

Coverage for ISO/IEC 8652:2012 and subsequent corrections in ACATS 3.x and 4.x
Clauses 4.1.3 - 4.4

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	Objective's		Objective Text	Objective notes	Submitted tests (will need work).
							New	Priority			
4.1.3	(1)		Redundant								
	(2)		Syntax								
	(3)		Syntax								
	(4)		Definitions	Widely Used	Expanded Name						
	(5)		NameRes						Check that if the prefix of a selected component denotes an enclosing construct, it is not interpreted as a component reference. 6	C-Test. Try F.C inside a a function F that returns a record R with a component C, while the function has an object C of the same type. This should resolve and not make a recursive call.	
									Check that if the prefix of a selected component denotes an enclosing protected type, it is not interpreted as an external reference to a protected entry or subprogram. 6	B-Test (?)	
	(6)		NameRes	Portion	Lead-in for next rule.				Check that if the prefix of a selected component denotes an enclosing construct, it is not interpreted as a prefix view. 6	B-Test (?) or a C-Test like the one described above.	
	(7)		NameRes			C41301A			Check that for the reference L.R, if R represents a component or discriminant of a record type, then L can represent an object or value of that type.		
				Negative					Check that for the reference L.R, if R represents a discriminant of a private, task, or protected type, then L can represent an object or value of that type. 4	C-Test. Try cases like those found in C41301A. Simple cases probably exist in many other ACATS tests, thus the low priority.	
	(8)		NameRes	Negative Portion	Lead-in for next rule.	B940005 contains a single example.			Check that for the reference L.R, if R represents a component of a protected type, and L represents an object or value of that type, the reference is illegal. 4	B-Test. Try many kinds of prefixes.	
	(9)		NameRes			C41306B (func, access-to-task), C41306C (func, access-to-task), <i>C413006</i> (not access)			Check that for the reference L.R, if L represents a task value or object, R can represent a task entry or family.		
	(9.1/2)		NameRes	Portion	Lean-in for next rule.				Check that for the reference L.R, if L represents a protected value or object, R can represent a protected entry or subprogram. 5	C-Test. Simple cases are scattered throughout the ACATS; we mainly need to test examples like those in C41306x.	
	(9.2/3)	1	NameRes		These objectives mostly cover the first three lines.	C413001		All	Check that for the reference L.R, if L represents an object or value of a tagged type T, that R may represent a subprogram with a first parameter of the type T that is declared immediately in the declarative region of an ancestor of T.		

	C413002	All	Check that for the reference L.R, if L represents an object or value of an access type designating a tagged type T, that R may represent a subprogram with a first parameter of the type T that is declared immediately in the declarative region of an ancestor of T.	
	C413001	All	Check that for the reference L.R, if L represents an object or value of a tagged type T, that R may represent a subprogram with a first parameter of a classwide type that covers T that is declared immediately in the declarative region of an ancestor of T.	
	C413002	All	Check that for the reference L.R, if L represents an object or value of an access type designating a tagged type T, that R may represent a subprogram with a first parameter of a classwide type that covers T that is declared immediately in the declarative region of an ancestor of T.	
	C413003	All	Check that for the reference L.R, if L represents an object or value of a tagged type T, that R may represent a subprogram with a first access parameter that designates T that is declared immediately in the declarative region of an ancestor of T.	
	C413004	All	Check that for the reference L.R, if L represents an object or value of an access type designating a tagged type T, that R may represent a subprogram with a first access parameter that designates T that is declared immediately in the declarative region of an ancestor of T.	
	C413003	All	Check that for the reference L.R, if L represents an object or value of a tagged type T, that R may represent a subprogram with a first access parameter that designates a classwide type that covers T that is declared immediately in the declarative region of an ancestor of T.	
Negative	C413004	All	Check that for the reference L.R, if L represents an object or value of an access type designating a tagged type T, that R may represent a subprogram with a first access parameter that designates a classwide type that covers T that is declared immediately in the declarative region of an ancestor of T.	
			Check that for the reference L.R, if L represents an object or value of an access type designating a tagged type T with the value null, and R represents an appropriate subprogram for a prefixed view, that Constraint_Error is raised when the name L.R is evaluated.	C-Test. Check that this happens even for a first access parameter that does not exclude null (this would have to be a classwide parameter). [This is required as this is a dereference.]
	B413004	Part	Check that for the reference L.R, if L represents an object or value of an non-access untagged type T or an access type designating an untagged type T, and R represents a subprogram with a first parameter of T, the reference is illegal even if the subprogram is primitive for T.	B-Test. Try other types, including protected, task, limited record, float, fixed, decimal, modular, enum. But this isn't very important.

			Negative		B413004	Part	<div>Check that for the reference L.R, if L represents an object or value of a non-access untagged type T or an access type designating an untagged type T, and R represents a subprogram with a first access parameter designating T, the reference is illegal even if the subprogram is primitive for T.</div> <div>2</div>	B-Test. Try other types, including protected, task, limited record, float, fixed, decimal, modular, enum. But this isn't very important.
2			Negative				<div>Check that for the reference L.R, if L represents an object of a tagged type T or an access type designating a tagged type T, and R represents a subprogram with a first parameter of the type T or a classwide type that is covered by T that is not declared immediately in the declarative region of an ancestor of T, the reference is illegal.</div> <div>8</div>	B-Test.
3			Negative				<div>Check that for the reference L.R, if L represents an object of a tagged type T or an access type designating a tagged type T, and R represents a subprogram with some parameter other than the first parameter of the type T and a first parameter of a non-access untagged type that is declared immediately in the declarative region of an ancestor of T, the reference is illegal.</div> <div>7</div>	B-Test.
4					B413001	All	<div>Check that the reference L.R is not interpreted as a prefixed view if the designator R represents a component of the type T visible at the point of the reference.</div>	
			Negative				<div>Check that the reference L.R can be interpreted as a prefixed view if the designator R represents a component of the type T that is not visible at the point of the reference.</div> <div>3</div>	C-Test. B431001 includes this case, which is why the priority is low.
5			Widely Used	A new rule in Ada 2012, necessary to allow ordinary Ada83-style prefix calls to tagged task and protected operations. We don't need to test this separately as any test of tagged task or protected types will necessarily make prefix calls.				
6	StaticSem	Subpart		Prefixed view calls are tested in 6.4(10.1/2).				
					C413005	All	<div>Check that a prefixed view is the name of a subprogram (with the first parameter omitted from the profile) that can be renamed and passed as a generic formal parameter.</div> <div>Check that a call of a prefixed view cannot repeat the first parameter in the parameter list.</div> <div>8</div>	B-Test.
6	Definitions		Negative	Prefixed view.				
(10)	NameRes	Portion		Lead-in for the following rules.				
(11)	NameRes	Subpart		Tested in the next two rules.				
(12)	NameRes						<div>Check that for the reference L.R, if L represents the name of a package, then R can name any visible declaration in the package.</div> <div>3</div>	C-Test: commonly used but no obvious test to report here.

