Ada 2022 Reference Card

italic Ada 2022
[] Optional {} Repeatable | Alternative ... Identical

S - subtype
P - program unit
C - component
F - function
E - entry declaration or exception
A - discriminated type or array
D - library declaration
V - value sequence
T - task
R - record
X - object
L - label

Attributes

Access `P|X'Access return access_type`
Access to subprogram or object.

Address `X|P|L'Address return System.Address`
Address of the first of the storage elements allocated to object, program unit, or label.

Adjacent `S'Adjacent (X,Towards:T) return T`
Adjacent floating point number to X in the direction of Towards.

Aft `S'Aft return universal_integer`
Number of decimal digits needed after the decimal point to accommodate the delta.

Alignment `S|X'Alignment return universal_integer`
Alignment of object.

Base `S'Base return S'Base`
Denotes the base unconstrained subtype of S.

Bit_Order `S'Bit_Order return System.Bit_Order`
Record subtype bit ordering.

Body_Version `P'Body_Version return String`
Version of the compilation unit that contains the body.

Callable `T'Callable return Boolean`
True when the task denoted by T is callable.

Caller `E'Caller return Task_ID`
Identifies the task whose call is now being serviced.

Ceiling `S'Ceiling (X:T) return T`
Smallest (most negative) integral value greater than or equal to argument.

Class `S'Class return class-wide type`
Returns the class-wide type of tagged type S.

Class `S'Class return class-wide type`
Returns the class-wide type for subtype S of an untagged private type whose full view is tagged.

Component_Size `X'Component_Size return universal_integer`
Size in bits of components of the array subtype or object.

Compose `S'Compose (Fraction:T;Exponent:universal_integer) return T`
Combine fraction and exponent into a floating point subtype.

Constrained `A'Constrained return Boolean`
True if A of discriminated type denotes a constant, a value, or a constrained variable.

Copy_Sign `S'Copy_Sign (Value,Sign:T) return T`
Result whose magnitude is that of float Value and whose sign is that of Sign.

Count `E'Count return universal_integer`
Number of calls presently queued on the entry.

Definite `S'Definite return Boolean`
True if the actual subtype of a a formal indefinite subtype is definite.

Delta `S'Delta return universal_real`
The delta of the fixed point subtype.

Denorm `S'Denorm return Boolean`
True if every value is expressible in canonical form with an an exponent of T'Machine_Emin.

Digits `S'Digits return universal_integer`
Number of digits of the decimal fixed point subtype.

Digits `S'Digits return universal_integer`
Number of decimal mantissa digits for floating point subtype.

Enum_Rep `S'Enum_Rep (X:S'Base) return universal_integer`
Return the number representing a given enumeration literal.

Enum_Val `S'Enum_Val (X:universal_integer) return S'Base`
Return the enumeration literal represented by a given number.

Exponent `S'Exponent (X:T) return universal_integer`
Normalized exponent of the floating point argument.

External_Tag `S'External_Tag return String`
An external string representation of the tagged type.

First `A'First (N) return index_type`
Lower bound of N-th index of [constrained] array type.

First `A'First return index_type`
Lower bound of first index of [constrained] array type.

First `S'First return S`
Lower bound of the range of scalar subtype.

First_Bit `R.C'First_Bit return universal_integer`
Bit offset, from the start of the first of the storage elements occupied by C, of the first bit occupied by C.

First_Valid `S'First_Valid return S`
Denotes the smallest value that belongs to S and satisfies the predicates of S.

Floor `S'Floor (X:T) return T`
Largest integral value less than or equal to the argument.

Fore `S'Fore return universal_integer`
Minimum number of characters needed before the decimal point.

Fraction `S'Fraction (X:T) return T`
Decompose floating point argument into fractional part.

Has_Same_Storage `X'Has_Same_Storage (X2:any_type) return Boolean`
Returns True if the representation of X2 occupies exactly the same bits as the representation of X and the objects occupy at least one bit.

Identity `E'Identity return Exception_Id`
Yields unique identity of the exception.

Identity `T'Identity return Task_Id`
Yields unique identity of the task.

Image `S'Image (X) return String`
Image of the value of X as a String.

Image `X'Image return String`
Image of the value of X as a String.

Index `E'Index return entry_index_subtype`
Within a precondition or postcondition expression for entry family E, denotes the value of the entry index for the call of E.

Class'Input `S'Class'Input (Stream:access Ada.Streams.Root_Stream_Type'Class) return T'Class`
First reads the external tag from Stream and determines the corresponding internal tag which can raise Tag_Error and then dispatches to the subprogram denoted by the Input attribute of the specific type identified by the internal tag.

Input `S'Input (Stream:access Ada.Streams.Root_Stream_Type'Class) return T`
Reads and returns one value from the Stream argument.

Last `A'Last (N) return index_type`
Upper bound of N-th index range of [constrained] array type.

Last `A'Last return index_type`
Upper bound of first index range of [constrained] array type.

Last `S'Last return T`
Upper bound of the range of scalar subtype.

