has a loop_statement_identifier that matches the one on the end loop, the loop properly executes.	C-Test. Need tests for unconditional and while loops as well as new forms of loop.	
					B55A01A		3 If a loop_statement does not have a loop_statement_identifier, there cannot be an identifier following end loop.	B-Test. We still need tests for the new kinds of loops.	

			Negative		B55A01A		3	If a loop_statement has a loop_statement_identifier, then the loop is illegal if there is no identifier after end loop, or one that is different.	B-Test. We still need tests for the new kinds of loops.
(6/5)		Definitions		“loop parameter” and its subtype. Clarified by AI12-0061-1, post-Corrigendum, not not really changed.	B55B12B			Check that the subtype of a loop parameter is determined properly by using it in a case statement.	
					C55B11A			Check that the type of a loop parameter is determined properly by assigning it.	
(7)		Dynamic		This is probably common, but it ought to be explicitly tested somewhere.			4	Check that a loop without an iteration scheme executes until it is left by a transfer of control.	C-Test. Unlikely to be wrong, but no tests can be found. Try all forms of control flow (exit, goto, exception).
(8)		Dynamic			C55C02A (false conditions), C55C02B (evaluation)			Check that a while loop condition is evaluated each time through the loop, and loop is complete if the condition is False.	
(9/4)	1	Dynamic	Portion	The next sentence provides testable information.					
	2	Dynamic			C55B04A (Integer)		3	Check that the discrete_subtype_definition is evaluated at the start of a for loop.	C-Test. This should be tried for other types (at least enumeration and modular, also generic formal discrete, integer, and modular, possibly also character types and Boolean). Low priority because it's unlikely to be wrong.
	3	Dynamic			C55B04A (Integer)		3	Check that if the discrete subtype of a for loop identifies a null range, the loop body is not executed.	C-Test. This should be tried for other types (at least enumeration and modular, also generic formal discrete, integer, and modular, possibly also character types and Boolean). Low priority because it's unlikely to be wrong.
	4	Dynamic	Portion	Tested below.					
	5	Dynamic			C55B03A (Integer), C450001 (Modular, only number of iterations)		3	Check that the loop parameter is assigned values in ascending order for a normal for loop and the loop body is executed once for each value when the subtype does not have a predicate.	C-Test. This should be tried for other types (at least enumeration and modular, also generic formal discrete, integer, and modular, possibly also character types and Boolean). Low priority because it's unlikely to be wrong.
				AI12-0071-1 slightly changes the wording but makes no material change to the semantics.	***C550001***	All		Check that the loop parameter is assigned only values that satisfy the predicate, in ascending order for a normal for loop and the loop body is executed once for each value when the subtype has a static predicate.	
					C55B03A (Integer), C450001 (Modular, only number of iterations)		3	Check that the loop parameter is assigned values in descending order for a reverse for loop and the loop body is executed once for each value when the subtype does not have a predicate.	C-Test. This should be tried for other types (at least enumeration and modular, also generic formal discrete, integer, and modular, possibly also character types and Boolean). Low priority because it's unlikely to be wrong.
				AI12-0071-1 slightly changes the wording but makes no material change to the semantics.	***C550001***	All		Check that the loop parameter is assigned only values that satisfy the predicate, in descending order for a reverse for loop and the loop body is executed once for each value when the subtype has a static predicate.	