(13.2/2)	Legality	Subpart	Prefixed view calls are tested in 6.4(10.1/2).			
		Negative		B413003	All	Check that the prefixed view L.R is illegal if the first parameter of R is a parameter with mode in out or out and L does not denote a variable.
		Negative		B413003	All	Check that the prefixed view L.R is illegal if the first parameter of R is a parameter with an access-to-variable type and L does not denote a variable.
(14)	Dynamic	Widely Used	Testing the evaluation of the name will necessarily test evaluation of the prefix.	C41304A		Check that L.R raises Constraint_Error when L has the access value null.
(15)	Dynamic			C41304B		Check that L.R raises Constraint_Error when L denotes a record object with discriminant values such that component R does not exist.
(16)	NonNormative		Start of examples.			
(17/2)	NonNormative					
(18)	NonNormative					
(19)	NonNormative		End of examples.			
<hr/>						
4.1.4	(1)	Redundant				
	(2)	Syntax				
				C41404A ('Image'First, etc.)	2	Check that the prefix of an attribute can be another attribute.
	(3)	Syntax				
	(4)	Syntax				
	(5)	Syntax				
				C41401A (Callable, Terminated, First, Last, Length, Range)	4	Check that the prefix of attributes that do not apply to objects of an access types can be interpreted as an implicit dereference.
	(6)	1	NameRes			C-Test. Test Component_Size, Constrained, Tag, and Valid. Test by determining that a prefix with the value null raises Constraint_Error (and resolves).
				C41402A (Address, Size, First_Bit, Last_Bit, Position)	4	Check that the prefix of attributes that apply to objects of an access types are not interpreted as an implicit dereference.
					5	Check that the prefix of attributes that apply to objects but not functions is interpreted as a parameterless function call.
					5	Check that the prefix of attributes that apply to both objects and functions is never interpreted as a parameterless function call.
						C-Test. Test Access, Alignment, Storage_Size, Unchecked_Access. Test by determining that a prefix with the value null does not Constraint_Error.
						C-Test. Test Callable, Terminated, First, Last, Length, Range, Size, First_Bit, Last_Bit, Position, Alignment, Storage_Size, Component_Size, Constrained, Tag, Valid.
						C-Test. The Address attribute: check that the function is not called. B-Test: The prefix of Access can't be a function for a access-to-object type. (Is this a good idea?)

				Negative	These are the only rules for most attributes.	B87B26A (First_Bit, Last_Bit, Position, Callable, Terminated, First, Last, Length, Range, Count)			4	Check that information about the kind of entity expected as the prefix of attributes without extra resolution rules is not used to resolve the prefix.	B-Test(s). Test any untested attributes for which an example can be made (all of the ones mentioned/tested for previous objectives are candidates; there probably are others). AARM 4.1.4(6.d-h) give some examples for the Valid attribute.	
(7)	NameRes								1	Check that the expression of a First, Last, Length, or Range attribute can be resolved even if there are interpretations of a non-integer type.	C-Test. I believe this objective cannot be tested because of the requirement that this expression be static (we need user-defined functions to get interesting overloading).	
(8)	Legality		Negative				B36201A has a single test case for Length.		2	Check that the expression of a First, Last, Length, or Range attribute must be static.	B-Test. Test the other three attributes, and try cases where the non-staticness isn't obvious (as for a constant defined of a generic formal integer type).	
(9/3)	1	StaticSem	Subpart	This is untestable by itself; it will be tested as part of testing each attribute.								
				2	Added by AI05-0006-1.				4	Check that a First, Last, First_Valid, or Last_Valid attribute can be used as the expression of a case statement, and coverage is required for the base subtype of its type.	B-Test.	
				3	Added by AI05-0006-1.				4	Check that a Pred, Succ, Val, or Input attribute can be used as the expression of a case statement, and coverage is required for the base subtype of its type.	B-Test.	
(10)	Redundant											
(11)	Dynamic		Subpart							3	For attributes designating objects, check that evaluating the attribute evaluates the prefix.	C-Test. This is slightly covered by the tests for the objectives of 4.1.4(6), thus the low priority.
(12/1)	Impl-Def		Not Testable	The only effect of this is that a test checking that Small is not defined for floating point types is incorrect.								
(13)	NonNormative				A note.							
(14/2)	NonNormative				Another note.							
(15)	NonNormative				Start of examples.							
(16)	NonNormative				End of examples.							
4.1.5	(1/3)	StaticSem	Portion	This entire subclause is new in Ada 2012. The rule is tested below.						Check that the name given for an Implicit_Dereference aspect must be that of an access discriminant for the associated type.		
				(2/3)	StaticSem	Negative	B415001	All				
						Subpart	This will necessarily be used in any C-Tests testing other rules here.					
				(3/3)	Definitions							
				(4/3)	Syntax							
				(5/3)	NameRes	Subpart	This will necessarily be used in any C-Tests testing other rules here.					

4.1.5	(5.1/4)	StaticSem	Subpart	Added by AI12-0138-1. The rules are enumerated in 13.1.1(18.2-5/4), and the objectives are there.				
	(6/3)	StaticSem		Not clear that any semantic effect of this beyond those caused by the dynamic rules (there doesn't seem to be any other sensible meaning).				
	(7/3)	1		Note: The “if not overridden” wording doesn't need to be tested, as 5.1/4 makes it illegal to override.			7 Check that a generalized reference can be used for an object of a derived type that inherits the Implicit_Dereference aspect from its parent type.	C-Test.
		2					6 Check that a generalized reference for a derived type refers to the new discriminant when that discriminant constrains an inherited reference discriminant.	C-Test.
		3	Redundant	Even though this is redundant (because it follows from the rules for constraining an inherited discriminant), we test it here as it's unlikely that such a combination would be tested elsewhere.			6 Check that a generalized reference for a derived type whose inherited reference discriminant is constrained refers to the constrained value.	C-Test.
	(8/3)	1	Dynamic				3 Check that the reference_object_name is evaluated by the evaluation of a generalized reference.	
		2	Dynamic		C415001	All	Check that Constraint_Error is raised by a generalized reference whose discriminant value is null.	C-Test. Try a name containing a function call. Low priority because it's hard to imagine a compiler getting this wrong, being the same as evaluating any other name that's part of an expression.
		3	Dynamic	Portion				
		4	Dynamic		C415001	Part	4 Check that a generalized reference denotes the object or subprogram designated by the value of the reference discriminant.	Need a C-Test that tries this for an access-to-subprogram discriminant (but much less important).
					C415001	All	Check that the object denoted by a generalized reference can be modified if the discriminant has an access-to-variable type.	
4.1.6					B415002	All	Check that the object denoted by a generalized reference cannot be used as a variable if the discriminant has an access-to-constant type.	
	(9/3)	NonNormative						
	(10/3)	NonNormative						
	(11/3)	NonNormative						
	(12/3)	NonNormative						
	(13/3)	NonNormative						
	(14/3)	NonNormative						
	(15/3)	NonNormative						
4.1.6	(1/3)	StaticSem	Negative	This entire subclause is new in Ada 2012.	B416001	All	Check that a Constant_Indexing or Variable_Indexing aspect can only be specified on a tagged type declaration.	
	(2/3)	1	StaticSem	Negative	B416001	Part	4 Check that name of a Constant_Indexing aspect cannot denote an entity other than a function declared in the same declaration list as the type declaration.	B-Test. Should try denoting the wrong kind of entity (procedures in particular).