Last_Bit `R.C'Last_Bit return universal_integer`
Bit offset, from the start of the first of the storage elements occupied by C, of the last bit occupied by C.

Leading_Part `S'Leading_Part (X:T;Radix_Digits:universal_integer) return T`
The leading part of floating point value with number of radix digits given by second argument.

Length `A'Length (N) return universal_integer`
Number of values of the N-th index range of [constrained] array type.

Length `A'Length return universal_integer`
Number of values of the first index range of [constrained] array type.

Machine `S'Machine (X:T) return T`
Machine representation of floating point argument.

Machine_Emax `S'Machine_Emax return universal_integer`
Largest (most positive) value of floating point exponent.

Machine_Emin `S'Machine_Emin return universal_integer`
Smallest (most negative) value of floating point exponent.

Machine_Mantissa `S'Machine_Mantissa return universal_integer`
Number of digits in machine representation of mantissa.

Machine_Overflows `S'Machine_Overflows return Boolean`
True if numeric overflow detected for fixed or floating point.

Machine_Radix `S'Machine_Radix return universal_integer`
Radix of machine representation of the fixed or floating point.

Machine_Rounds `S'Machine_Rounds return Boolean`
True if rounding is performed on inexact results of the fixed or floating point.

Machine_Rounding

`S'Machine_Rounding (X:T) return T`
Yields the integral value nearest to X.

Max `S'Max (X1,X2:S) return S`
Returns the greater of the values of the two parameters.

Max_Alignment_For_Allocation

`S'Max_Alignment_For_Allocation return universal_integer`
Maximum value for Alignment that can be requested by the implementation via Allocate for an access type whose designated subtype is S.

Max_Size_In_Storage_Elements

`S'Max_Size_In_Storage_Elements return universal_integer`
Maximum value for Size_In_Storage_Elements that will be requested via Allocate.

Min `S'Min (X1,X2:S) return S`
The lesser of the values of the two scalar arguments.

Mod `S'Mod (X:T) return S`
Will correctly convert any integer type to a given modular type (S), using wraparound semantics.

Model `S'Model (X:T) return T`
Model number of floating point type.

Model_Emin `S'Model_Emin return universal_integer`
Model number version of S"Machine_Emin.

Model_Epsilon

`S'Model_Epsilon return universal_real`
Absolute difference between the model number 1.0 and the next model number above for subtype.

Model_Mantissa

`S'Model_Mantissa return universal_integer`
Model number version of S'Machine_Mantissa.

Model_Small `S'Model_Small return universal_real`
Smallest positive model number of subtype.

Modulus `S'Modulus return universal_integer`
The modulus of the modular subtype.

Object_Size `S'Object_Size return universal_integer`
The size of an object of subtype S. Must be a value that the compiler is able to allocate (usually an entire storage unit).

Old `X'Old return T`
The value of X on entry, has same type as X.

Class'Output `S'Class'Output (Stream:access Ada.Streams.Root_Stream_-Type'Class;X)`
Writes the external tag of Item to Stream and then dispatches to the subprogram denoted by the Output attribute of the specific type identified by the tag.

Output `S'Output (Stream:access Ada.Streams.Root_Stream_Type'Class;X)`
Writes the value of X to Stream, including any bounds or discriminants.

Overlaps_Storage

`X'Overlaps_Storage (X2) return Boolean`
Returns True if the representation of X2 shares at least one bit with the representation of the object denoted by X.

Parallel_Reduce

`X'Parallel_Reduce (Reducer,Initial_Value)`
Reduction expression that yields a result equivalent to replacing the attribute identifier with Reduce and the prefix of the attribute with the value_-sequence.

Partition_ID `D'Partition_ID return universal_integer`
Identifies the partition in which D was elaborated.

Pos `S'Pos (X) return universal_integer`
Position of the value of the discrete subtype argument.

Position `R.C'Position return universal_integer`
Same as R.C'Address - R'Address for component C.

Pred `S'Pred (X) return S`
Predecessor of the argument.

Preelaborable_Initialization

`S'Preelaborable_Initialization return Boolean`
Returns whether the type of S has preelaborable initialization.

Priority `P'Priority return System.Any_Priority`
Returns the priority of P.

Put_Image `S'Put_Image (Buffer:Ada.Strings.Text_Buffers.Root_Buffer_-Type'Class;X)`
Writes an image of the value of X.

Range `A'Range return range`
Equivalent to the range A'First .. A'Last.

Range `S'Range return range`
Equivalent to the range S'First .. S'Last.

Range `A'Range (N) return range`
Equivalent to the range A'First(N) .. A'Last(N).

Read `S'Read (Stream:access Ada.Streams.Root_Stream_Type'Class;X:out T)`
Reads the value of X from Stream.

Read `S'Class'Read(Stream:access Ada.Streams.Root_Stream_-Type'Class;X:out T'Class)`
Reads the value of X from Stream.

Reduce `X|V'Reduce(Reducer, Initial_Value)`
This attribute represents a reduction expression, and is in the form of a reduction_attribute_reference.

Relative_Deadline

`P'Relative_Deadline return Ada.Real_Time.Time_Span`
Relative deadline of P.