	6	Dynamic	Portion	Tested above.				
(9.1/3)		Redundant		Just a pointer to the following sections.				
(10)		NonNormative		A note.				
(11)		NonNormative		Another note.				
(12)		NonNormative		Another note.				
(13)		NonNormative		Part of the previous note.				
(14)		NonNormative		Start of examples				
(15)		NonNormative						
(16)		NonNormative						
(17)		NonNormative						
(18)		NonNormative						
(19)		NonNormative						
(20)		NonNormative						
(21)		NonNormative		End of examples.				
5.5.1								
(1/3)		StaticSem	Subpart	Lead-in for the package.				
(2/3)		StaticSem	Portion	We don't directly test the contents of this (or any) package, but we will test it's use in other rules.				
(3/3)		StaticSem	Portion					
(4/3)		StaticSem	Portion					
(5/3)		StaticSem	Portion					
(6/3)	1	Definitions		“iterator type”				
	2			“reversible iterator type”				
	3			“iterator object”				
	4			“reversible iterator object”				
	5			“iteration cursor subtype”				
(7/3)		StaticSem			***B551001***	All	Check that a Default_Iterator aspect cannot be specified on an untagged type nor on a type that does not have one of the indexing aspects.	
					B551002	All	Check that a Iterator_Element aspect cannot be specified on an untagged type nor on a type that does not have one of the indexing aspects.	

(8/3)	1	StaticSem			***B551001***	All		Check that name of a Default_Iterator aspect cannot denote an entity other than a function declared in the same declaration list as the type declaration.	
					B551001	All		Check that the name specified by a Default_Iterator aspect cannot denote a function with zero parameters.	
					B551001	All		Check that the name specified by a Default_Iterator aspect cannot denote a function whose first parameter has a type other than T or T'Class or an access-to-object designating T or T'Class.	
					B551001	All		Check that the name specified by a Default_Iterator aspect cannot denote a function whose other parameters are not defaulted.	
					B551001	All		Check that the name specified by a Default_Iterator aspect cannot denote a function whose result type is other than an iterator type.	
					B551001	All		Check that the name specified by a Default_Iterator aspect cannot denote multiple functions that meet all of these requirements.	
					B551001	Part	5	Check that the name specified by a Default_Iterator aspect can denote a function whose parameters beyond the first are defaulted, and that the result can be used in an iterator.	C-Test. Existing test tries the declaration, we also need to test it in a loop.
	2	Definitions		“default iterator function”					
	3	Definitions		“default iterator subtype”					
	4	Definitions		“default cursor subtype”					
(9/3)	1	StaticSem			***B551002***	All		Check that name of a Iterator_Element aspect cannot denote an entity other than a subtype.	
	2	Definitions		“default element subtype”					
(10/3)		StaticSem					6	Check that aspects Default_Iterator and Iterator_Element are inherited by descendants of a type for which it is specified.	C-Test: check that container iterators work on such a type. We can't test these separately.
(11/3)	1	Definitions		“iterable container type”					
	2	Definitions		“reversible iterable container type”					
	3	Definitions		“iterable container object”					
	4	Definitions		“reversible iterable container object”					
(11.1/4)		StaticSem	Subpart	Added by AI12-0138-1. The rules are enumerated in 13.1.1(18.2-5/4).					
(12/3)		Legality	Portion	Lead-in for following.					
(13/3)		Legality	Subpart	Correct containers will test					
			Negative				6	Check that an iterable container type T is illegal if there is no Constant_Indexing function whose result type is covered by the default element type of T or is a reference type designating a type covered by the default element type of T.	B-Test. Careful: No Constant_Indexing at all is legal.
(14/3)		Legality	Subpart	Correct containers will test					
			Negative				6	Check that an iterable container type T is illegal if there is no Constant_Indexing function whose second parameter covered the default cursor type of T.	B-Test. Careful: No Constant_Indexing at all is legal.

	(15/3)		Legality						
				Negative					
				Negative					
	(16/3)		Definitions		“default constant indexing function”				
	(17/3)		Legality	Portion	Lead-in for following.				
	(18/3)		Legality	Subpart	Correct containers will test				
				Negative					
	(19/3)		Legality	Subpart	Correct containers will test				
				Negative					
	(20/3)		Legality						
				Negative					
				Negative					
	(21/3)		Definitions		“default variable indexing function”				
5.5.2	(1/3)		General						
	(2/5)		Syntax		Generalized by AI12-0156-1.				
	(2.1/5)		Syntax		Added by AI12-0156-1.				
	(3/3)	1	NameRes						
					We cannot check a “not a type” objective here, as this the same syntax as a normal for loop.	B552A02	All		

