

Coverage for ISO/IEC 8652:2007 Ed. 3 in ACATS 3.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.x are shown in **bold**.

Clause	Para.	Lines	Kind	Subkind	Notes	Tests	Objective's New Priority	Objective Text	Objective notes	Submitted tests (will need work).
4.1.3	(1)		Redundant							
	(2)		Syntax							
	(3)		Syntax							
	(4)		Definition	Widely Used	Expanded Name					
	(5)		NameRes					<p>Check that if the prefix of a selected component denotes an enclosing construct, it is not interpreted as a component reference.</p> <p>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.</p> <p>Check that if the prefix of a selected component denotes an enclosing construct, it is not interpreted as a prefix view.</p>	<p>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.</p> <p>B-Test (?)</p> <p>B-Test (?) or a C-Test like the one described above.</p>	
	(6)		NameRes	Portion	Lead-in for next rule.			<p>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.</p>		
	(7)		NameRes			C41301A		<p>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.</p> <p>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.</p>	<p>C-Test. Try cases like those found in C41301A. Simple cases probably exist in many other ACATS tests, thus the low priority.</p>	
	(8)		NameRes	Negative Portion	Lead-in for next rule.	B940005 contains a single example.			B-Test. Try many kinds of prefixes.	
	(9)		NameRes			C41306A, C41306B, C41306C		<p>Check that for the reference L,R, if L represents a task value or object, R can represent a task entry or family.</p>	<p>Many tests must use the simple case of T.E, but I don't have a reference.</p>	
	(9.1/2)		NameRes	Portion	Lean-in for next rule.			<p>Check that for the reference L,R, if L represents a protected value or object, R can represent a protected entry or subprogram.</p>	<p>C-Test. Simple cases are scattered throughout the ACATS; we mainly need to test examples like those in C41306x.</p>	
	(9.2/2)	1	NameRes		These objectives mostly cover the first three lines.	C413001	All	<p>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.</p>		

C413002	All	<p>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.</p>	
C413001	All	<p>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.</p>	
C413002	All	<p>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.</p>	
C413003	All	<p>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.</p>	
C413004	All	<p>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.</p>	
C413003	All	<p>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.</p>	
C413004	All	<p>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.</p>	
C413004	All	<p>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.</p>	<p>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.]</p>

		Negative		B413004	Part	<p>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 parameter of T, the reference is illegal even if the subprogram is primitive for T.</p>	3	B-Test. Try other types, including protected, task, limited record, float, fixed, decimal, modular, enum. But this isn't very important.
		Negative		B413004	Part	<p>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.</p>	2	B-Test. Try other types, including protected, task, limited record, float, fixed, decimal, modular, enum. But this isn't very important.
2		Negative				<p>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.</p>	8	B-Test.
3		Negative				<p>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.</p>	7	B-Test.
4				B431001	All	<p>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.</p>		
5	StaticSem	Negative	Subpart			<p>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.</p>	3	C-Test. B431001 includes this case, which is why the priority is low.
				C413005	All	<p>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.</p>		
6	Definitions	Negative				<p>Check that a call of a prefixed view cannot repeat the first parameter in the parameter list.</p>	8	B-Test.
(10)	NameRes	Portion						
(11)	NameRes	Subpart				<p>Lead-in for the following rules. Tested in the next two rules.</p>		
(12)	NameRes					<p>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.</p>	3	C-Test: commonly used but no obvious test to report here.