Remainder `S'Remainder (X,Y:T) return T`
Remainder after dividing the first floating point argument by its second.

Result `F'Result return X`
Within a postcondition expression for F, denotes the return object of the function call for which the postcondition expression is evaluated.

Round `F'Round (X) return S`
Fixed-point value obtained by rounding X (away from 0, if X is midway between two values).

Rounding `S'Rounding (X:T) return T`
Floating-point integral value nearest to X, rounding away from zero if X lies exactly halfway between two integers.

Safe_First `S'Safe_First return universal_real`
Returns lower bound of the safe range.

Safe_Last `S'Safe_Last return universal_real`
Returns upper bound of the safe range

Scale `S'Scale return universal_integer`
Position of the fixed-point relative to the rightmost significant digits of values of subtype S.

Scaling `S'Scaling (X:T;Adjustment:universal_integer) return T`
Scaling by a power of the hardware radix.

Signed_Zeros `S'Signed_Zeros return Boolean`
True if positive and negative signed zeros are representable.

Size `S'Size universal_integer`
Size in bits of objects instantiated from subtype.

Size `X'Size return universal_integer`
Size in bits of the representation of the object.

Small `S'Small return universal_real`
Small of the fixed-point type.

Storage_Pool `S'Storage_Pool return Root_Storage_Pool'Class`
Returns Storage pool of the access subtype.

Storage_Size `S'Storage_Size return universal_integer`
Number of storage elements reserved for the storage pool.

Storage_Size `T'Storage_Size return universal_integer`
Number of storage elements reserved for the task.

Stream_Size `S'Stream_Size return universal_integer`
Number of bits read from or written to a stream by the default implementations of S'Read and S'Write.

Succ `S'Succ (X:T) return T`
Returns successor of the X.

Tag `X|S'Tag return Tag`
Returns the tag of the [class-wide] tagged type or of object X that is a class-wide tagged type.

Terminated `T'Terminated return Boolean`
Returns True if the task denoted by T is terminated.

Truncation `S'Truncation (X:T) return T`
Returns the value Ceiling(X) when X is negative, else Floor(X).

Unbiased_Rounding

`S'Unbiased_Rounding (X:T) return T`
Integral value nearest to X, rounding toward the even integer if X lies exactly halfway between two integers.

Unchecked_Access

`X'Unchecked_Access (X:T) return access type`
Same as X'Access but lacks accessibility rules/checks.

Val `S'Val (universal_integer) return S`
Value of the discrete subtype whose position number equals the value of argument.

Val `X'Valid return Boolean`
True if and only if the scalar object denoted by X is normal and has a valid representation.

Value `S'Value (X:String) return S`
Returns a value of the subtype given an image of the value as a String argument.

Version `P'Version return String`
Yields string that identifies the version of the compilation unit that contains the declaration of the program unit.

Wide_Image `S'Wide_Image (X:S) return Wide_String`
Image of the value of X as a Wide_String.

Wide_Image `X'Wide_Image return Wide_String`
Image of the value of X as a Wide_String.

Wide_Value `S'Wide_Value (X:String) return S`
Returns a value given an image of the value as a Wide_String argument (X).

Wide_Width `S'Wide_Width return universal_integer`
Maximum length of Wide_String returned by S'Image.

Wide_Wide_Image
`S'Wide_Wide_Image (X:S) return Wide_Wide_String`
Image of the value of X as a Wide_Wide_String.

Wide_Wide_Image
`X'Wide_Wide_Image return Wide_Wide_String`
Image of the value of X as a Wide_Wide_String.

Wide_Wide_Value
`S'Wide_Wide_Value (X:String) return S`
Returns a value given an image of the value as a Wide_Wide_String argument (X).

Wide_Wide_Width
`S'Wide_Wide_Width return universal_integer`
Maximum length of Wide_Wide_String returned by S'Image.

Width `S'Width return universal_integer`
Maximum length of String returned by S'Image.

Class'Write `S'Class'Write (Stream:access Ada.Streams.Root_Stream_Type'Class;X:T'Class)`
Writes X to Stream.

Write `S'Write (Stream:access Ada.Streams.Root_Stream_Type'Class;X:T)`
Writes X to Stream.

Aspects

Address `X|P|L with Address => System.Address`
Address of the first of the storage elements allocated.

Aggregate `S with Aggregate => (aggregate)`
Mechanism to define user-defined aggregates.

Alignment `X|S with Alignment => universal_integer`
Alignment of object or subtype.

All_Calls_Remote
`P with All_Calls_Remote => Boolean`
All indirect or dispatching remote subprogram calls, and all direct remote subprogram calls, should use the Partition Communication Subsystem.

Allows_Exit `P with Allows_Exit => Boolean`
An indication of whether a subprogram will operate correctly for arbitrary transfers of control.

Asynchronous `P with Asynchronous => Boolean`
Remote procedure calls are asynchronous; the caller continues without waiting for the call to return.

Atomic `S|X|C with Atomic => Boolean`
Declare that a type, object, or component is atomic.