	Check that an iterable container type T is legal and can be used if the only matching Constant_Indexing function has more than two parameters where all of the extra parameters have defaults.	C-Test. Check that the order that the values are defined in the predicate is immaterial.
4		
	Check that an iterable container type T is illegal if all Constant_Indexing functions have 3 or more parameters without defaults.	B-Test. Careful: No Constant_Indexing at all is legal.
6		
	Check that an iterable container type T is illegal if there are more than one Constant_Indexing function that matches the rules for the default constant indexing function of T.	B-Test.
6		
	Check that an iterable container type is legal even if Constant_Indexing is not specified.	C-Test. Try in an iterator. In this case, Variable_Indexing must be specified.
5		
	Check that an iterable container type T is illegal if there is no Variable_Indexing function whose result type is a reference type designating a type covered by the default element type of T.	B-Test. Careful: No Variable_Indexing at all is legal.
6		
	Check that an iterable container type T is illegal if there is no Variable_Indexing function whose second parameter covers the default cursor type of T.	B-Test. Careful: No Variable_Indexing at all is legal.
6		
	Check that an iterable container type T is legal and can be used if the only matching Variable_Indexing function has more than two parameters where all of the extra parameters have defaults.	C-Test. Check that the order that the values are defined in the predicate is immaterial.
4		
	Check that an iterable container type T is illegal if there is all Variable_Indexing functions have 3 or more parameters without defaults.	B-Test. Careful: No Variable_Indexing at all is legal.
6		
	Check that an iterable container type T is illegal if there are more than one Variable_Indexing function that matches the rules for the default variable indexing function of T.	B-Test.
6		
	Check that an iterable container type is legal even if Variable_Indexing is not specified.	The Bingo_Balls foundation tests this.
	Check that the iterator_name of a generalized iterator can be of any iterator type.	C-Test.
5		
	Check that the iterator name of a generalized iterator can be resolved if it is an overloaded function call, of which exactly one has an iterator type.	C-Test.
5		
	Check that the iterator_name of a generalized iterator cannot be of a non-iterator type.	

	2	NameRes						5	Check that the iterable_name of an iterator can be of any array or iterable container type.	C-Test.
								5	Check that the iterable_name of an iterator can be resolved if it is an overloaded function call, of which exactly one has an array or iterable container type.	C-Test. Try at least one case of each kind of type.
					B552A02	All			Check that the iterable_name of an iterator cannot be of a type that is neither an array nor an iterable container type.	
					B552A02	All			Check that the iterable_name of an iterator cannot denote a type.	
	3	Definitions		“array component iterator”, “container element iterator”						
(4/3)	1	Definitions		“reverse iterator”, “forward iterator”						
	2	Legality			**B552A01, C552A01**	All			Check that “reverse” can be used in a generalized iterator if the type of the iterator_name is a reversible iterator type.	
			Negative		**B552A01**	All			Check that “reverse” cannot be used in a generalized iterator if the type of the iterator_name is not a reversible iterator type.	
	3	Legality			**B552A01, C552A02**	All			Check that “reverse” can be used in a container element iterator if the default iterator type of the type of the iterable_name is a reversible iterator type.	
					B552A01	All			Check that “reverse” cannot be used in a container element iterator if the default iterator type of the type of the iterable_name is not a reversible iterator type.	
(5/5)	1	Legality	Subpart	Added by AI12-0156-1. Any legal generalized iterator with a subtype indication will test.						
			Negative					1	Check that a generalized iterator is illegal if there is a subtype_indication, and it does not statically match the iteration cursor subtype.	B-Test. Added by Amendment 1, so ultra low priority until that comes out.
	2	Legality	Subpart	Any legal array component iterator with a subtype indication will test.						
								1	Check that a generalized iterator is illegal if there is an access_definition.	B-Test. This is never legal, as the actual type for a generic parameter cannot be anonymous. Added by Amendment 1, so ultra low priority until that comes out.
			Negative	Changed to static matching by AI12-0151-1.	***B552001***	All		1	Check that an array component iterator can have an access_definition that statically matches the component subtype of the type of the iterable_name.	C-Test. Added by Amendment 1, so ultra low priority until that comes out. Then medium priority; this is the case that could happen. We don't seem to have any other place to put this test.
									Check that an array component iterator is illegal if there is a subtype_indication, and it does not statically match the component subtype of the type of the iterable_name.	
								1	Check that an array component iterator is illegal if there is an access_definition, and it does not statically match the component subtype of the type of the iterable_name.	B-Test. Added by Amendment 1, so ultra low priority until that comes out.
	3	Legality	Subkind	Any legal container element iterator with a subtype_indication will test.						