						6	Check that the name of a Constant_Indexing aspect can denote entities in other scopes so long as at least one qualifying function exists in the same declaration list as the type declaration.	C-Test. We don't want other visible things to cause issues. Test in a child package where the parent makes conflicting things visible.
						4	Check that the name of a Constant_Indexing aspect can denote other kinds of entities in the same declaration list so long as at least one qualifying function exists in the same declaration list as the type declaration.	C-Test. (Procedures in particular.) Low priority as it doesn't seem particularly likely to occur in practice.
	2		Negative		B416001	All	Check that the name specified by a Constant_Indexing aspect cannot denote a function with zero or one parameters.	
			Negative		B416001	All	Check that the name specified by a Constant_Indexing aspect cannot denote a function whose first parameter has a type other than T or T'Class or an access-to-constant designating T or T'Class.	
					B416001, C416A01	All	Check that the name specified for a Constant_Indexing can refer to a set of overloaded functions.	
					B416001, C416A01	All	Check that the name specified for a Constant_Indexing can have more than two parameters.	
(3/3)	1	StaticSem	Negative		B416001	Part	4 Check that name of a Variable_Indexing aspect cannot denote an entity other than a function declared in the same declaration list as the type declaration.	B-Test. Should try denoting the wrong kind of entity (procedures in particular).
						6	Check that the name of a Variable_Indexing aspect can denote entities in other scopes so long as at least one qualifying function exists in the same declaration list as the type declaration.	C-Test. We don't want other visible things to cause issues. Test in a child package where the parent makes conflicting things visible.
						4	Check that the name of a Variable_Indexing aspect can denote other kinds of entities in the same declaration list so long as at least one qualifying function exists in the same declaration list as the type declaration.	C-Test. (Procedures in particular.) Low priority as it doesn't seem particularly likely to occur in practice.
	2		Negative		B416001	All	Check that the name specified by a Variable_Indexing aspect cannot denote a function with zero or one parameters.	
			Negative		B416001	All	Check that the name specified by a Variable_Indexing aspect cannot denote a function whose first parameter has a type other than T or T'Class or an access-to-variable designating T or T'Class.	
	3		Negative		B416001	All	Check that the name specified by a Variable_Indexing aspect cannot denote a function that returns a type other than a reference type for an access-to-variable.	
					B416001, C416A01	All	Check that the name specified for a Variable_Indexing can refer to a set of overloaded functions.	
					B416001, C416A01	All	Check that the name specified for a Variable_Indexing can have more than two parameters.	
(4/3)	1	StaticSem			C416A02	All	Check that a generalized indexing can be used for an object of a derived type that inherits the Constant_Indexing or Variable_Indexing aspect from its parent type.	
	2	StaticSem	Redundant	This sentence is deleted by AI12-0104-1.				
(5/3)		Definitions						

(5.1/4)	Static-Sem	Subpart	Added by AI12-0138-1. The rules are enumerated in 13.1.1(18.2-5/4).				
(6/5)	Legality		Removed by AI12-0138-1 (Corrigendum), replaced by AI12-0160-1 (post-Corrigendum).	1	Check that an aspect <code>Constant_Indexing</code> or <code>Variable_Indexing</code> is illegal if it is specified for a type derived from a tagged type with the other attribute specified.	B-Test. Try both attributes, try in both visible and private parts; but wait until the next document (Amendment?) is issued. (We're only interested in cases here that don't violate the nonoverridable rules.)	
(7/4)	Deleted		Removed by AI12-0138-1.				
(8/4)	Deleted		Removed by AI12-0138-1.				
(9/5)	Legality		Removed by AI12-0138-1 (Corrigendum), replaced by AI12-0160-1 (post-Corrigendum).	1	Check that an instance is illegal if it contains a derivation of a formal tagged type for which aspect <code>Constant_Indexing</code> or <code>Variable_Indexing</code> is specified, and the actual type is a tagged type with the other attribute specified.	B-Test. Try both attributes, try in both the visible and private parts; but wait until the next document (Amendment?) is issued.	
(10/3)	Syntax						
(11/3)	NameRes			2	Check that the prefix of a generalized indexing resolves if it denotes overloaded functions where one option is a function returning an indexable container type and the other function is not.	C-Test. Low priority because the other overloaded function has to return a type that cannot be any kind of prefix (else it probably would ambiguous for other reasons). It's also not clear that requiring this level of resolution is a good idea.	
(12/3)	NameRes	Portion	This is lead-in text.				
(13/3)	NameRes			5	Check that a generalized indexing calls the <code>Constant_Indexing</code> function if no <code>Variable_Indexing</code> is specified, even in a variable context.	C-Test. To try variable contexts, a <code>Constant_Indexing</code> function that returns an access-to-variable type is needed; a dereference then can be used in a variable context. Assume that the object being indexed is some sort of handle.	
					Check that a generalized indexing is illegal in a variable context if no <code>Variable_Indexing</code> is specified and <code>Constant_Indexing</code> specifies a function returning an ordinary object.		
			B416002	All			
				5	Check that a generalized indexing is illegal in a variable context if no <code>Variable_Indexing</code> is specified and <code>Constant_Indexing</code> specifies a function returning a reference type with an access constant discriminant.	B-Test. Not very important (unlikely the previous would work and this would fail), but could make a version of the existing test.	
(14/3)	NameRes			6	When both <code>Constant_Indexing</code> and <code>Variable_Indexing</code> are specified, check that the variable indexing function is called in variable contexts if the prefix is a variable, and that the constant indexing function is called in all other cases (including variable indexing contexts when the prefix is a constant).	C-Test. To try variable contexts, a <code>Constant_Indexing</code> function that returns an access-to-variable type is needed; a dereference then can be used in a variable context. Assume that the object being indexed is some sort of handle. (Existing test checks that the right routine is called when the prefix is a variable.)	C4160RB
					When both <code>Constant_Indexing</code> and <code>Variable_Indexing</code> are specified, check that a generalized indexing is illegal if it is called in variable contexts when the prefix is a constant and <code>Constant_Indexing</code> specifies a function returning an ordinary object.		
			B416A01	All			

					When both Constant_Indexing and Variable_Indexing are specified, check that a generalized indexing is illegal if it is called in variable contexts when the prefix is a constant and Constant_Indexing specifies a function returning a reference type with an access constant discriminant.		B-Test. Not very important (unlikely the previous would work and this would fail, also partially covered by the containers tests), but could make a version of the existing test.		
					When only a Variable_Indexing is specified, check that a generalized indexing with a prefix of a constant is illegal.		B-Test. (There's no fallover in this case, like there is in the others.)		
(15/3)	NameRes	Portion	Included in the other test objectives.						
(16/3)	NameRes	Portion	Included in the other test objectives.						
(17/3)	NameRes	Subpart	This is necessarily tested in any C-Test that uses a generalized indexing.						
(18/4)	NonNormative		Added by AI12-0104-1. We test these cases here as there is no other natural point to do so, and they're important.	C416A02	All	Check that if a function used by an inherited Constant_Indexing or Variable_Indexing is overridden, the overridden function is called by a generalized indexing.			
				C416A02	All	Check that if a function used by an inherited Constant_Indexing or Variable_Indexing is overloaded (with a different profile), the overloaded function can be called by a generalized indexing.			
(19/3)	NonNormative	Example; the paragraphs were renumbered by AI12-0104-1.							
(20/3)	NonNormative								
(21/3)	NonNormative								
(22/3)	NonNormative								
4.2	(1)	1	Redundant						
		2	Definitions Deleted	Widely Used	Literal				
(2/2)									
(3)	NameRes				Check that the value of a character literal is not used to determine its type.		B-Test. Try a call of two overloaded procedures taking parameters of different character types, only one of which has the appropriate literal. This is marked as untested in ACATS 2.x.		
					Check that an overloaded call can be resolved when an actual parameter is a character literal and only one of the subprograms has a character type parameter.	C-Test. This is marked as untested in ACATS 2.x.			
					Check that a character literal can be used as the actual for an appropriate formal function.	C-Test. This is marked as untested in ACATS 2.x.			
				B46002A (type conversion)	Check that a character literal is illegal in a context that does not identify a single type.		B-Test. Try other contexts that require a single type (if there are any that allow characters).		
(4)	NameRes				C87B27A	Check that an overloaded call can be resolved when an actual parameter is a string literal and only one of the subprograms has a string type parameter.			
				B46002A (type conversion)	Check that a string literal is illegal in a context that does not identify a single type.		B-Test. Try other contexts that require a single type (if there are any that allow strings).		