Atomic_Components
`A|X with Atomic_Components => Boolean`
Declare that the components of an array type or object are atomic.

Attach_Handler
`P with Attach_Handler => Ada.Interrupts.Interrupt_Id`
Protected procedure is attached to an interrupt.

Bit_Order `S with Bit_Order => System.Bit_Order`
Order of bit numbering in a record_representation_clause.

Component_Size
`A|X with Component_Size => universal_integer`
Size in bits of a component of an array type.

Constant_Indexing
`S with Constant_Indexing => P`
Defines function to implement user-defined indexed_components.

Convention `S|P with Convention => convention_identifier`
Calling convention or other convention used for interfacing to other languages.

CPU `T with CPU => System.Multiprocessors.CPU_Range`
Processor on which a given task, or calling task for a protected operation, should run.

Default_Component_Value
`S with Default_Component_Value => Component_Type`
Default value for the components of an array-of-scalar subtype.

Default_Initial_Condition
`S with Default_Initial_Condition => Boolean`
If the Default_Initial_Condition aspect is specified for a type T, then the default initial condition expression applies to S and to all descendants of S.

Default_Iterator
`S with Default_Iterator => P`
Default iterator to be used in for loops.

Default_Value `S with Default_Value => scalar value`
Default value for a scalar subtype.

Discard_Names
`S|E with Discard_Names => Boolean`
Requests a reduction in storage.

Dispatching `P with Dispatching => dispatching_operation_specifier`
.

Dispatching_Domain
`T with Dispatching_Domain => System.Multiprocessors.Dispatching_Domains.Dispatching_Domain`
Domain (group of processors) on which a given task should run.

Dynamic_Predicate
`S with Dynamic_Predicate => Boolean`
Condition that will hold true for objects of a given subtype; the subtype is not static.

Elaborate_Body
`D with Elaborate_Body => Boolean`
A given package will have a body, and that body is elaborated immediately after the declaration.

Exclusive_Functions
`S with Exclusive_Functions => Boolean`
Specifies mutual exclusion behavior of protected functions in a protected type.

Export `P|X with Export => Boolean`
Entity is exported to another language.

External_Name `P|X with External_Name => String`
Name used to identify an imported or exported entity.

External_Tag `S with External_Tag => String`
Unique identifier for a tagged type in streams.

Full_Access_Only
`X|C with Full_Access_Only => Boolean`
Declare that a volatile type, object, or component is full access.

Global `D with Global => global_aspect_definition`
Global object usage contract.

Global'Class `D with Global'Class => global_aspect_definition`
Global object usage contract inherited on derivation.

Implicit_Dereference
`A with Implicit_Dereference => Discriminant`
Mechanism for user-defined implicit .all.

Import `P|X with Import => Boolean`
Entity is imported from another language.

Independent `X|S with Independent => Boolean`
Declare that a type, object, or component is independently addressable.

Independent_Components
`A|R with Independent_Components => Boolean`
Declare that the components of an array or record type, or an array object, are independently addressable.

Inline `P|E with Inline => Boolean`
For efficiency, Inline calls are requested for a subprogram.

Input `Input`
Function to read a value from a stream for a given type, including any bounds and discriminants.

Input'Class `Input'Class`
Function to read a value from a stream for a the class-wide type associated with a given type, including any bounds and discriminants.

Integer_Literal `Integer_Literal`
Defines a function to implement user-defined integer literals.

Interrupt_Handler
`Interrupt_Handler`
Protected procedure may be attached to interrupts.

Interrupt_Priority
`Interrupt_Priority`
Priority of a task object or type, or priority of a protected object or type; the priority is in the interrupt range.

Iterator_Element

`Iterator_Element`
Element type to be used for user-defined iterators.

Iterator_View

`Iterator_View`
An alternative type to used for container element iterators.

Layout

`Layout (record)`
Layout of record components. Specified by a record_representation_clause, not by an aspect_specification.

Link_Name

`Link_Name`
Linker symbol used to identify an imported or exported entity.

Machine_Radix

`Machine_Radix`
Radix (2 or 10) that is used to represent a decimal fixed point type.

Max_Entry_Queue_Length

`Max_Entry_Queue_Length`
The maximum entry queue length for a task type, protected type, or entry.

No_Controlled_Parts

`No_Controlled_Parts`
A specification that a type and its descendants do not have controlled parts.

No_Return

`P with No_Return => Boolean`
Procedure cannot return normally; it may raise an exception, loop forever, or terminate the program.

Nonblocking

`Nonblocking`
Specifies that an associated subprogram does not block.

Output

`Output`
Procedure to write a value to a stream for a given type, including any bounds and discriminants.

Output'Class

`Output'Class`
Procedure to write a value to a stream for a the class-wide type associated with a given type, including any bounds and discriminants.

Pack

`Pack`
Minimize storage when laying out records and arrays.

Parallel_Calls

`Parallel_Calls`
Specifies whether a given subprogram is expected to be called in parallel.

Parallel_Iterator

`Parallel_Iterator`
An indication of whether a subprogram may use multiple threads of control to invoke a loop body procedure.