								C-Test. Added by Amendment 1, so ultra low priority until that comes out. Then still low priority; this requires a custom, non-generic container type with anonymous access element. We don't seem to have a natural place to put this test.
						1	Check that a container element iterator is illegal if there is an access_definition, and it does not statically match the default element subtype of the type of the iterable_name.	
		Negative	Changed to static matching by AI12-0151-1.	***B552A04***	All		Check that a container element iterator is illegal if there is a subtype_indication, and it does not statically match the default element subtype of the type of the iterable_name.	
						1	Check that a container element iterator is illegal if there is an access_definition, and it does not statically match the default element subtype of the type of the iterable_name.	B-Test. Added by Amendment 1, so ultra low priority until that comes out. (And never very likely even then.)
(6/3)		Legality	Subkind	Any legal container element iterator on a constant will test.				
		Negative		***B552A04***	All		Check that a container element iterator is illegal if the iterable_name is a constant of type T, and Constant_Indexing is not specified for T.	
			If Variable_Indexing is not specified for an iterable container type, then Constant_Indexing must be, else Default_Iterator would be illegal as the type would not be indexable. As such, the second part of this rule is untestable (and is redundant).					
							Check that the iterator_name of a generalized iterator does not denote a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained and which is not known to be constrained.	
(6.1/4)		Legality	Rule added by AI12-0047-1.	***B552A05***	All			
				B552A04	All		Check that the iterable_name of a container element iterator does not denote a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained and which is not known to be constrained.	
				B552001	All		Check that the iterable_name of an array component iterator does not denote a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained and which is not known to be constrained.	
								B-Test: Other cases: Try an error in a default expression; and possibly an accessibility problem with the container object (if someone can figure out how to do that).
(6.2/4)		Legality	Rule added by AI12-0120-1.	***B552A04***	Part	4	Check that a container element iterator loop is illegal if the call to the default element iterator is illegal.	
(6.3/4)	1	Legality	Rule added by AI12-0120-1.	***B552A05***	All		Check that a generalized iterator loop is illegal if the cursor type is limited at the point of the loop.	
	2			***B552A04***	All		Check that a container element iterator loop is illegal if the default cursor subtype is limited at the point of the loop.	
(7/5)	1	Definitions	“loop parameter”					