			C41320A (enum), C41321A (derived Boolean), C41322A (signed integer), C41323A (float), C41324A (fix), C41325A (array), C41326A (access), C41327A (private), C41328A (inherited subs, derived type)	<p>Check that for the reference L.R, if L represents the name of a package, then R can name any implicitly declared declarations in the visible part of the package.</p> <p>If L represents the name of a package, check that for the reference L.R given in the private part or body of package L or the private part or body of a public child of L or in a private child of L, then R can name any declaration in the package private part of the package.</p> <p>If L represents a renaming of a package P, check that for the reference L.R, R can name any visible declaration in the package P.</p> <p>Check that for the reference L.R, if L represents the name of a package, then R cannot name any declaration of the package not visible at the point of the reference.</p> <p>Check that for the reference L.R, if L represents the name of an enclosing construct, then R cannot name an entity declared other than in that enclosing construct. Check that an expanded name can reference a declaration in a callable construct, type declaration, accept statement, block statement, or loop statement if it is given within that construct.</p> <p>Check that for reference L.R, L can be an operator symbol if R is a declaration in that operator and the reference occurs within the operator.</p> <p>Check that an expanded name is illegal if it tries to reference a declaration inside of a callable construct, accept statement, block statement, or loop statement outside of that construct by naming the construct in its prefix.</p> <p>Check that an expanded name is illegal if it tries to reference a declaration inside of a type declaration outside of the type declaration by naming the type in its prefix.</p> <p>Check that a family index is not allowed in an expanded name for an access statement or entry body. Check that if expanded name occurs within a callable construct, and the prefix of an expanded name denotes more than one enclosing callable construct, the expanded name is illegal.</p>	<p>C-Test. Need modular types and decimal fixed types; maybe types derived from interfaces as well.</p> <p>C-Test. Expanded names don't appear in the relevant Section 10 tests, so we need them here.</p> <p>C-Test. An untested Ada 83 objective.</p> <p>B-Test.</p> <p>B-Test. Try items declared in nested and outer scopes.</p> <p>C-Test (need to test a protected function, protected procedure, and entry body).</p> <p>C-Test. An untested Ada 83 objective.</p> <p>B-Test.</p> <p>B-Test.</p> <p>B-Test.</p>
(13)	1	NameRes	C41307D	<p>Negative</p> <p>Negative</p> <p>Negative</p> <p>Negative</p> <p>Negative</p>	

(13.1/2)	Legality	Subpart	Prefixed view calls are tested in 6.4(10.1/2).					
		Negative		B413002	All		Check that the prefixed view L.R is illegal if the first parameter of R is an access parameter and L is not an aliased view of an object.	
(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.)			Check that the prefix of an attribute can be another attribute.	C-Test. Check that T'Class'something and T'Base'something work. These may be covered by other existing tests scattered throughout the test suite.
	(3)	Syntax						
	(4)	Syntax						
	(5)	Syntax						
				C41401A (Callable, Terminated, First, Last, Length, Range)			Check that the prefix of attributes that do not apply to objects of an access types can be interpreted as an implicit dereference.	C-Test. Test Component_Size, Constrained, Tag, and Valid. Test by determining that a prefix with the value null raises Constraint_Error (and resolves).
(6)	1	NameRes						
				C41402A (Address, Size, First_Bit, Last_Bit, Position)			Check that the prefix of attributes that apply to objects of an access types are not interpreted as an implicit dereference.	C-Test. Test Access, Alignment, Storage_Size, Unchecked_Access. Test by determining that a prefix with the value null does not Constraint_Error.

2

Check that the prefix of attributes that apply to objects but not functions is interpreted as a parameterless function call.

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?)

Check that the prefix of attributes that apply to both objects and functions is never interpreted as a parameterless function call.

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. C-Test. I believe this objective cannot be tests because of the requirement that this expression be static (we need user-defined functions to get interesting overloading).

Negative
These are the only rules for most attributes.
B87B26A (First_Bit, Last_Bit, Position, Callable, Terminated, First, Last, Length, Range, Count)

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.

(7) NameRes

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.

(8) Legality Negative

This is untestable by itself; it will be tested as part of testing each attribute.
B36201A has a single test case for Length.

Check that the expression of a First, Last, Length, or Range attribute must be static.

(9/3) 1 StaticSem Subpart

Added by AI05-0006-1.

Check that a First or Last attribute can be used as the expression of a case statement, and coverage is required for the base subtype of its type.

B-Test.

2

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) 3 Redundant

Added by AI05-0006-1.

(11) Dynamic Subpart

The only effect of this is that a test checking that Small is not defined for floating point types is incorrect.

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

A note.

(13) NonNormative

Another note.

(14/2) NonNormative

Start of examples.

(15) NonNormative

End of examples.