Post

`with Post => Condition`
Postcondition; a condition that will hold true after a call.

Post'Class

`with Post'Class`
Postcondition that applies to corresponding subprograms of descendant types.

Pre

`with Pre => Condition`
Precondition; a condition that is expected to hold true before a call.

Pre'Class

`with Pre'Class => Condition`
Precondition that applies to corresponding subprograms of descendant types.

Predicate_Failure

`Predicate_Failure`
Action to be performed when a predicate check fails.

Preelaborable_Initialization

`Preelaborable_Initialization`
Declares that a type has preelaborable initialization.

Preelaborate

`Preelaborate`
Code execution during elaboration is avoided for a given package.

Priority

`Priority`
Priority of a task object or type, or priority of a protected object or type; the priority is not in the interrupt range.

Pure

`D with Pure`
Side effects are avoided in the subprograms of a given package.

Put_Image

`Put_Image`
Procedure to define the image of a given type.

Read

`Read`
Procedure to read a value from a stream for a given type.

Read'Class

`Read'Class`
Procedure to read a value from a stream for the class-wide type associated with a given type.

Real_Literal

`Real_Literal`
Defines a function or functions to implement user-defined real literals.

Relative_Deadline

`T with Relative_Deadline => RD`
Ensures that the absolute deadline of the task when created is RD of type Real_Time.Time_Span.

Remote_Call_Interface

`Remote_Call_Interface`
Subprograms in a given package may be used in remote procedure calls.

Remote_Types

`Remote_Types`
Types in a given package may be used in remote procedure calls.

Shared_Passive

`Shared_Passive`
A given package is used to represent shared memory in a distributed system.

Size

`Size(S|X)`
Size in bits of objects instantiated from subtype.

Small

`Small`
Scale factor for a fixed point type.

Stable_Properties

`Stable_Properties`
A list of functions describing characteristics that usually are unchanged by primitive operations of the type or an individual primitive subprogram.

Stable_Properties'Class

`Stable_Properties'Class`
A list of functions describing characteristics that usually are unchanged by primitive operations of a class of types or a primitive subprogram for such a class.

Static

`Static`
Specifies that an associated expression function can be used in static expressions.

Static_Predicate

`Static_Predicate`
Condition that will hold true for objects of a given subtype; the subtype may be static.

Storage_Pool

`Storage_Pool`
Pool of memory from which new will allocate for a given access type.

Storage_Size

`Storage_Size (access)`
Sets memory size for allocations for an access type.

Storage_Size

`Storage_Size (task)`
Size in storage elements reserved for a task type or single task object.

Stream_Size

`Stream_Size`
Size in bits used to represent elementary objects in a stream.

String_Literal

`String_Literal`
Defines a function to implement user-defined string literals.

Synchronization

`P with Synchronization => By_Entry | By_Protected_Procedure | Optional`
Defines whether a given primitive operation of a synchronized interface will be implemented by an entry or protected procedure.

Type_Invariant

`Type_Invariant`
Condition that will hold true for all objects of a type.

Type_Invariant'Class

`Type_Invariant'Class`
A condition that will hold true for all objects in a class of types.

Unchecked_Union

`Unchecked_Union`
Type is used to interface to a C union type.

Use_Formal

`Use_Formal`
Generic formal parameters used in the implementation of an entity.

Variable_Indexing

`Variable_Indexing`
Defines function(s) to implement user-defined indexed_components.

Volatile

`S|X|C with Volatile`
Declare that a type, object, or component is volatile.

Volatile_Components

`A|X with Volatile_Components`
Declare that the components of an array type or object are volatile.

Write

`Write`
Procedure to write a value to a stream for a given type.

Write'Class

`Write'Class`
Procedure to write a value to a stream for a the class-wide type associated with a given type.

Yield

`Yield`
Ensures that a callable entity includes a task dispatching point.

PRAGMAS

Admission_Policy

```
pragma Admission_Policy (policy_identifier)
```
An admission policy governs the order in which competing tasks are evaluated for acquiring the execution resource associated with a protected object.

All_Calls_Remote

```
pragma All_Calls_Remote [(library_unit_name)]
```
Force all calls on a remote-call-interface library unit from other library units in the same active partition to be remote.

Assert

```
pragma Assert([Check =>] boolean_expression[, [Message =>]
string_expression])
```
Raises Assertion_Error exception with an optional message when the expression is false.

Assertion_Policy

```
pragma Assertion_Policy(Check | Ignore)
```
Enables or disables assertions including pre and post conditions.

Assertion_Policy

```
pragma Assertion_Policy(Pre => Check | Ignore, Post => Check |
Ignore)
```
Enables or disables pre and post conditions.

Asynchronous

```
pragma Asynchronous (local_name)
```
The return message is dispensed with for a remote call on a procedure marked asynchronous.

Atomic

```
pragma Atomic (local_name)
```
Is used with types and variables to specify that the code generated must read and write the type or variable from memory atomically, i.e. as a single/non-interruptible operation.