	2	StaticSem					1	Check that the nominal subtype of the loop parameter for a generalized iterator is the named subtype when one is given.	C-Test. Added by Amendment 1, so ultra-low priority until that comes out, then Low priority as testing this requires using the loop parameter as a case selecting expression and thus a discrete cursor type. That is not a very likely usage, thus this test will not be very usage-oriented.
							6	Check that the nominal subtype of the loop parameter of an array component iterator is the named subtype when one is given.	C-Test. Make the array component a discrete type and use it in a case selecting expression.
							6	Check that the nominal subtype of the loop parameter of a container element iterator is the named subtype when one is given.	C-Test. Make the container element a discrete type and use it in a case selecting expression.
	2	StaticSem					3	Check that the nominal subtype of the loop parameter for a generalized iterator is the iteration cursor subtype if no subtype name is given.	C-Test. Low priority as testing this requires using the loop parameter as a case selecting expression and thus a discrete cursor type. That is not a very likely usage, thus this test will not be very usage-oriented.
	4	StaticSem					6	Check that the nominal subtype of the loop parameter of an array component iterator is the subtype of the array component if no subtype name is given in the iterator.	C-Test. Make the array component a discrete type and use it in a case selecting expression.
	5	StaticSem					6	Check that the nominal subtype of the loop parameter of a component element iterator is the subtype of the container element if no subtype name is given in the iterator.	C-Test. Make the container element a discrete type and use it in a case selecting expression.
(8/3)	1	StaticSem		**B552A03**	All			Check that the loop parameter of a generalized iterator cannot be assigned.	
	2	StaticSem		***C552001***	All			Check that the loop parameter of an array component iterator can be assigned if the array object is a variable view.	Would be nice to try more component types than just Integer.
				B552A03	All			Check that the loop parameter of an array component iterator cannot be assigned if the array object is a constant view.	
	3	StaticSem		**C552A02**	All			Check that the loop parameter of a component element iterator can be assigned if the container object is a variable view and the Variable_Indexing aspect has been specified for the type of the container object.	
				B552A03	All			Check that the loop parameter of a component element iterator cannot be assigned if the container object is a constant view.	
				B552A03	All			Check that the loop parameter of a component element iterator cannot be assigned if the type of the container object does not have Variable_Indexing specified.	
			From AI12-0093-1, AARM 5.5.2(8.a/4).				4	Check that the loop parameter of a generalized iterator is finalized when the loop is left.	C-Test. Careful: it also will be finalized when it is assigned for each iteration of the loop. Controlled cursors are pretty weird, so we give this a lower priority (as the test will not be very usage-oriented).
			From AI12-0093-1, AARM 5.5.2(8.a/4).				4	Check that the loop parameter of an array component iterator is not finalized when the loop is left or when it iterates.	C-Test. The components should only be finalized when the array object goes away (and as part of assignments to them).

				From AI12-0093-1.				4	Check that the loop parameter of a component element iterator is not finalized when the loop is left or when it iterates.
(9/3)		Dynamic						4	Check that the iterator specification is elaborated before the loop executes.
(10/3)	1	Dynamic			C552A01	All			Check that the iterator_name of a generalized iterator is evaluated exactly once at the start of the loop.
	2	Dynamic	Subpart	Tested on line 5.					
	3	Dynamic	Subpart	Tested on line 5.					
	4	Dynamic	Subpart	Tested on line 5.					
	5	Dynamic			C552A01	All			Check that the execution of a forward generalized iterator calls First initially, then Next until Has_Element returns False (assuming no transfer of control), executing the sequence of statements each time.
	6	Dynamic			C552A01	Part		3	Check that the execution of a forward generalized iterator calls First initially, then Next until the loop is left by a transfer of control, executing the sequence of statements each time.
					C552A01	All			Check that the execution of a forward generalized iterator for a reversible iterator type never calls Last or Previous.
					C552A01	All			Check that the execution of a forward generalized iterator never iterates or calls Next if Has_Element is initially False.
					C552A01	All			Check that the execution of a reverse generalized iterator calls Last initially, then Previous until Has_Element returns False (assuming no transfer of control), executing the sequence of statements each time.
					C552A01	Part		3	Check that the execution of a reverse generalized iterator calls Last initially, then Previous until the loop is left by a transfer of control, executing the sequence of statements each time.
					C552A01	All			Check that the execution of a reverse generalized iterator never calls First or Next.
					C552A01	All			Check that the execution of a reverse generalized iterator never iterates or calls Previous if Has_Element is initially False.

C-Test. The elements should only be finalized when the array object goes away (and as part of assignments to them).