4.3	(5)	Legality	Widely Used	Any character literal.			Check that a character literal is illegal if it is not a value of the expected type.	B-Test. This is marked as untested in ACATS 2.x.
	(6)	Legality	Negative					
			Widely Used	Any string literal.				
			Negative				Check that a string literal is illegal if any character is not a value of the component type of the expected type.	B-Test. This is marked as untested in ACATS 2.x. Also see 4.3.3(19).
	(7/2)	Deleted						
	(8/2)	Definitions	Widely Used	Types of literals.				
			Negative			Check that an expression will not resolve if the expected type of an integer literal is not an integer type.	B-Test. Try many other kinds of types: float, fixed, enumeration, access, record, array, task, protected, etc.	
							Check that an expression will not resolve if the expected type of a real literal is not an float or fixed type.	B-Test. Try many other kinds of types: integer, enumeration, access, record, array, task, protected, etc.
							Check that an expression will not resolve if the expected type of null is not an access type.	B-Test. Try many other kinds of types: integer, float, fixed, enumeration, access, record, array, task, protected, etc.
	(9)	Dynamic	Widely Used	Nothing will work of the values of literals are wrong.				
	(10)	Dynamic			C42007E		Check that the bounds of non-null string literals are determined properly.	Also see 4.3.3(26).
							Check that if the upper bound of a non-null string literal is outside of the appropriate index subtype, Constraint_Error is raised.	C-Test.
							Check that the bounds of null string literals are determined properly when the upper bound is in the index base type.	C-Test.
(11)	1	Dynamic		C42006A		Check that if any character of a string literal does not belong to the dynamic component subtype of the expected type, Constraint_Error is raised.		
						Check that if any character of a static string literal does not belong to the static component subtype of the expected type, the literal is illegal.	B-Test. This happens because of 4.9(34).	
	2	Dynamic		C420001		Check that non-static null string literals whose upper bound is not in the index base type raise Constraint_Error.		
				B420001		Check that static null string literals whose upper bound is not in the index subtype are illegal.	This happens because of 4.9(34).	
(12)		NonNormative						
(13)		NonNormative						
(14)		NonNormative						
4.3	(1)	Definitions	Subpart	Aggregate, test the individual types.				
	(2)	Syntax						

									B43005A (array dimensions), B43005B (number of components), B43005F (mixed notation), B43105C (type of expressions), B432221A (completeness of array), B43221B (length of array), B43223A (others choice)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						</
--	--	--	--	--	--	--	--	--	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

			Negative	B43002K		Check that positional components can precede any named components in an extension aggregate. 4 Check that named components cannot precede any positional components in a record aggregate.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
				B431002	All	Check that named components cannot precede any positional components in an extension aggregate.	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
3	Legality	Subpart	Other tests will test the others choice.				
			Negative	B43002F, B43002H		Check that an others component association cannot appear anywhere in a record aggregate other than last.	Note: This also prevents multiple others association, since one of them is not last.
				B431002	All	Check that an others component association cannot appear anywhere in an extension aggregate other than last.	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
(7)	NameRes		This is not really a syntax rule, but rather a resolution one, because of the ambiguity in the syntax.	C87B29A		Check that an expression surrounded by parens is interpreted as a parenthesized expression, not a record aggregate.	
(8/2)	NameRes	Widely Used	This simply determines the type of the aggregate for reference to other rules; real resolution issues are tested for 4.3(3/2).				
				C431A01	All	Check that a record aggregate can have a limited type.	We only test this objective because it is a change from Ada 95.
(9)	1	Definitions	Subpart	Needed - tested by other aggregate tests.			
	2	NameRes	Subpart	Tested by any named notation			
			Negative	B43101A		Check that the selector names in a record aggregate can only name components and discriminants of the record type, and cannot name components of other variants.	
				B431004	All	Check that the selector names in an extension aggregate can only name components and discriminants of the record extension, and cannot name components of other variants or of the type of the ancestor part..	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
(10)	NameRes	Portion	Lead-in for following bullets				
(11)	NameRes			C87B30A		Check that overloaded expressions can be resolved in positional associations of a record aggregate because the type of the associated component is known.	
						Check that overloaded expressions can be resolved in positional associations of an extension aggregate because the type of the associated component is known. 3	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
(12)	NameRes			C43105A, C43105B, C87B30A		Check that overloaded expressions can be resolved in named associations of a record aggregate because the type of the associated component is known.	

(13)	NameRes				3	Check that overloaded expressions can be resolved in named associations of an extension aggregate because the type of the associated component is known.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
					3	Check that overloaded expressions can be resolved in others associations of a record aggregate because the type of the associated component is known.	C-Test.
(14)	Legality					Check that overloaded expressions can be resolved in others associations of an extension aggregate because the type of the associated component is known.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
			C431001			Check that a record aggregate can be for a record extension if it is not descended from any private types.	
		Negative	B431003	All		Check that the type of a record aggregate cannot be for a record extension that is descended from any private type or private extension.	Not tested in ACATS 2.x.
(15/3)	Legality	Negative			7	Check that the type of a record aggregate cannot be a derived type that has an ancestor for which the current view of the parent of the derived type is not a descendant of the full view of the ancestor. descendant of the ancestor.	B-Test. Try examples like those given in AI05-0115-1.
			These objectives use the approved correction of AI05-0016.			Check that null record may appear in place of component associations if no components are needed in a record aggregate.	This isn't the primary objective of these tests, but it is tested.
						Check that null record may appear in place of component associations if no components are needed in an extension aggregate.	This isn't the primary objective of these tests, but it is tested.
		Negative				Check that null record cannot appear in place of component associations if any components are needed in a record aggregate.	
						Check that null record cannot appear in place of component associations if any components are needed in an extension aggregate.	
(16/4)	1	Legality	Subpart	Tested in every legal aggregate. We test others => <> separately because it is new, and because it is different than other associations.			
						B430001, C431001	
						B430001, C432001, C432004	
						B430001	
						B430001	
						C431A01, C431002	All
						C431A01, C431002	All
		Negative			2	Check that a component association (other than others => <>) in a record aggregate is illegal if it does not have an associated component.	B-Test: Test too many positional components.
		Negative				Check that a component association (other than others => <>) in an extension aggregate is illegal if it does not have an associated component.	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
		Negative				Check that a record aggregate is illegal if it has needed components that are not associated with any component associations.	

2	Legality	Negative		B431004	All	Check that an extension aggregate is illegal if it has needed components that are not associated with any component associations.	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
		Negative		B43101A (two named choices, one positional and one named)		Check that a record aggregate is illegal if it has a needed component that is associated with more than one component association.	
		Negative		B431004	All	Check that an extension aggregate is illegal if it has a needed component that is associated with more than one component association.	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
		Subpart	Test should be checked when the number of expression evaluations is tested.				
			We test others => <> and A B => <> separately because they are new, and because they are different than other associations.	C431A01	All	Check that a component association in a record aggregate with a <> may have two or more associated components of different types.	
				C431A01	All	Check that a component association in an extension aggregate with a <> may have two or more associated components of different types.	These rules also apply to extension aggregates, so we test them for those as well.
			Allowed by AI05-0199-1			7 Check that a component association in a record aggregate may have two or more associated components with anonymous access types that statically match.	C-Test.
			Allowed by AI05-0199-1			6 Check that a component association in an extension aggregate may have two or more associated components with anonymous access types that statically match.	C-Test.
		Negative		B43101A		Check that a component association in a record aggregate with an expression cannot have two or more associated components of different types.	
				B431004	All	Check that a component association in an extension aggregate with an expression cannot have two or more associated components of different types.	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
(17/5)	Legality	Subpart	From approved AI12-0046-1. Any aggregate for a variant record will test.	B431005	All	Check that Legality Rules are enforced for all associated components, even when the results vary.	
		Negative	Modified by AI05-0220-1; the previous rule could be circular. Modified again by AI12-0086-1.	B431006	All	Check that if a variant part is not nested in an unselected variant, the value of the governing discriminant of a variant in a record aggregate cannot be non-static.	Note: This objective is wrong, strictly speaking, after the post-Corrigendum document is issued, but the test is OK even when AI12-0086-1 is in effect.
		Negative				3 Check that if a variant part is not nested in an unselected variant, the value of the governing discriminant of a variant in an extension aggregate cannot be non-static unless the subtype is static and the subtype covers only one variant part.	B-Test. Low priority because mixing variants and extensions is rare.

