**Interfacing with Spark**

As Spark is a subset of Ada, it can be complicated to get code that is written in full Ada and Spark to work Together.

This means that Spark Correctness cannot be proven if execution flows into non-Spark code.

Interfacing between these requires Careful thought, and steps should be taken in Place to be able to use Spark and Ada code together in a execution of a main or a subprogram.

Spark Mode

Either code is in Spark, or not in Spark, which gives the client full use of the Ada Language. Subprogram which is in Spark cannot call a subprogram with Non-Spark features.

It is possible to call subprogram declaration even if body is not written In full Spark.

In order to interface with Spark, you have to specify if the Construct is either in spark or not by supplying the Spark Mode aspect.

For library, it is said to have a special Value of Auto.

Only be changed within the Packages or Subprograms, They cannot be included

**Proof**

**Loop Variants**

This is used to prove that loops terminate. The Tools in Spark generate a Verification Condition Using the pragma Loop\_Variant.

This pragmas has two attributes which are Increases and Decreases which can be addressed at the same time.

For example used in the Spark book.

Loop\_Variant(Increases => X, Decreases => Y – Z)

Here shows that after each iteration the Loop Variant gives us a choice of either using Increases or Decreases. If one of these are executed Spark Tools are happy to prove the Termination of the Loop.

If the pragma contains a discrete Type, they are therefore Bounded by the smallest value they can take and the Largest Value they can take.

To help the tools all that is needed is:

* that there is a known bound such as Integer’Last, Integer’First which specifies the lower and min bound.
* The Loop decreases or increases a known value within the Loop.

This enables Spark Tools to prove the Termination of a Loop something That is hard to achieve in other Languages such as Java.

The use of the Loop Variant means that there is something here that needs proving in which a verification condition is created to prove that these will never fail.

In Java, most of these Problems are found during Testing or at Run time. The Spark Tools can use the Loop Variant to then Prove that the Loop will always Terminate.

The advantage of this is Massive as using Spark to prove these very frustrating problems makes the Programmers job easier, in comparison to Java where there are no tools for proving such problems.

If it fails in either Language, you will end up with an infinite loop, stalling progress of the program. The Spark Tools use the Progress which is known by the programmer to then Prove this does terminate.

There is not such tool in Java which can be used to prove termination only thing that can be done is careful programming, and a good design to avoid such problems.