Scala Namespace Management (DRAFT)

Purpose

This document describes the guidelines for managing two important namespaces: the package `scala._`, and the Maven group `org.scala-lang`.

These questions become more important as we modularize the distribution, and as we publish new modules (such as ScalaJS, scala-async, scala-pickling.)

Goals

- Consistency: modules should follow the same conventions.
- Extensibility: What happens when a single JAR module evolves into a multi-JAR module?
- Backwards Compatibility: avoid requiring early adopters of a module to modify imports as the module graduates to non-experimental status.
- Cleanliness: avoid overloading global namespaces
- Independence: names added to one module should not interact with other modules

Maven Artifact Naming

The core of the scala distribution should be published as per the status quo:

```
"org.scala-lang" % "scala-library" % "2.10.2"
```

Modules are cross versioned with an independent version number. The artifact name is prefixed with `scala-`.

```
"org.scala-lang.modules" %% "scala-parser-combinators" % "1.0"
```

Multi-JAR modules (or modules that could potentially become multi-JAR in the future) should include the module name in the Maven group ID. The artifact name is prefixed with `scala-${moduleName}-`.

```
"org.scala-lang.modules.js" %% "scala-js-compiler" % "0.1"
"org.scala-lang.modules.js" %% "scala-js-library" % "0.1"
```

The aggregator (aka parent) project for these would be called:

```
"org.scala-lang.modules.js" %% "scala-js" % "0.1"
```

These names should be reviewed and agreed upon during the SIP Process (initially, on the [scala-sips] mailing list).

Packaging

This section discusses the choice of package names.

Use of java._ and javax._ in the JDK

- The JVM security manager actively prevents third party code from residing in `java._`.
- Standard extensions were placed in `javax._` (e.g Swing)
- Discussion of java vs javax: http://stackoverflow.com/questions/727844/javax-vs-java-package

The JDK ships classes from both `java._` and `javax._` in classes.jar. Other JARs contribute to `javax`. The appendix lists the package structure.

Imports and Name binding in Scala

All Scala compilation units automatically include *root imports*:

```
import java.lang._
import scala._
import scala.Predef._
```

Scala differs from Java, which only includes `java.lang._`.

This begs the question: if we introduce `scala.foo` in a new release, will we change the meaning or break compilation of the following code?

```
// import scala._  // automatically added import
import O._
object Test { foo }
object O { def foo = () }
```

If the root imports were literally added to the source as shown, this would be a problem. But to avoid this fragility, the Scala compiler treats the root imports as though they were in an outer scope:

```
import java.lang._
```

```
{
   import scala._
   {
     import scala.Predef._
     {
        // YOUR CODE HERE
     }
   }
}
```

The rules of name binding then ensure that your imports take precendence. Similarly, if `foo` is defined in the same or an enclosing, nested package, it will bind with higher precedence than `scala.foo`.

The appendix contains the list of names in root imports that are shadowed by other root imports. Changes to that list will introduce source incompatibilities. For example, addition of `scala.reflect` in Scala 2.10.0 shadows `java.lang.reflect`.

```
% scala29 -nc  -e 'type t = reflect.Method'
% scala210 -nc  -e 'type t = reflect.Method'
error: type Method is not a member of package reflect
```

Scala package structure

The current Scala pacakge structure, and the full listing of the `scala` package is included in the appendix.

Packaging for modules:

1. Modules that used to be in the standard library will retain the existing packaging (rationale: backwards compatibility.)
2. **Modules may use the Scala namespace**, e.g. `scala.pickling` or `scala.async`. (Rationale: we considred `scalax._`, but this essentially leaves modules in a “second class” namespace forever).
3. Modules that are “incubating” (have not yet reached version 1.0 and don’t comply with the source and binary compatibility guarantees of Scala must be **in a package that is annotated with @scala.incubating**. The Scala compiler in 2.11 will emit a warning when an incubating package is referenced. Scaladoc will mark the API docs for these

modules with a warning about stability.

Once again, the choice of package names should be discussed and agreed up in the SIP mailing list. You *may* publish an early version of the package before this agreement is reached; unlike the choice of Group ID, some early fluctuations in your package names don't leave detritus in our repository, even if they do cause some rework for early adopters of the module.