(17.1/2)	Legality	Double Negative	C43103A, C43102B	Check that if a discriminant does not govern a variant part, or the variant part is nested within a variant that is not selected, the value in a record aggregate can be non-static. Probably should correct the objective to be similar to the next one, but that's not a priority.	C-Test: check that the last aggregate in the question of AI05-0220-1 is legal (already tried in B431006). The other case is adequately tested by the existing tests.
				3	
		Double Negative		Check that if a discriminant does not govern a variant part, or the variant part is nested within a variant that is not selected, the value in an extension aggregate can be non-static.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
				3	
		Subpart	Additional objectives from AI12-0086-1.	Check that if a variant part is not nested in an unselected variant, the value of the governing discriminant of a variant in a record aggregate cannot have a non-static subtype.	B-Test. Part of next (post-corrigendum) document; more important when that is issued.
				1	
				Check that if a variant part is not nested in an unselected variant, the value of the governing discriminant of a variant in an extension aggregate cannot have a non-static subtype.	B-Test. Part of next (post-corrigendum) document; more important when that is issued.
				1	
				Check that if a variant part is not nested in an unselected variant, the value of the governing discriminant of a variant in a record aggregate cannot have values that cover more than one variant.	B-Test. Part of next (post-corrigendum) document; more important when that is issued.
				1	
				Check that if a variant part is not nested in an unselected variant, the value of the governing discriminant of a variant in an extension aggregate cannot have values that cover more than one variant.	B-Test. Part of next (post-corrigendum) document; more important when that is issued.
				1	
		Negative		Check that the association in a record aggregate for a discriminant with a default can be given by <>.	C-Test.
				6	
(18)	Dynamic	Not Testable	Discriminant associations with expressions are tested by many legal aggregates.	Check that the association in an extension aggregate for a discriminant with a default can be given by <>.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
				6	
		Widely used	Defines basic execution.	Check the association in a record aggregate for a discriminant without a default cannot be given by <>.	B-Test.
				7	
(19/5)	1	Dynamic	Basic execution is tested by any record aggregate.	Check the association in an extension aggregate for a discriminant without a default cannot be given by <>.	B-Test. These rules also apply to extension aggregates, so we test them for those as well.
				7	
			C43004A, C43004C	Check that each expression of a record aggregate is converted to the appropriate subtype and appropriate checks are made.	C-Test. Try array conversions (length checks).
				2	
				Check that each expression of an extension aggregate is converted to the appropriate subtype and appropriate checks are made.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
				3	
				Check that per-object constraints are elaborated in a record aggregate, and that happens before the associated expression is evaluated and after the value of the discriminant is evaluated.	C-Test. Be sure to check cases where the per-object constraint raises an exception. Marked as not tested in ACATS 2.x.
				8	

								Check that per-object constraints are elaborated in an extension aggregate, and that happens before the associated expression is evaluated and after the value of the discriminant is evaluated.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
	3			Newly added by AI12-0061-1, after Corrigendum 1.				If the value of a discriminant that governs a variant part of a record aggregate is given by a nonstatic expression with a static nominal subtype and has a static predicate, and the static predicate is disabled, then Constraint_Error is raised if the value of the discriminant does not satisfy the predicate.	C-Test. Test only after the next document (Amendment or Revision) is issued; raise the priority at that time.
				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.				If the value of a discriminant that governs a variant part of an extension aggregate is given by a nonstatic expression with a static nominal subtype and has a static predicate, and the static predicate is disabled, then Constraint_Error is raised if the value of the discriminant does not satisfy the predicate.	C-Test. Test only after the next document (Amendment or Revision) is issued; raise the priority at that time.
(19.1/2)	1	Dynamic	Widely used	Any record aggregate.					
	2	Dynamic			C431A02	All		Check that for each association with a <> in a record aggregate, if the associated component has a default expression, that expression is used and not the default initialization of the type of the component.	
					C431A02	All		Check that for each association with a <> in an extension aggregate, if the associated component has a default expression, that expression is used and not the default initialization of the type of the component.	These rules also apply to extension aggregates, so we test them for those as well.
					C431A02	All		Check that for each association with a <> in a record aggregate, if the associated component does not have a default expression, the component is initialized by default.	
					C431A02	All		Check that for each association with a <> in an extension aggregate, if the associated component does not have a default expression, the component is initialized by default.	These rules also apply to extension aggregates, so we test them for those as well.
					C431A01, C431A03	All		Check that if a <> in a record aggregate has multiple associated components, each one is appropriately initialized (either from the default expression or the initialized by default). In particular, check that if these components have the same type and default expression, the expression is evaluated for each one.	
					C431A01, C431A03	All		Check that if a <> in an extension aggregate has multiple associated components, each one is appropriately initialized (either from the default expression or the initialized by default). In particular, check that if these components have the same type and default expression, the expression is evaluated for each one.	These rules also apply to extension aggregates, so we test them for those as well.
(20)		Dynamic			C43107A			Check that each expression in a record aggregate is evaluated once for each associated component when there are multiple associated components.	
								Check that each expression in an extension aggregate is evaluated once for each associated component when there are multiple associated components.	C-Test. These rules also apply to extension aggregates, so we test them for those as well.
(21)		NonNormative		A note.					

	(22)	NonNormative		Start of examples...						
	(23)	NonNormative								
	(24)	NonNormative								
	(25)	NonNormative								
	(26)	NonNormative								
	(27/2)	NonNormative								
	(28)	NonNormative								
	(29)	NonNormative								
	(29.1/2)	NonNormative								
	(29/2/2)	NonNormative								
	(30)	NonNormative								
	(31)	NonNormative		...end of examples.						
<hr/>										
4.3.2	(1)	Redundant								
	(2)	Syntax								
	(3)	Syntax								
	(4/2)	1	Legality	Resolution tested under 4.3(3/2).						
			Negative	This rule is not a Name Resolution rule; that's in 4.3(3/2).	B432001	All	Check that an extension aggregate cannot be of a tagged record type or of a private extension.	Not tested in ACATS 2.x.		
		2	NameRes	Subpart	Any ancestor part expression will test.					
					C432005	All	Check that the ancestor expression type in an extension aggregate can be limited.			
			Negative		B432001	All	Check that the type of an extension aggregate is not used to resolve an ancestor part expression.	Not tested in ACATS 2.x.		
	(5/3)	1	Legality	Subpart	Any ancestor part subtype will test.					
					C432001 (private and private extension), C432004 (abstract)		Check that the subtype of an ancestor part of an extension aggregate can be a private type, a private extension, or an abstract type.	These are the odd cases.		
							7 Check that the ancestor subtype mark in an extension aggregate can be an interface type.	C-Test. This is a new capability in Ada 2005.		
					C432005	All	Check that the ancestor subtype mark in an extension aggregate can be limited.			
			Negative		B432001	All	Check that the subtype of an ancestor part of an extension aggregate cannot be a classwide subtype, nor an untagged type.			
		2	Legality	Subpart	Any ancestor part expression will test.					
			Negative		B432001	All	Check that the expression of an ancestor part cannot be dynamically tagged.	New rule in the Amendment.		
		3	Legality		This rule was substantially modified by AI05-0115-1.	C432001, C432004	Check that the type of an extension aggregate is a descendant of from the type of the ancestor, through one or more record extensions.	The only change was to replace "derived from" to "a descendant of". That changes the wording of the objective, but only makes a difference in obscure visibility cases (we have a separate objective for them).		
			Negative		B432001	All	Check that the type of an extension aggregate cannot be unrelated from the type of the ancestor part.	Same as the first objective for this line.		