C-Test. Make sure that it is elaborated only once per loop (not per iteration). Careful about the static matching requirement for the subtype_indication. Try all three kinds of loops.

C-Test. Test tries unconditional exit on first iteration, we should also try conditionally exiting on a later iteration, exiting via goto, and exiting via an exception. None of these are likely to be wrong if the test passes, so we give them a low priority. Could use the foundation to construct such a test.

C-Test. Test tries unconditional exit on first iteration, we should also try conditionally exiting on a later iteration, exiting via goto, and exiting via an exception. None of these are likely to be wrong if the test passes, so we give them a low priority. Could use the foundation to construct such a test.

				From AI12-0093-1, AARM 5.5.2(10.a/4).				5	If the cursor subtype is indefinite, Constraint_Error is raised by a generalized iterator if Next or Previous returns an object with a different constraint than the initial value.	C-Test. This is somewhat of a pathology, but we may want to test it as some people may try to misuse iterators this way.
				From AI12-0093-1, AARM 5.5.2(10.a/4).				5	If the cursor subtype is class-wide, Constraint_Error is raised by a generalized iterator if Next or Previous returns an object with a different tag than the initial value.	C-Test. This is somewhat of a pathology, but we may want to test it as some people may try to misuse iterators this way.
				From AI12-0093-1, AARM 5.5.2(10.b/4).				4	If the cursor subtype of an iterator type is limited, any attempt to use the iterator type in a generalized iterator is illegal.	B-Test. This is definitely a pathology, but some detection is necessary.
(11/3)	1	Dynamic			***C552001***	All			Check that the iterable_name of an array component iterator is evaluated exactly once at the start of the loop.	
	2	Dynamic			***C552001***	All			Check that the execution of an array component iterator is immediately complete if the array is a null array.	
	3	Dynamic	Subpart	Tested on line 4 and 5.						
	4	Dynamic	Subpart		***C552001***	All			Check that a forward array component iterator visits each component of a one-dimensional array exactly once, in the order of index values (first to last).	Would be nice to try more component types than just Integer.
					C552001	All			Check that a forward array component iterator for a two-dimensional array visits each component in canonical order, with the last dimension varying fastest.	Would be nice to try more component types than just Integer.
					C552002	All			Check that a forward array component iterator for a two-dimensional array with convention Fortran visits each component in canonical order, with the first dimension varying fastest.	We could check dynamic and parameter cases here, but hardly worth it.
					C552001	All			Check that a forward array component iterator properly visits elements for more than two dimensional arrays.	We could check dynamic and parameter cases here, but hardly worth it.
	5	Dynamic	Subpart		***C552001***	All			Check that a reverse array component iterator visits each component of a one-dimensional array exactly once, in the reverse order of index values (last to first).	Would be nice to try more component types than just Integer.
					C552001	All			Check that a reverse array component iterator for a two-dimensional array visits each component in canonical order, with the last dimension varying fastest.	Would be nice to try more component types than just Integer.
					C552002	All			Check that a reverse array component iterator for a two-dimensional array with convention Fortran visits each component in canonical order, with the first dimension varying fastest.	We could check dynamic and parameter cases here, but hardly worth it.
					C552001	All			Check that a reverse array component iterator properly visits elements for more than two dimensional arrays.	We could check dynamic and parameter cases here, but hardly worth it.
	6	Dynamic	Subpart	Tested above.						
(12/3)	1	Dynamic			**C552A02**	All			Check that the iterable_name of a container element iterator is evaluated exactly once at the start of the loop.	
	2	Dynamic			**C552A02**	All			Check that the default iterator function is evaluated for a container element iterator exactly once at the start of the loop and after the iterable name.	
	3	Dynamic						4	Check that a object of the default cursor subtype is created and initialized appropriately at the start of a container element iterator.	C-Test. But testing requires a weird cursor type with a controlled part, and is tough to test well. So a low priority.