(16) NonNormative

4.2	(1)	1	Redundant						
	(2/2)	2	Definitions Deleted	Widely Used	Literal				
	(3)		NameRes						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 the value of a character literal is not used to determine its type.		
							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.		B-Test. Try an aggregate as the ancestor of an extension aggregate.
						B46002A (type conversion)	Check that a string literal is illegal in a context that does not identify a single type.		
	(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. Try other contexts that require a single type (if there are any that allow strings).
	(6)		Legality	Negative Widely Used	Any string 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.
	(7/2)		Deleted	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).
	(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 a 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, 3 Constraint_Error is raised.	C-Test.
					Check that the bounds of null string literals are determined properly when the upper bound is in the 2 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 2 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.	
(12)		NonNormative		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).
(13)		NonNormative				
(14)		NonNormative				
<hr/>						
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)	
	(3/2)	NameRes			Check that the contents of an aggregate are not used to 2 resolve it.	B-Tests. Additional cases (extension aggregates must be tagged, null record must be record) should be tried here.
					Check that limitedness is not used to resolve 5 expressions containing aggregates	B-Test. Try a subprogram overloaded on limited and nonlimited record types.
					Check that an overloaded call can be resolved when an actual parameter is an aggregate and only one of the 3 subprograms has a record or array parameter.	C-Test. Try record, array, and extension. Contrast to private types completed by record or arrays (which are not considered). Not tested in ACATS 2.x.
					Check that an aggregate is illegal in a context that does 4 not identify a single type.	B-Test. Try an aggregate as the ancestor of an extension aggregate.
					B46002A (type conversion)	
	(4)	Legality		B430001	Check that an aggregate cannot be of a class-wide type.	
	(5)	Dynamic	Widely Used	Any aggregate does this.		
				This says that no order can be		
	2	Dynamic	Not Testable	depended upon.		
	3	Dynamic	Widely Used	Any aggregate will test this.		

	(6)	Dynamic						Check that an aggregate of a tagged type that does not belong to the first subtype of its type raises 4 Constraint_Error.	C-Test. A constrained first subtype with inherited discriminants is necessary, and an ancestor object that is not in that subtype.
4.3.1	(1) (2) (3) (4/2)	Redundant Syntax Syntax Syntax							
			Negative	This grammar may be ambiguous.	B431001	All		Check that a positional component association in a record aggregate cannot have a <> rather than an expression. Check that a positional component association in an extension aggregate cannot have a <> rather than an expression.	Note: These rules also apply to extension aggregates, so we test the rules for them as well as record aggs.
	(5)	Syntax			B431002	All			
	(6)	1 Definitions	Widely Used	Any aggregate will test one or the other.					
		2 Legality			C43106A			Check that positional components can precede any named components in a record aggregate.	
			Negative		B43002K			Check that positional components can precede any 4 named components in an extension aggregate. 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. Check that an others component association cannot appear anywhere in an extension aggregate other than last.	Note: This also prevents multiple others association, since one of them is not 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. This simply determines the type of the aggregate for reference to other rules; real resolution issues are tested for 4.3(3/2).	C87B29A			Check that an expression surrounded by parens is interpreted as a parenthesized expression, not a record aggregate.	
	(8/2)	NameRes	Widely Used						
					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				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.	
			Negative		B43101A			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	B431004	All			
(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.	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.	
								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.
(13)		NameRes						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.
								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.
(14)		Legality			C431001			Check that a record aggregate can be for a record extension if it is not descended from any private types. 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.
			Negative		B431003	All		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.
(15/3)		Legality		These objectives use the approved correction of AI05-0016.	B430001, C431001			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.
					B430001, C432001, C432004				

		Negative		B430001			Check that null record cannot appear in place of component associations if any components are needed in a record aggregate.	
				B430001			Check that null record cannot appear in place of component associations if any components are needed in an extension aggregate.	
(16/2)	1	Legality	Subpart	Tested in every legal aggregate.				
				We test others => <> separately because it is new, and because it is different than other associations.	C431A01	All	Check that a component association of others => <> in a record aggregate may have any number of associated components, including none.	
					C431A01	All	Check that a component association of others => <> in an extension aggregate may have any number of associated components, including none.	These rules also apply to extension aggregates, so we test them for those as well.
		Negative			B43101A (others)		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			B431004	All	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			B43101A		Check that a record aggregate is illegal if it has needed components that are not associated with any component associations.	
		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.
		Negative			B43101A		Check that a component association in a record aggregate with an expression cannot have two or more associated components of different types.	
	2	Legality	Subpart					