					B432001	All	Check that the type of the ancestor part cannot be the same as or descended from the type of the extension aggregate.	Same as the first objective for this line.
					B432001	All	Check that the type of an extension aggregate cannot be descended from the type of the ancestor through any private extensions.	Same as the first objective for this line.
							Check that the type of an extension aggregate cannot be a type such that the current view of the parent of the aggregate type is not a descendant of the full view of the ancestor type. descendant of the ancestor.	B-Test. Try examples like those given in AI05-0115-1.
							Check that the subtype_mark of an extension aggregate cannot denote a view of a type that has unknown discriminants.	B-Test. Try both private types with unknown discriminants, and generic formal types with unknown discriminants.
4	Legality				Added by AI05-0115-1.			
(5.1/3)	Legality	Portion			Added by AI05-0067-1 and AI05-0244-1. This is the lead-in.			
(5.2/3)	Legality				Added by AI05-0067-1 and AI05-0244-1.		Check that the ancestor_part of a limited extension aggregate can be a function call (even if the function has an unconstrained nominal subtype) if the extension part has no needed components.	C-Test. Try parenthesizing and qualifying the function call.
							Check that the ancestor_part of a limited extension aggregate can be an object that has an unconstrained nominal subtype, even if the extension part has components.	C-Test. Try parameters, class-wide objects, and array objects. Try parenthesizing and qualifying the objects.
		Negative					Check that the ancestor_part of a limited extension aggregate cannot be a function call of a function with an unconstrained result subtype unless the aggregate has no needed extension components.	B-Test.
(5.3/3)	Legality				Added by AI05-0067-1 and AI05-0244-1.	We suggest testing these cases as part of the previous (positive) objectives; they have little value on their own.		
		Negative					Check that the ancestor_part of a limited extension aggregate cannot be a parenthesized or qualified function call of a function with an unconstrained result subtype unless the aggregate has no needed extension components.	B-Test. (Combine with the previous?)
(5.4/3)	Legality				Added by AI05-0067-1 and AI05-0244-1.		Check that the ancestor_part of a limited extension aggregate can be a conditional expression that has a dependent expression that has an unconstrained nominal subtype if the extension part has no needed components.	C-Test.
							Check that the ancestor_part of a limited extension aggregate can be a conditional expression where one or more dependent expressions are objects that have an unconstrained nominal subtype, even if the extension part has components.	C-Test.
		Negative					Check that the ancestor_part of a limited extension aggregate cannot be a conditional expression that has a dependent expression that is a function call of a function with an unconstrained result subtype unless the aggregate has no needed extension components.	C-Test. Try parenthesizing and qualifying the conditional expression and the dependent expression.

(6)	StaticSem	Subpart	Any extension aggregate will test normal cases.				Check that inherited discriminants are needed by an extension aggregate if the ancestor subtype mark denotes an unconstrained subtype. 6	C-Test. (?)
		Negative					Check that values cannot be given for components included in the ancestor expression or subtype of an extension aggregate. 6	B-Test.
		Negative		C432002 (not given).			Check that inherited discriminants are not needed by an extension aggregate if the ancestor part is an expression or constrained subtype. 4	B-Test (try to give them).
			Extra and missing needed components are tested in 4.3.1(16/2).					
(7)	Dynamic			C432003, C432004			Check that the components associated with an ancestor part subtype mark in an extension aggregate are initialized by default. Check that the components associated with an ancestor part expression in an extension are initialized by the expression.	
			Order of discriminant evaluation tested in 4.3.1(19).	C432001				
(8/3)	Dynamic		This check was broadened by AI05-0282-1.	C432002, C432003			If the type of the ancestor part has discriminants that are not inherited by the type of an extension aggregate, then check that the values of the discriminants are checked and constraint_error raised if needed.	This is the Ada 95 objective. C-Test. Test cases not covered in the existing tests: discriminants that come from a constrained ancestor. See the example in the AI05-0282-1. Be sure to test the inconsistency mentioned in 4.3.2(13.d/3).
							If the type of the ancestor part has discriminants and the ancestor_part is not an unconstrained subtype name, then check that the values of the discriminants are checks and Constraint_Error is raised if needed. 5	
(9)	NonNormative		A note.					
(10)	NonNormative		Another note.					
(11)	NonNormative		Start of examples...					
(12)	NonNormative							
(13)	NonNormative		...end of examples.					
4.3.3	(1)	1	Redundant					
		2-3	Definitions	Positional and named array aggregates.				
			Syntax					
			Syntax					
			Syntax					
			Syntax	Changed by AI12-0061-1.				
				Since the syntax of array and record aggregates has to be shared, this is likely to be a check outside of the syntax.	B43201A		Check that an array aggregate cannot have mixed positional and named notation (excepting others). Check that <> is not allowed in positional array aggregates other than in an others choice.	
			Negative		B433001	All		

(5.1/5)		Syntax		Added by AI12-0061-1, post Corrigendum 1.				
(6)		Definitions		How n-dimensional array aggregates work.				
(6.1/5)		Definitions		Added by AI12-0061-1, post Corrigendum 1. "Index Parameter".				
(7/2)	1	NameRes	Widely Used	This simply determines the type of the aggregate for reference to other rules; real resolution issues are tested for 4.3(3/2).				
	2	NameRes			C87B31A (one-dim)		Check that array component expressions in an array aggregate can be resolved because they must have the component type of the array type of the aggregate.	C-Test: check multi-dimensional arrays.
			Negative		B43201D		Check that array component expressions in an array aggregate must have the component type of the array type of the aggregate.	
					C433A01 (one-dim), C433A02 (two-dim)	All	Check that the array component expressions in an array aggregate may have a limited type.	
(8)					C87B31A (one-dim)		Check that the choices in a named array aggregate can be resolved because they must have the corresponding index type of the array type of the aggregate.	C-Test: check multi-dimensional arrays.
			Negative		B43201D		Check that the choices in a named array aggregate must have the corresponding index type of the array type of the aggregate.	
(9)		Legality	Subpart	All M-dimensional array aggregates will test.				
			Negative					
(10)		Legality	Portion	Lead-in for following bullets				
			Negative		B43202C (operators, membership)			Are there any other such contexts?? I can't think of any, if there are any I've missed they should be tested.
(11/4)		Legality		The positive objectives really belong to 4.3.3(25), but here we can spread them out more clearly.	C43204A, C433001		Check that the constraint of the constrained array subtype of an explicit actual parameter of a subprogram or entry call is used to determine the bounds of an array aggregate with an others choice.	C-Test. Try entry calls and others => <>.
					C43204C		Check that the constraint of the constrained array subtype of an explicit actual parameter of an instantiation is used to determine the bounds of an array aggregate with an others choice.	
					C433005	All	Check that the constraint of the constrained array subtype of a function return is used to determine the bounds of an array aggregate with an others choice in the expression of a return statement.	
				Added by AI12-0157-1 (although always assumed).	C433006	All	Check that the constraint of the constrained array subtype of a function return is used to determine the bounds of an array aggregate with an others choice in the return expression of an expression function.	