(13/3)	1	Dynamic	Subpart	Tested on line 4 and 5.						
	2	Dynamic	Subpart	Tested on line 4 and 5.						
	3	Dynamic	Subpart	Tested on line 4 and 5.						
	4	Dynamic			C552A02	All		Check that the execution of a forward container element iterator calls First initially, then Next until Has_Element returns False (assuming no transfer of control), executing the sequence of statements each time.		
					C552A02	Part	3	Check that the execution of a forward container element iterator calls First initially, then Next until the loop is left by a transfer of control, executing the sequence of statements each time.	C-Test. Test tries unconditional exit on first iteration, we should also try conditionally exiting on a later iteration, exiting via goto, and exiting via an exception. None of these are likely to be wrong if the test passes, so we give them a low priority. Could use the foundation to construct such a test.
					C552A02	All		Check that the execution of a forward container element iterator never calls Last or Previous.		
					C552A02	All		Check that the execution of a forward container element iterator never iterates or calls Next if Has_Element is initially False.		
	5	Dynamic			C552A02	All		Check that the execution of a reverse container element iterator calls Last initially, then Previous until Has_Element returns False (assuming no transfer of control), executing the sequence of statements each time.		
					C552A02	Part	3	Check that the execution of a reverse container element iterator calls Last initially, then Previous until the loop is left by a transfer of control, executing the sequence of statements each time.	C-Test. Test tries unconditional exit on first iteration, we should also try conditionally exiting on a later iteration, exiting via goto, and exiting via an exception. None of these are likely to be wrong if the test passes, so we give them a low priority. Could use the foundation to construct such a test.
					C552A02	All		Check that the execution of a reverse container element iterator never calls Last or Previous.		
					C552A02	All		Check that the execution of a reverse container element iterator never iterates or calls Previous if Has_Element is initially False.		
	6	Dynamic			C552A02	All		Check that the loop parameter of a container element iterator denotes the default variable indexing using the current cursor of the container if the container object is a variable view and the Variable_Indexing aspect was specified for the container type.		
					C552A02	Part	4	Check that the loop parameter of a container element iterator denotes the default constant indexing using the current cursor of the container if the container object is a constant view or the Variable_Indexing aspect was not specified for the container type.	C-Test: still need to check the case of a container type that doesn't have Variable_Indexing.
					C552A02	All		Check that the default variable indexing of the container type is evaluated once per iteration of the loop for a container element iterator when the container is a variable view and the Variable_Indexing aspect was specified for the container type.		

			C552A02	Part	4	Check that the default constant indexing of the container type is evaluated once per iteration of the loop for a container element iterator when the container is a constant view or the Variable_Indexing aspect was not specified for the container type.	C-Test: still need to check the case of a container type that doesn't have Variable_Indexing.
(14/4)	Dynamic	Added by AI12-0120-1			4	Check that an exception propagated by a call or assignment executed as part of a container element iterator cannot be handled inside of the sequence_of_statements of the loop, but can be handled outside of the loop.	C-Test.
					4	Check that an exception propagated by a call or assignment executed as part of a generalized iterator cannot be handled inside of the sequence_of_statements of the loop, but can be handled outside of the loop.	C-Test.
(15/3)	NonNormative	An example. Paragraph number changed by AI12-0120-1.					
(16/3)	NonNormative	A reference to other examples.					

Paragraphs:		Objectives with tests:	Objectives to test:	Total objectives:	Objectives with submitted tests:
4 82		99	64	139	0

Must be tested	Objectives with Priority 10	0
	Objectives with Priority 9	0
Important to test	Objectives with Priority 8	0
	Objectives with Priority 7	0
Valuable to test	Objectives with Priority 6	13
	Objectives with Priority 5	9
Ought to be tested	Objectives with Priority 4	15
	Objectives with Priority 3	19
Worth testing	Objectives with Priority 2	1
Not worth testing	Objectives with Priority 1	7
	Total:	64

Objectives covered by new tests since ACATS 2.6	78
Completely:	70