OCI | WE ARE SOFTWARE ENGINEERS.

Groovy 2.5+ Roadmap

Dr Paul King
OCI Groovy Lead
@paulk_asert

objectcomputing.com

WE ARE SOFTWARE ENGINEERS.

We deliver mission-critical software solutions that accelerate innovation within your organization and stand up to the evolving demands of your business.

- 160+ engineers
- Home of Grails & Micronaut
- Friend of Groovy
- Global Footprint

Groovy by the Numbers

- 30+ new contributors in the last 12 months
- Currently > 4 million downloads per month and growing (23m 2016, 50m 2017)
- Groovy 2.4, 2.5, 2.6, 3.0 release chains in progress:
  - 2.5.0-rc-3/2.5.0: soon!!
  - 2.5.0-rc-2: 2018-05-06
  - 3.0.0-alpha-2: 2018-04-17
  - 2.5.0-rc-1: 2018-04-09
  - 2.4.15: 2018-03-27
  - 2.6.0-alpha-3: 2018-03-06
  - 2.4.14: 2018-03-02
  - 2.5.0-beta-3: 2018-02-23

Groovy Roadmap

- **Groovy 2.5**
  - RC-2 out now, RC-3 about to start voting, GA first half of 2018
  - Macros, AST transformation improvements, various misc features
  - JDK 7 minimum, runs on JDK 9/10 with warnings
- **Groovy 2.6/3.0**
  - Alphas out now, RCs by end 2018/early 2019
  - Parrot parser, various misc features
  - JDK 8 minimum (3.0), address most JDK 9/10 issues

Groovy 2.5

- AST Transformation improvements
- Macros
- Misc improvements
- Runs on JDK 9/10
  - But with benign (but annoying) warnings

AST Transformations – Groovy 2.4, Groovy 2.5, Groovy 3.0

@ASTTest
@AutoClone
@AutoExternalize
@BaseScript
@Bindable
@Builder
@Canonical
@Category
@CompileDynamic
@CompileStatic
@ConditionalInterrupt
@Delegate
@EqualsAndHashCode
@ExternalizeMethods
@ExternalizeVerifier
@Field

@Grab
- @GrabConfig
- @GrabResolver
- @GrabExclude

@Grapes
@Immutable
@IndexedProperty
@InheritConstructors
@Lazy
Logging:
- @Commons
- @Log
- @Log4j
- @Log4j2
- @Slf4j

@ListenerList
@~~Mixin~~

@Newify
@NotYetImplemented
@PackageScope
@Singleton
@Sortable
@SourceURI
@Synchronized
@TailRecursive
@ThreadInterrupt
@TimedInterrupt
@ToString
@Trait
@TupleConstructor
@TypeChecked
@Vetoable
@WithReadLock
@WithWriteLock

@AutoFinal
@AutoImplement
@ImmutableBase
@ImmutableOptions
@MapConstructor
@NamedDelegate
@NamedParam
@NamedParams
@NamedVariant
@PropertyOptions
@VisibilityOptions

@GroovyDoc

AST Transformations – Groovy 2.4, Groovy 2.5

Numerous annotations have additional annotation attributes, e.g @TupleConstructor

```
String[] excludes() default {}
String[] includes() default {Undefined.STRING}
boolean includeProperties() default true
boolean includeFields() default false
boolean includeSuperProperties() default false
boolean includeSuperFields() default false
boolean callSuper() default false
boolean force() default false

boolean defaults() default true
boolean useSetters() default false
boolean allNames() default false
boolean allProperties() default false
String visibilityId() default Undefined.STRING
Class pre() default Undefined.CLASS
Class post() default Undefined.CLASS
```

AST Transformations – Groovy 2.5

Some existing annotations totally reworked:

@Canonical and @Immutable are now meta-annotations (annotation collectors)

AST Transforms: @Canonical becomes meta-annotation

@Canonical =>

    @ToString, @TupleConstructor, @EqualsAndHashCode

AST Transforms: @Canonical becomes meta-annotation

@Canonical =>

@ToString, @TupleConstructor, @EqualsAndHashCode

```
@AnnotationCollector(
  value=[ToString, TupleConstructor, EqualsAndHashCode],
  mode=AnnotationCollectorMode.PREFER_EXPLICIT_MERGED)
public @interface Canonical { }
```

