CocoaAda

Before I start programming an automatic parser for Cocoa framework headers, I want to share with you some interrogations.

Beside these questions I've made choices that I want to have feedback from potential future users.

Assuming, objective-c instance is hidden in an Ada tagged record in a package named from the class:

```
package NSObject is
  -- Class reference declaration
  NSObjectClass : constant ObjC.Class := ObjC.Runtime.Get_Class ("NSObject");
  -- Instance type declaration
  type TInst is tagged record
    Inst : ObjC.id;
  end record;
  subtype TRef is TInst'Class;
  -- Convenious declaration
  subtype id is TInst'Class;
  nil : constant TInst := (Inst => null);
  -- Instance method declaration
  function init (Self : TInst) return id;
  -- Class method declaration
  function alloc (From : ObjC.Class) return TInst;
end;
```

The following declarations will be included in "package NSString is":

Q1) hierarchical packages or flat packages:

a) type TInst is new Foundation.NSObject.TInst;

b) type TInst is new NSObject.TInst;

I've preferred form b) - hierarchical classification is done with directories, as Cocoa does

Q2) Object type named with the class name or a generic name:

a) type NSString_I is new NSObject.NSObject;
   subtype NSString_C is TInst'Class;
  function substringFromIndex (Self : NSString_I; from : NSUInteger) return NSString_C;

b) type TInst is new NSObject.TInst;
   subtype TRef is TInst'Class;
  function substringFromIndex (Self : TInst; from : NSUInteger) return TRef;

I've preferred form b) - easier readability, no repetition

Q3) Access object parameter or not:

```
        a)    type TInst is new NSObject.TInst;
                  type TRef is access all TInst'Class;
  function substringFromIndex (Self : access TInst; from : NSUInteger) return TRef;
        b)    type TInst is new NSObject.TInst;
                  subtype TRef is TInst'Class;
  function substringFromIndex (Self : TInst; from : NSUInteger) return TRef;
```

I've preferred form b) - the dynamic allocation is only done by Objective-C alloc class method

Q4) class wide parameter or not

```
        a)   function NSRunAlertPanel
    (title           : NSString.TRef;
     msgFormat       : NSString.TRef;
     defaultButton   : NSString.TRef;
     alternateButton : NSString.TRef;
     otherButton     : NSString.TRef)
     return            NSObjCRuntime.NSInteger
        b)   function NSRunAlertPanel
    (title           : NSString.TInst;
     msgFormat       : NSString.TInst;
     defaultButton   : NSString.TInst;
     alternateButton : NSString.TInst;
     otherButton     : NSString.TInst)
     return            NSObjCRuntime.NSInteger
```

I've preferred form a) - Objective-C class parameter are in principle class wide

Q5) id or actual class return parameter

```
        a)   function initWithCString
    (Self                : TInst;
     nullTerminatedCString : ObjC.STR;
     encoding              : NSStringEncoding)
     return                  id;
        b)   function initWithCString
    (Self                : TInst;
     nullTerminatedCString : ObjC.STR;
     encoding              : NSStringEncoding)
     return                  TRef;
```

I've preferred form a) - closer from Objective-C declaration but need some type casting:

```
  astr : NSString.TRef := NSString.TRef(initWithCString (astr, New_String ("Hello with %@"), 1));
```

or for each class declare id but less compliant:

```
  subtype id is TInst'Class;
```

Q6) hierarchical categories or embedded categories:

a) package NSString.NSStringExtensionMethods is

b) package NSString is
```
...
-- NSStringExtensionMethods category
...
```
I've preferred form b) - all methods are in the same file but b) let user to defined their own category
In this latter case, how to name anonymous category?

Q7) @optional in protocol

a) type TProt is interface;

b) no idea!

In Ada interface needs that all primitive will be implemented. I've no idea to translate "@optional"
Note that in Ada the primitives need to be declared again in the derived type, not in Objective-C.

Thanks in advance for your feedback, Pascal Pignard, april'2013.