(17)	Legality	Subpart	Any aggregate for a variant record will test.	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.
		Negative		B43102A		Check that if the components of a variant part are needed, the value of the governing discriminant in a record aggregate cannot be non-static. Check that if the components of a variant part are needed, the value of the governing discriminant in an extension aggregate cannot be non-static.	B-Test. Low priority because mixing variants and extensions is rare.
		Negative				2 Check that if a discriminant does not govern a needed variant part, the value in a record aggregate can be non-static.	
		Double Negative		C43103A, C43102B		3 Check that if a discriminant does not govern a needed variant part, 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.
		Double Negative				3 Check that the association in a record aggregate for a discriminant with a default can be given by <>.	C-Test.
(17.1/2)	Legality					6 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.
		Subpart	Discriminant associations with expressions are tested by many legal aggregates.				
		Negative				7 Check the association in a record aggregate for a discriminant without a default cannot be given by <>.	B-Test.
		Negative				7 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.
(18)	Dynamic	Not Testable	Defines basic execution. Basic execution is tested by any record aggregate.				
(19)	Dynamic	Widely used				2 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).
				C43004A, C43004C		3 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.
						8 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.
						7 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.
(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.				
4.3.2	(1)	Redundant						
	(2)	Syntax						
	(3)	Syntax						
	(4/2)	1	Legality	Resolution tested under 4.3(3/2). This rule is not a Name Resolution rule; that's in 4.3(3/2). Any ancestor part expression will test.	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	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.
	(5/2)	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.
					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		C432001, C432004		Check that the type of an extension aggregate is derived from the type of the ancestor, through one or more record extensions.	
				Negative	B432001	All	Check that the type of an extension aggregate cannot be unrelated from the type of the ancestor part.	
					B432001	All	Check that the type of the ancestor part cannot be the same as or derived from the type of the extension aggregate.	
					B432001	All	Check that the type of an extension aggregate cannot be derived from the type of the ancestor through any private extensions.	
	(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 6 denotes an unconstrained subtype.	C-Test. (?)

			Negative			Check that values cannot be given for components included in the ancestor expression or subtype of an extension aggregate.	B-Test.
			Negative	Extra and missing needed components are tested in 4.3.1(16/2).	C432002 (not given).	Check that inherited discriminants are not needed by an extension aggregate if the ancestor part is an expression or constrained subtype.	B-Test (try to give them).
(7)		Dynamic			C432003, C432004	Check that the components associated with an ancestor part subtype mark in an extension aggregate are initialized by default.	
					C432001	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).			
(8)		Dynamic			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 checks and constraint_error raised if needed.	
(9)		NonNormative		A note.			
(10)		NonNormative		Another note.			
(11)		NonNormative		Start of examples...			
(12)		NonNormative		...end of examples.			
(13)		NonNormative					
4.3.3	(1)	1	Redundant				
		2-3	Definitions	Positional and named array aggregates.			
(2)			Syntax				
(3/2)			Syntax				
(4)			Syntax				
(5/2)			Syntax				
			Negative	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).	
					B433001	Check that <> is not allowed in positional array aggregates other than in an others choice.	
(6)			Definitions	How n-dimensional array aggregates work. This simply determines the type of the aggregate for reference to other rules; real resolution issues are tested for			All
(7/2)	1	1	NameRes	Widely Used			
				4.3(3/2).			