Atomic_Components

```
pragma Atomic_Components (array_local_name)
```
The components of the named array or every array of the named type is to be examined and updated atomically.

Attach_Handler

```
pragma Attach_Handler (handler_name, expression)
```
The handler procedure is attached to the specified interrupt.

Conflict_Check_Policy

```
pragma Conflict_Check_Policy (policy_identifier[,
policy_identifier])
```
This subclause determines what checks are performed relating to possible concurrent conflicting actions.

Convention

```
pragma Convention ([Convention =>] convention_identifier,
[Entity =>] local_name)
```
Directs the compiler to represent a type or subprogram using a foreign language convention.

CPU

```
pragma CPU (System.Multiprocessors.CPU_Range)
```
Processor on which a given task, or calling task for a protected operation, should run.

Default_Storage_Pool

```
pragma Default_Storage_Pool (storage_pool_indicator)
```
Specifies the storage pool that will be used in the absence of an explicit specification of a storage pool or storage size for an access type.

Detect_Blocking

```
pragma Detect_Blocking
```
Raises Program_Error when a potentially blocking operation is detected that occurs during the execution of a protected operation or a parallel construct defined within a compilation unit to which the pragma applies.

Discard_Names

```
pragma Discard_Names [([On => ] local_name)]
```
Reduce the memory needed to store names of Ada entities, where no operation uses those names.

Dispatching_Domain

```
pragma Dispatching_Domain (expression)
```
Domain (group of processors) on which a given task should run.

Elaborate

```
pragma Elaborate (library_unit_name, ...)
```
Guarantees that both the spec and body of its argument will be elaborated prior to the unit with the pragma.

Elaborate_All

```
pragma Elaborate_All (library_unit_name, ...)
```
Guarantees that both the spec and body of its argument will be elaborated prior to the unit with the pragma, as well as all units withed by the spec and body of the argument, recursively.

Elaborate_Body

```
pragma Elaborate_Body [(library_unit_name)]
```
Requires that the body of a unit is elaborated immediately after its spec. This restriction guarantees that no client scenario can invoke a server target before the target body has been elaborated.

Export

```
pragma Export ([Convention =>] convention_identifier, [Entity
=>] local_name [, [External_Name =>] string_expression]
[,[Link_Name =>] string_expression])
```
Directs the compiler to make available subprograms or data objects written in Ada to foreign computer languages.

Generate_Deadlines

```
pragma Generate_Deadlines
```
Makes the deadline of a task be recomputed each time it becomes ready. The new deadline is the value of Real_Time.Clock at the time the task is added to a ready queue plus the value returned by Get_Relative_Deadline.

Import

```
pragma Import ([Convention =>] convention_identifier, [Entity
=>] local_name [, [External_Name =>] string_expression]
[,[Link_Name =>] string_expression])
```
Directs the compiler to use code or data objects written in a foreign computer language.

Independent

```
pragma Independent (component_local_name)
```
Declare that a type, object, or component is independently addressable.

Independent_Components

```
pragma Independent_Components (local_name)
```
Declare that the components of an array or record type, or an array object, are independently addressable.

Inline

```
pragma Inline (name , ...)
```
Directs the compiler to inline the code of the given subprogram, making execution faster by eliminating overhead of the subprogram call.

Inspection_Point

```
pragma Inspection_Point [(object_name , ...)]
```
Directs the compiler to ensure that the specified variable is available where the pragma appears. This pragma aids in debugging.

Interrupt_Handler

```
pragma Interrupt_Handler (handler_name)
```
Tell the compiler this is an interrupt handler.

Interrupt_Priority

```
pragma Interrupt_Priority [(expression)]
```
Assigns the given priority to the whole protected object. No other interrupts at or below that level will be enabled whenever the procedure is executing.

Linker_Options

```
pragma Linker_Options (string_expression)
```
Used to specify the system linker parameters needed when a given compilation unit is included in a partition.

List

```
pragma List (identifier)
```
Specifies that listing of the compilation is to be continued (On) or suspended (Off) until a List pragma with the opposite argument is given within the same compilation.

Locking_Policy

```
pragma Locking_Policy (policy_identifier)
```
Chooses locking policy.

No_Return

```
pragma No_Return (subprogram_local_name,
subprogram_local_name)
```
States that a procedure will never return normally; that is, it will raise an exception, loop endlessly, or terminate the program.

Normalize_Scalars

```
pragma Normalize_Scalars
```
Directs the compiler to initialize otherwise uninitialized scalar variables with predictable values. If possible, the compiler will choose out-of-range values.

Optimize

```
pragma Optimize (identifier)
```
Gives advice to the implementation as to whether time (Time) or space (Space) is the primary optimization criterion, or that optional optimizations should be turned off (Off).

Pack

```
pragma Pack (first_subtype_local_name)
```
Directs the compiler to use type representations that favor conservation of storage space, rather than ease of access.

Page

```
pragma Page
```
Specifies that the program text which follows the pragma should start on a new page (if the compiler is currently producing a listing).

Partition_Elaboration_Policy

```
pragma Partition_Elaboration_Policy (policy_identifier)
```
Specifies the elaboration policy for a partition.