@Canonical

```groovy
import groovy.transform.*



@Canonical(includeNames=true)
class Golfer {
    String first, last
}

def gn = new Golfer('Greg', 'Norman')
assert gn.toString() ==
        'Golfer(first:Greg, last:Norman)'
```

```java
@interface ToString {
    String[] excludes() default {};
    boolean includeNames() default false;
    ...
}

@interface EqualsAndHashcode {
    String[] excludes() default {};
    ...
}

@interface TupleConstructor {
    String[] excludes() default {};
    ...
}
```

@Canonical

```
import groovy.transform.*

@TupleConstructor
@EqualsAndHashCode
@ToString(includeNames=true)
class Golfer {
    String first, last
}

def gn = new Golfer('Greg', 'Norman')
assert gn.toString() ==
        'Golfer(first:Greg, last:Norman)'
```

```
@interface ToString {
    String[] excludes() default {};
    boolean includeNames() default false;
    ...
}

@interface EqualsAndHashcode {
    String[] excludes() default {};
    ...
}

@interface TupleConstructor {
    String[] excludes() default {};
    ...
}
```

@Canonical

```
import groovy.transform.*



@Canonical(excludes='last')
class Golfer {
    String first, last
}

def greg = new Golfer('Greg')
assert greg.toString() == 'Golfer(Greg)'
```

```
@interface ToString {
    String[] excludes() default {};
    boolean includeNames() default false;
    ...
}

@interface EqualsAndHashcode {
    String[] excludes() default {};
    ...
}

@interface TupleConstructor {
    String[] excludes() default {};
    ...
}
```

@Canonical

```
import groovy.transform.*

@ToString(excludes='last')
@EqualsAndHashcode(excludes='last')
@TupleConstructor(excludes='last')
class Golfer {
    String first, last
}

def greg = new Golfer('Greg')
assert greg.toString() == 'Golfer(Greg)'
```

```
@interface ToString {
    String[] excludes() default {};
    boolean includeNames() default false;
    ...
}

@interface EqualsAndHashcode {
    String[] excludes() default {};
    ...
}

@interface TupleConstructor {
    String[] excludes() default {};
    ...
}
```

AST Transforms: @Immutable becomes meta-annotation

@Immutable =>

```
@ToString(cache = true, includeSuperProperties = true)
@EqualsAndHashCode(cache = true)
@ImmutableBase
@ImmutableOptions
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
@TupleConstructor(defaults = false)
@MapConstructor(noArg = true, includeSuperProperties = true, includeFields = true)
@KnownImmutable
@AnnotationCollector(mode=AnnotationCollectorMode.PREFER_EXPLICIT_MERGED)
@interface Immutable { }
```

AST Transforms: @Immutable enhancements

An immutable class with one constructor making it dependency injection friendly

```
import groovy.transform.*
import groovy.transform.options.*

@ImmutableBase
@PropertyOptions(propertyHandler = ImmutablePropertyHandler)
@Canonical(defaults=false)
class Shopper {
    String first, last
    Date born
    List items
}

println new Shopper('John', 'Smith', new Date(), [])
```

AST Transforms: @Immutable enhancements

JSR-310 classes recognized as immutable

```
java.time.DayOfWeek
java.time.Duration
java.time.Instant
java.time.LocalDate
java.time.LocalDateTime
java.time.LocalTime
java.time.Month
java.time.MonthDay
java.time.OffsetDateTime
java.time.OffsetTime
java.time.Period
java.time.Year
java.time.YearMonth
java.time.ZonedDateTime
java.time.ZoneOffset
java.time.ZoneRegion
// all interfaces from java.time.chrono.*
java.time.format.DecimalStyle
java.time.format.FormatStyle
java.time.format.ResolverStyle
java.time.format.SignStyle
java.time.format.TextStyle
java.time.temporal.IsoFields
java.time.temporal.JulianFields
java.time.temporal.ValueRange
java.time.temporal.WeekFields
```

AST Transforms: @Immutable enhancements

You can write custom property handlers, e.g. to use Guava immutable collections for any collection property

```
import groovy.transform.Immutable
import paulk.transform.construction.GuavaImmutablePropertyHandler
@Immutable(propertyHandler=GuavaImmutablePropertyHandler)
class Person {
    List names = ['John', 'Smith']
    List books = ['GinA', 'ReGinA']
}

['names', 'books'].each {
    println new Person()."$it".dump()
}
//<com.google.common.collect.RegularImmutableList@90b9bd9 array=[John, Smith]>
//<com.google.common.collect.RegularImmutableList@95b86f34 array=[GinA, ReGinA]>
```