Public API vs Implementation.

The Scala distribution places implementation details in be placed in sub-package(s) called `internal`(e.g. `scala.reflect.internal`.) Modules *should* follow this convention.

Rationale:

- such packages can be excluded from binary compatibility checking with MiMa. Simply making types`private[mymodule]` is not sufficient to exclude them from MiMa, as they are still bytecode-public.
- consistency helps users of our modules stick to the public API.

As always, modules should aim to present a minimal, coherent public API. This includes finalizing classes and methods unless they are designed with extension in mind, and using private access where applicable.

Open Issues

- Group ID and Package structure for SBT / Maven plugins

Appendix: Java Package Structure

```
classes.jar
    java
        applet
        awt
        beans
        io
        lang
        math
        net
        nio
```

```
        rmi
        security
        sql
        text
        util
    javax
        accessibility
        activation
        activity
        annotation
        imageio
        jws
        model
        management
        naming
        print
        rmi
        script
        security
        smartcardio
        sound
        sql
        swing
        tools
        transaction
        xml
dt.jar
    javax
        swing
jce.jar
    javax
        crypto
```

Appendix: Scala Package structure

```
scala-library.jar
    scala
        annotation
        beans
        collection
```

```
        compat
        concurrent
        io
        math
        ref
        reflect
        runtime
        sys
        text
        util

scala-reflect.jar
    scala
        reflect

scala-xml.jar
    scala
        xml

scala-parser-combinators.jar
    scala
        util
            parsing
```

Appendix: Contents of the scala._ provided by the standard library.

```
> println(ScalaPackageClass.info.decls.toList.filterNot(_.name.toString.contains('$')).mkString("\n"))
class AnyVal
object AnyVal
trait AnyValCompanion
object AnyValCompanion
class App
object App
class Array
object Array
class Boolean
object Boolean
class Byte
```

```
object Byte
class Char
object Char
trait Cloneable
object Cloneable
class Console
object Console
trait DelayedInit
object DelayedInit
class deprecated
object deprecated
class DeprecatedConsole
object DeprecatedConsole
class deprecatedInheritance
object deprecatedInheritance
class deprecatedName
object deprecatedName
class deprecatedOverriding
object deprecatedOverriding
trait DeprecatedPredef
object DeprecatedPredef
class Double
object Double
trait Dynamic
object Dynamic
class Enumeration
object Enumeration
trait Equals
object Equals
class FallbackArrayBuilding
object FallbackArrayBuilding
class Float
object Float
class Function
object Function
trait Function0
object Function0
trait Function1
object Function1
class Function10
object Function10
```

```
class Function11
object Function11
class Function12
object Function12
class Function13
object Function13
class Function14
object Function14
class Function15
object Function15
class Function16
object Function16
class Function17
object Function17
class Function18
object Function18
class Function19
object Function19
trait Function2
object Function2
class Function20
object Function20
class Function21
object Function21
class Function22
object Function22
trait Function3
object Function3
class Function4
object Function4
class Function5
object Function5
class Function6
object Function6
class Function7
object Function7
class Function8
object Function8
class Function9
object Function9
trait Immutable
```

```
object Immutable
class inline
object inline
class Int
object Int
class language
object language
class languageFeature
object languageFeature
class Long
object Long
class LowPriorityImplicits
object LowPriorityImplicits
class MatchError
object MatchError
trait Mutable
object Mutable
class native
object native
class noinline
object noinline
class None
object None
class NotImplementedError
object NotImplementedError
class NotNull
object NotNull
class Option
object Option
package package
package scala
trait PartialFunction
object PartialFunction
class Predef
object Predef
trait Product
object Product
class Product1
object Product1
class Product10
object Product10
```

```
class Product11
object Product11
class Product12
object Product12
class Product13
object Product13
class Product14
object Product14
class Product15
object Product15
class Product16
object Product16
class Product17
object Product17
class Product18
object Product18
class Product19
object Product19
trait Product2
object Product2
class Product20
object Product20
class Product21
object Product21
class Product22
object Product22
trait Product3
object Product3
class Product4
object Product4
class Product5
object Product5
class Product6
object Product6
class Product7
object Product7
class Product8
object Product8
class Product9
object Product9
trait Proxy
```

```
object Proxy
class remote
object remote
class Responder
object Responder
class ScalaReflectionException
object ScalaReflectionException
trait Serializable
object Serializable
class SerialVersionUID
object SerialVersionUID
class Short
object Short
class Some
object Some
trait Specializable
object Specializable
class specialized
object specialized
class StringContext
object StringContext
class Symbol
object Symbol
class throws
object throws
class transient
object transient
class Tuple1
object Tuple1
class Tuple10
object Tuple10
class Tuple11
object Tuple11
class Tuple12
object Tuple12
class Tuple13
object Tuple13
class Tuple14
object Tuple14
class Tuple15
object Tuple15
```