Preelaborable_Initialization

```
pragma Preelaborable_Initialization (direct_name)
```
Specifies that all objects of the type have preelaborable initialization expressions.

Preelaborate

```
pragma Preelaborate [(library_unit_name)]
```
Slightly less restrictive than pragma Pure, but still strong enough to prevent access before elaboration problems within a unit.

Priority

```
pragma Priority (Integer)
```
Sets a task's priority. The pragma must be called in the task specification.

Priority_Specific_Dispatching

```
pragma Priority_Specific_Dispatching (policy_identifier,
first_priority_expression, last_priority_expression)
```
Specifies the task dispatching policy for the specified range of priorities.

Profile
`pragma Profile (profile_identifier , profile_pragma_argument_association)`
Expresses the user's intent to abide by a set of Restrictions or other specified run-time policies. These may facilitate the construction of simpler run-time environments.

Pure
`pragma Pure [(library_unit_name)]`
Guarantees that no scenario within the unit can result in an access before elaboration problem.

Queuing_Policy
`pragma Queuing_Policy (FIFO_Queuing|Priority_Queuing)`
Defines the queuing policy used on task entry to an Ada partition.

Relative_Deadline
`pragma Relative_Deadline (Real_Time.Time_Span)`
Defines deadline.

Remote_Call_Interface
`pragma Remote_Call_Interface [(library_unit_name)]`
Categorizes a library-unit as a Remote-Call-Interface.

Remote_Types
`pragma Remote_Types [(library_unit_name)]`
Categorizes a library-unit as a Remote-Type.

Restrictions
`pragma Restrictions (restriction, ...)`
Used to forbid the utilization of some language features.

Reviewable
`pragma Reviewable`
Directs the compiler to provide information that aids inspection of the program's object code.

Shared_Passive
`pragma Shared_Passive [(library_unit_name)]`
Allows the use of passive partitions in the context described in the Ada Reference Manual; i.e., for communication between separate partitions of a distributed application using the features in Annex E.

Storage_Size
`pragma Storage_Size (expression)`
Specifies the amount of space to be allocated for the task stack. This cannot be extended, and if the stack is exhausted, then Storage_Error will be raised (if stack checking is enabled).

Suppress
`pragma Suppress (identifier)`
Gives the compiler permission to omit checks, but does not require the compiler to omit checks.

Task_Dispatching_Policy
`pragma Task_Dispatching_Policy (policy_identifier)`
Chooses scheduling policies.

Unchecked_Union
`pragma Unchecked_Union (first_subtype_local_name)`
Denotes an unconstrained discriminated record subtype having a variant_part.

Unsuppress
`pragma Unsuppress (identifier)`
Unsuppresses a given check.

Volatile
`pragma Volatile (local_name)`
Is used with types and variables to specify that the variable in question may suddenly change in value. For example, this may occur due to a device writing to a shared buffer.

Volatile_Components
`pragma Volatile_Components (array_local_name)`
notDeclares that the components of the array type — but not the array type itself — are volatile.e

STANDARD LIBRARY

package Standard

- `Boolean` True or False
- `Integer` Implementation defined
- `Natural` Integers >= 0
- `Positive` Integers > 0
- `Float` Implementation defined
- `Character` 8-bit ASCII/ISO 8859-1
- `Wide_Character` 16-bit ISO 10646
- `Wide_Wide_Character` 32-bit ISO 10646:2020
- `String` Array of Characters
- `Wide_String` Array of Wide_Character
- `Wide_Wide_String` Array of Wide_Wide_Character
- `Duration` Time in seconds
- `Constraint_Error` Predefined exception
- `Program_Error` Predefined exception
- `Storage_Error` Predefined exception
- `Tasking_Error` Predefined exception

package Ada

- Assertions
- Asynchronous_Task_Control
- Calendar
  - Arithmetic
  - Formatting
  - Time_Zones
- Characters
  - Conversions
  - Handling
  - Latin_1
- Command_Line
- Complex_Text_IO
- Containers
  - Bounded_Doubly_Linked_Lists
  - Bounded_Hashed_Maps
  - Bounded_Hashed_Sets
  - Bounded_Indefinite_Holders
  - Bounded_Multiway_Trees
  - Bounded_Ordered_Maps
  - Bounded_Ordered_Sets
  - Bounded_Priority_Queues
  - Bounded_Synchronized_Queues
  - Bounded_Vectors
  - Doubly_Linked_Lists
  - Generic_Array_Sort
  - Generic_Constrained_Array_Sort
  - Generic_Sort
  - Hashed_Maps
  - Hashed_Sets
  - Indefinite_Doubly_Linked_Lists
  - Indefinite_Hashed_Maps
  - Indefinite_Hashed_Sets
  - Indefinite_Holders
  - Indefinite_Multiway_Trees
  - Indefinite_Ordered_Maps
  - Indefinite_Ordered_Sets
  - Indefinite_Vectors
  - Multiway_Trees
  - Ordered_Maps
  - Ordered_Sets
  - Synchronized_Queue_Interfaces
  - Unbounded_Priority_Queues
  - Unbounded_Synchronized_Queues
  - Vectors
- Decimal
- Direct_IO
- Directories
  - Hierarchical_File_Names
  - Information
- Dispatching
  - EDF
  - Non_Preemptive
  - Round_Robin
- Dynamic_Priorities
- Environment_Variables
- Exceptions
- Execution_Time
  - Group_Budgets
  - Interrupts
  - Timers
- Finalization
- Float_Text_IO
- Float_Wide_Text_IO
- Float_Wide_Wide_Text_IO
- Integer_Text_IO
- Integer_Wide_Text_IO
- Integer_Wide_Wide_Text_IO
- Interrupts
  - Names
- IO_Exceptions
- Iterator_Interfaces
- Locales
- Numerics
  - *Big_Numbers*
    - *Big_Integers*
    - *Big_Reals*
  - Complex_Arrays
  - Complex_Elementary_Functions
  - Complex_Types
  - Discrete_Random