AST Transforms: @Immutable handles Optional

```
import groovy.transform.Immutable

@Immutable
class Entertainer {
    String first
    Optional<String> last
}

println new Entertainer('Sonny', Optional.of('Bono'))
println new Entertainer('Cher', Optional.empty())
```

AST Transforms: @Immutable handles Optional

```groovy
import groovy.transform.Immutable

@Immutable
class Entertainer {
    String first
    Optional<String> last
}

println new Entertainer('Sonny', Optional.of('Bono'))
println new Entertainer('Cher', Optional.empty())
```

```
Entertainer(Sonny, Optional[Bono])
Entertainer(Cher, Optional.empty)
```

AST Transforms: @Immutable handles Optional

```
import groovy.transform.Immutable

@Immutable
class Entertainer {
    String first
    Optional<String> last
}

println new Entertainer('Sonny', Optional.of('Bono'))
println new Entertainer('Cher', Optional.empty())
```

```
Entertainer(Sonny, Optional[Bono])
Entertainer(Cher, Optional.empty)
```

```
@Immutable
class Line {
    Optional<java.awt.Point> origin
}
```

@Immutable processor doesn't know how to handle field ‘origin' of type 'java.util.Optional‘ while compiling class Template…

AST Transforms: property name validation

- Transforms check property names and you can call the same methods in your custom transforms

```
import groovy.transform.ToString

@ToString(excludes = 'first')
class Cyclist {
    String firstName, lastName
}
println new Cyclist(firstName: 'Cadel', lastName: 'Evans')
```

AST Transforms: property name validation

```
import groovy.transform.ToString

@ToString(excludes = 'first')
class Cyclist {
    String firstName, lastName
}
println new Cyclist(firstName: 'Cadel', lastName: 'Evans')
```

Error during @ToString processing: 'excludes' property 'first' does not exist.

- Transforms check property names and you can call the same methods in your custom transforms

AST Transforms: @TupleConstructor defaults

```
import groovy.transform.TupleConstructor

@TupleConstructor(defaults = false)
class Flight {

    String fromCity, toCity
    Date leaving
}

@TupleConstructor(defaults = true)
class Cruise {

    String fromPort, toPort
    Date leaving
}
```

AST Transforms: @TupleConstructor defaults

```groovy
import groovy.transform.TupleConstructor

//@TupleConstructor(defaults = false)
class Flight {
    Flight(String fromCity, String toCity, Date leaving){/* ... */}
    String fromCity, toCity
    Date leaving
}

//@TupleConstructor(defaults = true)
class Cruise {
    Cruise(String fromPort=null, String toPort=null, Date leaving=null){/* ... */}
    String fromPort, toPort
    Date leaving
}
```

AST Transforms: @TupleConstructor defaults

```
import groovy.transform.TupleConstructor

//@TupleConstructor(defaults = false)
class Flight {
    Flight(String fromCity, String toCity, Date leaving){/* ... */}

    Date leaving
}

//@TupleConstructor(defaults = true)
class Cruise {
    Cruise(String fromPort=null, String toPort=null, Date leaving=null){/* ... */}


}

prin
prin
```

```
public Flight(java.lang.String,java.lang.String,java.util.Date)
```

```
public Cruise()
public Cruise(java.lang.String)
public Cruise(java.lang.String,java.lang.String)
public Cruise(java.lang.String,java.lang.String,java.util.Date)
```

AST Transforms: @TupleConstructor pre/post

```
import groovy.transform.ToString
import groovy.transform.TupleConstructor

import static groovy.test.GroovyAssert.shouldFail

@ToString
@TupleConstructor(
    pre = { first = first?.toLowerCase(); assert last },
    post = { this.last = first?.toUpperCase() }
)
class Actor {
    String first, last
}

assert new Actor('Johnny', 'Depp').toString() == 'Actor(johnny, JOHNNY)'
shouldFail(AssertionError) {
    println new Actor('Johnny')
}
```

AST Transforms: @TupleConstructor enhancements

Visibility can be specified, also works with MapConstructor and NamedVariant

```
import groovy.transform.*
import static groovy.transform.options.Visibility.PRIVATE

@TupleConstructor
@VisibilityOptions(PRIVATE)
class Person {
    String name
    static makePerson(String first, String last) {
        new Person("$first $last")
    }
}

assert 'Jane Eyre' == Person.makePerson('Jane', 'Eyre').name
def publicCons = Person.constructors
assert publicCons.size() == 0
```