			C43204E, C433001		Check that the constraint of the constrained array subtype of an object declaration (including constants) is used to determine the bounds of an array aggregate with an others choice in the initializing expression of the object.	
			C43204E		Check that the constraint of the constrained array subtype of a component declaration is used to determine the bounds of an array aggregate with an others choice in the default expression of the component.	
			C43204F (subprogram), C43204G (entry), C43204H (generic unit)		Check that the constraint of the constrained array subtype of a formal parameter is used to determine the bounds of an array aggregate with an others choice in the default expression of the parameter.	
	Negative		B43202A		Check that an others choice is not allowed in an array aggregate that is an explicit actual parameter of a subprogram or entry call if its subtype is unconstrained.	
	Negative		B43202A		Check that an others choice is not allowed in an array aggregate that is an explicit actual parameter of an instantiation if its subtype is unconstrained.	
	Negative		B43202A (simple return, others with value), B433003	All	Check that an others choice is not allowed in an array aggregate in the expression of a return statement if the subtype of the function return is unconstrained.	
	Negative	Added by AI12-0157-1 (although always assumed).	B433003	All	Check that an others choice is not allowed in an array aggregate in the return expression of an expression function if the subtype of the function return is unconstrained.	
	Negative		B43202A		Check that an others choice is not allowed in an array aggregate in the initializing expression of an object declaration if the subtype of the object is unconstrained.	B-Test. Try a variable declaration and others => <>.
	Negative				Check that an others choice is not allowed in an array aggregate in the default expression of a component declaration if the subtype of the component is unconstrained.	B-Test. Try others => <>.
	Negative		B43202A		Check that an others choice is not allowed in an array aggregate in the default expression of a formal parameter if the subtype of the parameter is unconstrained.	
(12)	Legality	The positive objectives really belong to 4.3.3(25), but here we can spread them out more clearly.	C43204I, C433001		Check that the constraint of the target array object of an assignment statement is used to determine the bounds of an array aggregate with an others choice in the source expression.	
	Negative	This is impossible, as an array object must be constrained.				
(13)	Legality	The positive objectives really belong to 4.3.3(25), but here we can spread them out more clearly.	C433001		Check that the constraint of the array subtype of a qualified expression is used to determine the bounds of an array aggregate with an others choice in the qualified expression.	
	Negative		B43202A		Check that an others choice is not allowed in an array aggregate in the expression of a qualified expression if the subtype_mark in the qualified expression is unconstrained.	

(14)	Legality	Negative	This is impossible, as components have to be definite.			<p>Check that the constraint of the component array subtype of an aggregate component is used to determine the bounds of an array aggregate with an others choice which is used as a component expression in a larger aggregate.</p> <p>1</p>	C-Test. This is not usefully testable, as it is not possible to find the bounds used for the component separate from the array object it is nested in.
(15/3)	Legality		A punctuation change by AI05-0147-1.	C433006 (expression functions)	Part	<p>Check that the constraint of the applicable index constraint of a parenthesized expression is used to determine the bounds of a parenthesized array aggregate with an others choice.</p> <p>3</p>	C-Test. Check parenthesized expressions in all of the other contexts noted under 4.3.3(11-14, 15.1).
		Negative		B430003 (return statements, expression functions)	Part	<p>Check that an others choice in a parenthesized array aggregate is not allowed in contexts where the appropriate subtype (as described by 4.3.3(11-15)) is unconstrained.</p> <p>3</p>	B-Test. Check parenthesized expressions in all of the other contexts.
(15.1/3)	Legality		Added by AI05-0147-1 (Ada 2012)			<p>Check that the constraint of the applicable index constraint of a conditional_expression is used to determine the bounds of a dependent array aggregate with an others choice.</p> <p>6</p>	C-Test. Check conditional_expressions in all of the other contexts noted under 4.3.3(11-14). Make sure to try both if and case expressions.
(16)	1	Definitions	Constraint that applies.			<p>Check that an others choice in an array aggregate is not allowed in contexts where the appropriate subtype (as described by 4.3.3(11-15.1)) is unconstrained.</p> <p>5</p>	B-Test. Check conditional expressions in all of the other contexts noted under 4.3.3(11-14). Make sure to try both if and case expressions.
	2	Legality	Negative			<p>Check that an others choice is not allowed in an array aggregate that is an explicit actual parameter of a subprogram call to a generic formal subprogram even if its subtype is constrained.</p> <p>6</p>	B-Test. This was a fix for a contract model violation in Ada 83, and it should be tested.
						<p>Check that an others choice is not allowed in an array aggregate in the default expression of a formal parameter of a generic formal subprogram even if the subtype of the parameter is constrained.</p> <p>6</p>	B-Test. This was a fix for a contract model violation in Ada 83, and it should be tested.
(17/5)	Legality	Negative	AI05-0153-1 modifies the syntax terms used here, but the effect is unchanged. AI12-0061-1 adds a parenthetical remark.	C43207D (range), C43208A (range), C43208B (range), C43224A (range attribute)		<p>Check that a single nonstatic choice is allowed in an array aggregate.</p> <p>2</p>	C-Test. Check that the single choice can be an expression, and after the next document (Amendment or Revision) is issued, also check that the choice can be an iterated_component_association.
						<p>Check if an array aggregate contains more than one choice or component association, it is illegal for any (other than a others choice) of them to be nonstatic.</p> <p>5</p>	B-Test. This is marked as untested in ACATS 2.x.
						<p>Check if an array aggregate contains more than component association, including one iterated_component_association, it is illegal for any (other than a others choice) of them to be nonstatic.</p> <p>1</p>	B-Test. Test after the next document (Amendment or Revision) is issued.
(18/3)	Legality	Widely Used	Any named notation array aggregate will test this.				
		Negative		B43201C		<p>Check that a named array aggregate cannot contain two choices that cover the same value.</p>	
			New cases added by AI05-0262-1.	B43201C, B433002	All	<p>Check that a named array aggregate is illegal if it does not cover a contiguous set of index values.</p>	

(19)	Legality			C43209A		Check that a bottom level subaggregate of an array aggregate can be a string literal.	
						Check that a string literal cannot be used as the bottom level subaggregate of an array aggregate if the component type is not a character type.	B-Test. Hard to imagine an implementation getting this wrong.
(20)	StaticSem					Check that a string literal used as the bottom level subaggregate of an array aggregate is illegal if any character is not a value of the component type of the aggregate.	B-Test. This is marked as untested in ACATS 2.x. (So is 4.2(7)).
				B43209B		Check that a string literal used as a bottom level subaggregate of an array aggregate cannot be enclosed in parentheses.	
(20.1/5)	StaticSem		Added by AI12-0061-1, post Corrigendum 1.			Check that the nominal subtype for an index parameter in an array aggregate is correct.	B-Test. Test as the selecting expression of a case expression (coverage should be correct). Test after the next document (Amendment or Revision) is issued.
(21)	Dynamic	Portion	Lead-in for following bullets				
(22)	Dynamic	Not Testable	Arbitrary order is untestable. The conversion could fail, but any case that did would also fail 4.3.3(28), so those tests dominate.				
(23)	Dynamic	Widely Used	Arbitrary order is untestable. Normal evaluation is tested by every array aggregate.				
				C43004A		Check that Constraint_Error is raised if a component expression fails the conversion to the component subtype of an array aggregate.	C-Test. Try composite types (discriminant checks, array length checks).
				C43207D (2-dim range), C43208A (1-dim range), C43208B (2-dim range), C43210A (1-dim & 2-dim, named & others)		Check that a component expression in an array aggregate with multiple associated components is evaluated once for each associated component.	
(23.1/4) 1	Dynamic	Widely Used	Any array aggregate				
				C433A01 (task, PO, lim-rec for 1-dim); C433A02 (task, PO, lim-rec for 2-dim); C433A04 (non-lim cases); C433003 (scalar types with Default_Value)	All	Check that for each association with a <> in an array aggregate, the component is initialized by default.	
2				C433A03	All	Check that for a <> in an array aggregate with multiple associated components, each associated component is default initialized individually.	
			Part added by AI12-0084-1.	C433004	All	Check that for each association with a <> in an array aggregate whose type has a Default_Component_Value, the component is initialized to the Default_Component_Value.	