- Elementary_Functions
- Float_Random
- Generic_Complex_Arrays
- Generic_Complex_Elementary_Functions
- Generic_Complex_Types
- Generic_Elementary_Functions
- Generic_Real_Arrays
- Real_Arrays

Real_Time
- Timing_Events

Sequential_IO

Storage_IO

Streams
- Storage_Streams
  - Bounded_FIFO_Streams
  - FIFO_Streams
- Stream_IO

Strings
- Bounded
  - Equal_Case_Insensitive
  - Hash
  - Hash_Case_Insensitive
  - Less_Case_Insensitive
- Equal_Case_Insensitive
- Fixed
  - Equal_Case_Insensitive
  - Hash
  - Hash_Case_Insensitive
  - Less_Case_Insensitive
- Hash
- Hash_Case_Insensitive
- Less_Case_Insensitive
- Maps
  - Constants
- Text_Buffers
  - Bounded
  - Unbounded
- Unbounded
  - Equal_Case_Insensitive
  - Hash
  - Hash_Case_Insensitive
  - Less_Case_Insensitive
- UTF_Encoding
  - Conversions
  - Strings
  - Wide_Strings
  - Wide_Wide_Strings
- Wide_Bounded
  - Wide_Equal_Case_Insensitive
  - Wide_Hash
  - Wide_Hash_Case_Insensitive
- Wide_Equal_Case_Insensitive
- Wide_Fixed
  - Wide_Equal_Case_Insensitive
  - Wide_Hash
  - Wide_Hash_Case_Insensitive
- Wide_Hash
- Wide_Hash_Case_Insensitive
- Wide_Maps
  - Wide_Constants
- Wide_Unbounded
  - Wide_Equal_Case_Insensitive
  - Wide_Hash
  - Wide_Hash_Case_Insensitive
- Wide_Wide_Bounded
  - Wide_Wide_Equal_Case_Insensitive
  - Wide_Wide_Hash
  - Wide_Wide_Hash_Case_Insensitive
- Wide_Wide_Equal_Case_Insensitive
- Wide_Wide_Fixed
  - Wide_Wide_Equal_Case_Insensitive
  - Wide_Wide_Hash
  - Wide_Wide_Hash_Case_Insensitive
- Wide_Wide_Hash
- Wide_Wide_Hash_Case_Insensitive
- Wide_Wide_Maps
  - Wide_Wide_Constants
- Wide_Wide_Unbounded
  - Wide_Wide_Equal_Case_Insensitive
  - Wide_Wide_Hash
  - Wide_Wide_Hash_Case_Insensitive
- Synchronous_Barriers
- Synchronous_Task_Control
  - EDF
- Tags
  - Generic_Dispatching_Constructor
- Task_Attributes
- Task_Identification
- Task_Termination
- Text_IO
  - Bounded_IO
  - Complex_IO
  - Editing
  - Text_Streams
  - Unbounded_IO
- Unchecked_Conversion
- Unchecked_Deallocate_Subpool
- Unchecked_Deallocation
- Wide_Characters
  - Handling
- Wide_Command_Line
- Wide_Directories
- Wide_Environment_Variables
- Wide_Text_IO
  - Complex_IO
  - Editing
  - Text_Streams
  - Wide_Bounded_IO
  - Wide_Unbounded_IO
- Wide_Wide_Characters
  - Handling
- Wide_Wide_Command_Line
- Wide_Wide_Directories
- Wide_Wide_Environment_Variables
- Wide_Wide_Text_IO
  - Complex_IO
  - Editing
  - Text_Streams
  - Wide_Wide_Bounded_IO
  - Wide_Wide_Unbounded_IO

package Interfaces
- C
  - Pointers
  - Strings
- COBOL
- Fortran

package System
- Address_To_Access_Conversions
- *Atomic_Operations*
  - *Exchange*
  - *Integer_Arithmetic*
  - *Modular_Arithmetic*
  - *Test_And_Set*
- Machine_Code
- Multiprocessors
  - Dispatching_Domains
- RPC
- Storage_Elements
- Storage_Pools
  - Subpools