2	NameRes			C87B31A (one-dim)		<p>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.</p> <p>Check that array component expressions in an array aggregate must have the component type of the array type of the aggregate.</p> <p>Check that the array component expressions in an array aggregate may have a limited type.</p>	C-Test: check multi-dimensional arrays.
		Negative		B43201D C433A01 (one-dim), C433A02 (two-dim)	All		
(8)				C87B31A (one-dim)		<p>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.</p> <p>Check that the choices in a named array aggregate must have the corresponding index type of the array type of the aggregate.</p>	C-Test: check multi-dimensional arrays.
		Negative		B43201D			
(9)	Legality	Subpart	All M-dimensional array aggregates will test.				
(10)	Legality	Negative Portion	Lead-in for following bullets			<p>Check that an n-dimensional array aggregate cannot be written as if it has some other dimensionality.</p>	B-Test. Try writing 2-dim arrays as 1-dim positional aggs.
		Negative		B43202C (operators, membership)		<p>Check that an others choice in an array aggregate is not allowed in contexts not described by 4.3.3(11-15).</p> <p>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.</p>	Are there any other such contexts?? I can't think of any, if there are any I've missed they should be tested.
(11/2)	Legality		The positive objectives really belong to 4.3.3(25), but here we can spread them out more clearly.	C43204A, C433001		<p>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.</p> <p>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.</p>	C-Test. Try entry calls and others => <>.
				C43204C		<p>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.</p> <p>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 component.</p>	C-Test. Check this in the expression of both a regular return statement and an extended return statement; try others => <>.
				C43204E, C433001			
				C43204E		<p>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.</p> <p>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.</p>	
				C43204F (subprogram), C43204G (entry), C43204H (generic unit)			

		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	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.	B-Test. Try the expression of an extended return statement and others => <>.
		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		B43202A	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 => <>.
(12)	Legality	Negative	The positive objectives really belong to 4.3.3(25), but here we can spread them out more clearly.	B43202A C43204I, C433001	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. 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.	
(13)	Legality	Negative	This is impossible, as an array object must be constrained. 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. 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.	B43202A	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.	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)	Legality	Negative			Check that the constraint of the applicable to a parenthesized expression is used to determine the bounds of a parenthesized array aggregate with an others choice.	C-Test. Check parenthesized expressions in all of the other contexts.

(16)	1	Definitions	Negative	Constraint that applies.		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)) is unconstrained.	B-Test. Check parenthesized expressions in all of the other contexts.
	2	Legality	Negative			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. 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.	B-Test. This was a fix for a contract model violation in Ada 83, and it should be tested. B-Test. This was a fix for a contract model violation in Ada 83, and it should be tested.
(17)		Legality			C43207D (range), C43208A (range), C43208B (range), C43224A (range attribute)	Check that a single nonstatic choice is allowed in an array aggregate. 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.	C-Test. Check that the single choice can be an expression. B-Test. This is marked as untested in ACATS 2.x.
(18)		Legality	Negative Widely Used	Any named notation array aggregate will test this.			
			Negative		B43201C	Check that a named array aggregate cannot contain two choices that cover the same value.	
					B43201C	Check that a named array aggregate is illegal if it does not cover a contiguous set of index values.	
(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. 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. Hard to imagine an implementation getting this wrong. B-Test. This is marked as untested in ACATS 2.x. (So is 4.2(7)).
(20)		StaticSem					
(21)		Dynamic	Portion	Lead-in for following bullets Arbitrary order is untestable. The conversion could fail, but any case that did would also fail 4.3.3(28), so those tests dominate.	B43209B	Check that a string literal used as a bottom level subaggregate of an array aggregate cannot be enclosed in parentheses.	
(22)		Dynamic	Not Testable	Arbitrary order is untestable. Normal evaluation is tested by every array aggregate.			
(23)		Dynamic	Widely Used				