(23.2/5)	Dynamic		Added by AI12-0061-1, post Corrigendum 1.			
(24)	Dynamic	Portion	Lead-in for following bullets.			
(25)	Dynamic		Tested under 4.3.3(11-15).			
(26)	Dynamic		C43205A (subprogram, unconstrained), C43205G (subprogram, constrained), C43214B (subprogram, constrained, string literal)	Check that the index parameter of an array aggregate takes on the value of each index of the covered array components. 1	C-Test. Test after the next document (Amendment or Revision) is issued; high priority then. Be careful that the order doesn't matter. Try nonstatic cases and non-contiguous cases.	
			C43205B (unconstrained), C43205H (constrained), C43214C (constrained, string literal)	Check that the constraint (or lack of one) the array subtype of an explicit actual parameter of a subprogram or entry call is used to determine the lower bound of a positional array aggregate or string literal. 3	C-Test. Try entry calls and string literals in unconstrained contexts.	
			C43205C (unconstrained), C43205I (constrained), C43214D (constrained, string literal)	Check that the constraint (or lack of one) of the array subtype of an explicit actual parameter of an instantiation is used to determine the lower bound of a positional array aggregate or string literal. 2	C-Test. Try string literals in unconstrained contexts.	
			C43205D (constant, unconstrained), C43205J (objects, constrained), C43214E (objects, constrained, string literal)	Check that the constraint (or lack of one) of the array subtype of a function return is used to determine the lower bound of a positional array aggregate the expression of a return statement or string literal. 2	C-Test. Try string literals in unconstrained contexts.	
				Check that the constraint (or lack of one) of the array subtype of an object declaration (including constants) is used to determine the lower bound of a positional array aggregate or string literal in the initializing expression of the object. 4	C-Test. Try an unconstrained variable, and string literals in any unconstrained contexts.	
				Check that the constraint (or lack of one) of the array subtype of a component declaration is used to determine the lower bound of a positional array aggregate or string literal in the default expression of the component. 3	C-Test.	
			C43205J (subprogram, generic; constrained)	Check that the constraint (or lack of one) of the array subtype of a formal parameter is used to determine the lower bound of a positional array aggregate or string literal in the default expression of the parameter. 3	C-Test. Try entry declarations, unconstrained contexts, string literals.	
				Check that the constraint of the target array object of an assignment statement is used to determine the lower bound of a positional array aggregate or string literal in the source expression. 1	C-Test. Not usefully testable because the bounds slide on the assignment.	
				Check that the constraint (or lack of one) of the array subtype of a qualified expression is used to determine the lower bound of a positional array aggregate or string literal in the qualified expression. 3	C-Test.	
			C43205K, C43214F (string literal), C460010	Check that the constraint of the component array subtype of an aggregate component is used to determine the lower bounds of a positional array aggregate or string literal which is used as a component expression in a larger aggregate. Check that the constraint of the applicable index constraint of a parenthesized expression is used to determine the lower bound of a parenthesized positional array aggregate or string literal. 3	C-Test.	

						Check that the lower bound of a positional array aggreate or string literal in a membership is always that of the index subtype, even if the subtype is constrained.	C-Test.
					C42007E (string literal), C43205E (string literal)	Check that the lower bound of a positional array aggreate or string literal used as the operand of a predefined operator is always that of the index subtype.	C-Test. Try aggregates.
(27)	Dynamic				C43206A (null).	Check that the bounds of a named array aggregate without others are determined by the choices.	C-Test. Try non-null aggregates.
(28)	Dynamic				C43215A, C43215B C43207B (ranges), C43211A (ranges), C43214A (ranges)	Check that Constraint_Error is raised if the upper bound of a positional array aggregate without an others choice would be outside of the index subtype.	
(29/3)	Dynamic				C433001	Check that Constraint_Error is raised if any non-null choice of a named array aggregate is outside of the index subtype.	C-Test. Try single choices.
			Approved AI05-0037 mandates this.	C433002	All	Check that Constraint_Error is raised if any choice of an aggregate with an others clause specifies a component outside of the bounds of the aggregate.	
(30)	Dynamic				C43212A, C43212C	Check that Constraint_Error is raised if any <> choice of an aggregate with an others clause specifies a component outside of the bounds of the aggregate.	
(31)	Dynamic	Portion	Defines the exception to raise for the previous rules.			Check that all subaggregates of a multidimensional array aggregate that correspond to the same index have the same bounds.	
(32/2)	NonNormative		A note.				
(33/5)	NonNormative		Another note, added by AI12-0061-1. All of the following numbers were changed.				
(34)	NonNormative		Start of examples...				
(35)	NonNormative						
(36)	NonNormative						
(37)	NonNormative						
(38)	NonNormative						
(39)	NonNormative						
(40)	NonNormative						
(41)	NonNormative						
(42)	NonNormative						
(43)	NonNormative						
(44)	NonNormative						
(45/5)	NonNormative		A new example added by AI12-0061-1.				
(46/2)	NonNormative						
(47/5)	NonNormative		End of examples. Corrected by AI12-0178-1.				
4.4	(1/3)	General	Modified by AI05-0147-1.				
	(2)	Syntax					

(2.1/3)	Syntax		Added by AI05-0158-1. Note that the important effects of these changes are tested in 3.8 (for choices) and 4.5.2 (for memberships)			
(2.2/3)	Syntax		Added by AI05-0158-1.			
(3/4)	Syntax		Added by AI05-0158-1 and AI12-0022-1; corrected by AI12-0039-1.			
(3.1/3)	Syntax		Added by AI05-0158-1.			
(3.2/4)	Syntax		Added by AI05-0158-1; corrected by AI12-0039-1.			
(4)	Syntax					
(5)	Syntax					
(6)	Syntax					
(7/3)	Syntax		These syntax changes are considered an Amendment (AI05-0003-1, AI05-0147-1, AI05-0176-1).			
			We test precedence separately, because it is possible for the syntax to be flattened.	C44003D (float), C44003F (enum), C44003G (Boolean)	1 Check that the precedence of operators is correct.	C-Test. Try integer, modular, and ordinary and decimal fixed point types. But not likely to be wrong.
(8)	NameRes				Check that a primary can be resolved because it must denote an object or value. 3	C-Test. Try a function overloaded with a procedure; use in an expression must resolve to the function call (even without parameters).
				B44001B (procedure), B44002B (tasks, entries), B44002C (exception)	Check that names that do not denote an object or value are not permitted as primaries. 1	B-Test. Try type and subtype names, package names, single protected object names, block and loop names. But not likely to be wrong.
(9)	Definitions	Negative				
(10)	Dynamic	Widely Used	Any object name used in an expression tests this.			
(11)	Impl-Def	Not Testable	Either something happens, or it doesn't. Can't test that.			
(12)	NonNormative		Start of examples...			
(13)	NonNormative					
(14)	NonNormative					
(15/2)	NonNormative		...end of examples.			

Paragraphs:		Objectives with tests:	Objectives to test:	Total objectives:	Objectives with submitted tests:
10	222	169	150	279	1
Must be tested		Objectives with Priority 10	0		
		Objectives with Priority 9	0		
Important to test		Objectives with Priority 8	5		
		Objectives with Priority 7	12		
Valuable to test		Objectives with Priority 6	20		
		Objectives with Priority 5	14		
Ought to be tested		Objectives with Priority 4	27		
		Objectives with Priority 3	31		
Worth testing		Objectives with Priority 2	25		
Not worth testing		Objectives with Priority 1	16		
		Total:	150		
		Objectives covered by new tests since ACATS 2.6	83		
		Completely:	75		