class Tuple16
object Tuple16
class Tuple17
object Tuple17
class Tuple18
object Tuple18
class Tuple19
object Tuple19
class Tuple2
object Tuple2
class Tuple20
object Tuple20
class Tuple21
object Tuple21
class Tuple22
object Tuple22
class Tuple3
object Tuple3
class Tuple4
object Tuple4
class Tuple5
object Tuple5
class Tuple6
object Tuple6
class Tuple7
object Tuple7
class Tuple8
object Tuple8
class Tuple9
object Tuple9
class unchecked
object unchecked
class UninitializedError
object UninitializedError
class UninitializedFieldError
object UninitializedFieldError
class UniquenessCache
object UniquenessCache
class Unit
object Unit
class volatile

```
object volatile
package annotation
package beans
package collection
package compat
package concurrent
package io
package math
package ref
package reflect
package runtime
package sys
package text
package util
package xml
package tools
package actors
type AnyRef
type Throwable
type Exception
type Error
type RuntimeException
type NullPointerException
type ClassCastException
type IndexOutOfBoundsException
type ArrayIndexOutOfBoundsException
type StringIndexOutOfBoundsException
type UnsupportedOperationException
type IllegalArgumentException
type NoSuchElementException
type NumberFormatException
type AbstractMethodError
type InterruptedException
value AnyRef
type TraversableOnce
type Traversable
value Traversable
type Iterable
value Iterable
type Seq
value Seq
```

type IndexedSeq
value IndexedSeq
type Iterator
value Iterator
type BufferedIterator
type List
value List
value Nil
type Stream
value Stream
type Vector
value Vector
type StringBuilder
value StringBuilder
type Range
value Range
type BigDecimal
value BigDecimal
type BigInt
value BigInt
type Equiv
value Equiv
type Fractional
value Fractional
type Integral
value Integral
type Numeric
value Numeric
type Ordered
value Ordered
type Ordering
value Ordering
type PartialOrdering
type PartiallyOrdered
type Either
value Either
type Left
value Left
type Right
value Right
class <repeated>

```
class Any
class Nothing
class <repeated...>
class <byname>
class Null
trait Singleton
```

Appendix: Root import shadowing

```
scala> def names(sym: Symbol) = sym.info.members.filter(x => x.info !=
NoType).map(_.name).toList.filterNot(_.encoded contains '$').toSet

scala> def shadow(sym1: Symbol, sym2: Symbol) = (names(sym1) intersect
names(sym2)).toList.map(_.longString).sorted.mkString("\n")
shadow: (sym1: $r.intp.global.Symbol, sym2: $r.intp.global.Symbol)String

scala> shadow(JavaLangPackageClass, ScalaPackageClass)
res59: String =
term Boolean
term Byte
term Double
term Float
term Iterable
term Long
term Short
term StringBuilder
term annotation
term ref
term reflect
type AbstractMethodError
type ArrayIndexOutOfBoundsException
type Boolean
type Byte
type ClassCastException
type Cloneable
type Double
type Error
type Exception
type Float
type IllegalArgumentException
```

```
type IndexOutOfBoundsException
type InterruptedException
type Iterable
type Long
type NullPointerException
type NumberFormatException
type RuntimeException
type Short
type StringBuilder
type StringIndexOutOfBoundsException
type Throwable
type UnsupportedOperationException

scala> shadow(JavaLangPackageClass, PredefModule.moduleClass)
res60: String =
type Class
type String

scala> shadow(ScalaPackageClass, PredefModule.moduleClass)
res61: String = ""
```