AST Transforms: @MapConstructor

```groovy
import groovy.transform.MapConstructor
import groovy.transform.ToString

@ToString(includeNames = true)
@MapConstructor
class Conference {
    String name
    String city
    Date start
}

println new Conference(
    name: 'Gr8confUS', city: 'Minneapolis', start: new Date() - 2)
println Conference.constructors
```

```
Conference(name:Gr8confUS, city:Minneapolis, start:Wed Jul 26 ...)
[public Conference(java.util.Map)]
```

AST Transforms: @AutoImplement

Designed to complement Groovy’s dynamic creation of “dummy” objects

```groovy
def testEmptyIterator(Iterator it) {
    assert it.toList() == []
}

def emptyIterator = [hasNext: {false}] as Iterator

testEmptyIterator(emptyIterator)

assert emptyIterator.class.name.contains('Proxy')
```

AST Transforms: @AutoImplement

```
@AutoImplement
class MyClass extends AbstractList<String>
              implements Closeable, Iterator<String> { }
```

AST Transforms: @AutoImplement

```
class MyClass extends AbstractList<String> implements Closeable, Iterator<String> {
    String get(int param0) {
        return null
    }

    String next() {
        return null
    }

    boolean hasNext() {
        return false
    }

    void close() throws Exception {
    }

    int size() {
        return 0
    }
}
```

```
@AutoImplement
class MyClass extends AbstractList<String>
                implements Closeable, Iterator<String> { }
```

AST Transforms: @AutoImplement

```
class MyClass extends AbstractList<String> implements Closeable, Iterator<String> {
    String get(int param0) {
        return null
    }

    String next() {
        return null
    }

    boolean hasNext() {
        return false
    }

    void close() throws Exception {
    }

    int size() {
        return 0
    }
}
```

```
@AutoImplement
class MyClass extends AbstractList<String>
                 implements Closeable, Iterator<String> { }
```

```
def myClass = new MyClass()

testEmptyIterator(myClass)

assert myClass instanceof MyClass
assert Modifier.isAbstract(Iterator.getDeclaredMethod('hasNext').modifiers)
assert !Modifier.isAbstract(MyClass.getDeclaredMethod('hasNext').modifiers)
```

AST Transforms: @AutoImplement

```groovy
@AutoImplement(exception = UncheckedIOException)
class MyWriter extends Writer { }
```

```groovy
@AutoImplement(exception = UnsupportedOperationException,
        message = 'Not supported by MyIterator')
class MyIterator implements Iterator<String> { }
```

```groovy
@AutoImplement(code = { throw new UnsupportedOperationException(
        'Should never be called but was called on ' + new Date()) })
class EmptyIterator implements Iterator<String> {
    boolean hasNext() { false }
}
```

Built-in AST Transformations @AutoFinal

Automatically adds final modifier to constructor and method parameters

```
import groovy.transform.AutoFinal

@AutoFinal
class Animal {
    private String type
    private Date lastFed

    Animal(String type) {
        this.type = type.toUpperCase()
    }

    def feed(String food) {
        lastFed == new Date()
    }
}
```

```
class Zoo {
    private animals = []
    @AutoFinal
    def createZoo(String animal) {
        animals << new Animal(animal)
    }

    def feedAll(String food) {
        animals.each{ it.feed(food) }
    }
}

new Zoo()
```

Built-in AST Transformations @AutoFinal

Automatically adds final modifier to constructor and method parameters

```
import groovy.transform.AutoFinal

@AutoFinal
class Animal {
    private String type
    private Date lastFed

    Animal(final String type) {
        this.type = type.toUpperCase()
    }

    def feed(final String food) {
        lastFed == new Date()
    }
}
```

```
class Zoo {
    private animals = []
    @AutoFinal
    def createZoo(final String animal) {
        animals << new Animal(animal)
    }

    def feedAll(final String food) {
        animals.each{ it.feed(food) }
    }
}

new Zoo()
```