				C43004A C43207D (2-dim range), C43208A (1-dim range), C43208B (2-dim range), C43210A (1-dim & 2-dim, named & others)	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).	
(23.1/2)	1	Dynamic	Widely Used	Any array aggregate	Check that a component expression in an array aggregate with multiple associated components is evaluated once for each associated component.		
	2			C433A01 (task, PO, lim-rec for 1-dim); C433A02 (task, PO, lim-rec for 2-dim). C433A04 (non-lim cases)	All	Check that for each association with a <> in an array aggregate, the component is initialized by default. Check that for a <> in an array aggregate with multiple associated components, each associated component is default initialized individually.	
(24) (25)		Dynamic Dynamic	Portion	Lead-in for following bullets. Tested under 4.3.3(11-15).	C433A03	All	
(26)		Dynamic		C43205A (subprogram, unconstrained), C43205G (subprogram, constrained), C43214B (subprogram, constrained, string literal) C43205B (unconstrained), C43205H (constrained), C43214C (constrained, string literal) C43205C (unconstrained), C43205I (constrained), C43214D (constrained, string literal) C43205D (constant, unconstrained), C43205J (objects, constrained), C43214E (objects, 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. 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. 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. 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.	3 2 2 4	C-Test. Try entry calls and string literals in unconstrained contexts. C-Test. Try string literals in unconstrained contexts. C-Test. Try string literals in unconstrained contexts. 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.

				<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>Check that the lower bound of a positional array aggregate or string literal in a membership is always that of the index subtype, even if the subtype is constrained.</p>	<p>C-Test. Not usefully testable because the bounds slide on the assignment.</p> <p>C-Test.</p> <p>C-Test.</p>
			C43205K, C43214F (string literal), C460010		
(27)	Dynamic		C42007E (string literal), C43205E (string literal)	<p>Check that the lower bound of a positional array aggregate or string literal used as the operand of a predefined operator is always that of the index subtype.</p> <p>Check that the bounds of a named array aggregate without others are determined by the choices.</p> <p>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.</p> <p>Check that Constraint_Error is raised if any non-null choice of a named array aggregate is outside of the index subtype.</p> <p>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.</p>	<p>C-Test. Try aggregates.</p> <p>C-Test. Try non-null aggregates.</p>
(28)	Dynamic		C43215A, C43215B C43207B (ranges), C43211A (ranges), C43214A (ranges)		C-Test. Try single choices.
(29/3)	Dynamic		C433001		
				<p>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.</p>	C-Test. We try this explicitly to test AI05-0037.
(30)	Dynamic		C43212A, C43212C	<p>Check that all subaggregates of a multidimensional array aggregate that correspond to the same index have the same bounds.</p>	
(31)	Dynamic	Portion		Defines the exception to raise for the previous rules.	
(32/2)	NonNormative			A note.	
(33)	NonNormative			Start of examples...	
(34)	NonNormative				
(35)	NonNormative				
(36)	NonNormative				

(37) NonNormative
 (38) NonNormative
 (39) NonNormative
 (40) NonNormative
 (41) NonNormative
 (42) NonNormative
 (43) NonNormative End of examples.

4.4	(1)	General				
	(2)	Syntax				
	(3)	Syntax				
	(4)	Syntax				
	(5)	Syntax				
	(6)	Syntax				
	(7)	Syntax				
			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			3 Check that a primary can be resolved because it must denote an object or value.	C-Test. Try a function overloaded with a procedure; use in an expression must resolve to the function call (even without parameters).
			Negative	B44001B (procedure), B44002B (tasks, entries), B44002C (exception)	3 Check that names that do not denote an object or value are not permitted as primaries.	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				
	(10)	Dynamic	Widely Used	Any object name used in an expression tests this. Either something happens, or it		
	(11)	Impl-Def	Not Testable	doesn't. Can't test that.		
	(12)	NonNormative		Start of examples...		
	(13)	NonNormative				
	(14)	NonNormative				
	(15/2)	NonNormative		...end of examples.		

Paragraphs:
8 166

Objectives with tests: 137
Objectives to test: 109
Total objectives: 213

Objectives with submitted tests: 1

Must be tested	Objectives with Priority 10	0
	Objectives with Priority 9	0
Important to test	Objectives with Priority 8	4
	Objectives with Priority 7	5
Valuable to test	Objectives with Priority 6	10
	Objectives with Priority 5	9
Ought to be tested	Objectives with Priority 4	23
	Objectives with Priority 3	28
Worth testing	Objectives with Priority 2	25
Not worth testing	Objectives with Priority 1	5
	Total:	109
	Objectives covered by new tests in ACATS 3.0	49
	Completely:	47