Built-in AST Transformations @Delegate enhancements

@Delegate can be placed on a getter rather than a field

```
class Person {
    String first, last
    @Delegate
    String getFullName() {
        "$first $last"
    }
}

def p = new Person(first: 'John', last: 'Smith')
assert p.equalsIgnoreCase('JOHN smith')
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*
import static groovy.transform.options.Visibility.*

class Color {
    private int r, g, b

    @VisibilityOptions(PUBLIC)
    @NamedVariant
    private Color(@NamedParam int r, @NamedParam int g, @NamedParam int b) {
        this.r = r
        this.g = g
        this.b = b
    }
}

def pubCons = Color.constructors
assert pubCons.size() == 1
assert pubCons[0].parameterTypes[0] == Map
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*
import static groovy.transform.options.Visibility.*

class Color {
    private int r, g, b

    @VisibilityOptions(PUBLIC)
    @NamedVariant
    private Color(@NamedParam int r, @NamedParam int g, @NamedParam int b) {
        this.r = r
        this.g = g
        this.b = b
    }
}

def pubCons = Color.constru
assert pubCons.size() == 1
assert pubCons[0].parameter
```

```
public Color(@NamedParam(value = 'r', type = int)
             @NamedParam(value = 'g', type = int)
             @NamedParam(value = 'b', type = int)
             Map __namedArgs) {
    this( __namedArgs.r, __namedArgs.g, __namedArgs.b )
    // plus some key value checking
}
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*

class Animal {
    String type, name
}

@ToString(includeNames=true)
class Color {
    Integer r, g, b
}

@NamedVariant
String foo(String s1, @NamedParam String s2,
           @NamedDelegate Color shade,
           @NamedDelegate Animal pet) {
    "$s1 $s2 ${pet.type?.toUpperCase()}:$pet.name $shade"
}

def result = foo(s2: 'S2', g: 12, b: 42, r: 12,
        type: 'Dog', name: 'Rover', 'S1')
assert result == 'S1 S2 DOG:Rover Color(r:12, g:12, b:42)'
```

@NamedVariant, @NamedParam, @NamedDelegate

```
import groovy.transform.*

class Animal {
    String type, name
}

@ToSt
class
    I
}

@Name
Strin



    "
}

def result = foo(s2: 'S2', g: 12, b: 42, r: 12,
        type: 'Dog', name: 'Rover', 'S1')
assert result == 'S1 S2 DOG:Rover Color(r:12, g:12, b:42)'
```

```
String foo(@NamedParam(value = 's2', type = String)
           @NamedParam(value = 'r', type = Integer)
           @NamedParam(value = 'g', type = Integer)
           @NamedParam(value = 'b', type = Integer)
           @NamedParam(value = 'type', type = String)
           @NamedParam(value = 'name', type = String)
           Map __namedArgs, String s1) {
    // some key validation code ...
    return this.foo(s1, __namedArgs.s2,
            ['r': __namedArgs.r, 'g': __namedArgs.g, 'b': __namedArgs.b] as Color,
            ['type': __namedArgs.type, 'name': __namedArgs.name] as Animal)
}
```

Macros

- macro method, MacroClass
- AST matcher
- Macro methods (custom macros)

Without Macros

```
import org.codehaus.groovy.ast.*
import org.codehaus.groovy.ast.stmt.*
import org.codehaus.groovy.ast.expr.*

def ast = new ReturnStatement(
    new ConstructorCallExpression(
            ClassHelper.make(Date),
            ArgumentListExpression.EMPTY_ARGUMENTS
    )
)
```

```
def ast = macro {
    return new Date()
}
```

With Macros (Groovy 2.5+)

```
import org.codehaus.groovy.ast.*
import org.codehaus.groovy.ast.stmt.*
import org.codehaus.groovy.ast.expr.*

def ast = new ReturnStatement(
    new ConstructorCallExpression(
            ClassHelper.make(Date),
            ArgumentListExpression.EMPTY_ARGUMENTS
    )
)
```

```
def ast = macro {
    return new Date()
}
```

Macros (Groovy 2.5+)

- Variations:
  - Expressions, Statements, Classes
  - Supports variable substitution, specifying compilation phase

```
def varX = new VariableExpression('x')
def varY = new VariableExpression('y')

def pythagoras = macro {
  return Math.sqrt($v{varX} ** 2 + $v{varY} ** 2).intValue()
}
```

```
def pythagoras = macro(CANONICALIZATION, true) {
    Math.sqrt($v{varX} ** 2 + $v{varY} ** 2).intValue()
}
```

Macros (Groovy 2.5+)

- Variations:
  - Expressions, Statements, Classes
  - Supports variable substitution, specifying compilation phase

```groovy
@Statistics
class Person {
    Integer age
    String name
}

def p = new Person(age: 12,
                   name: 'john')

assert p.methodCount == 0
assert p.fieldCount  == 2
```

```groovy
ClassNode buildTemplateClass(ClassNode reference) {
    def methodCount = constX(reference.methods.size())
    def fieldCount = constX(reference.fields.size())

    return new MacroClass() {
        class Statistics {
            java.lang.Integer getMethodCount() {
                return $v { methodCount }
            }

            java.lang.Integer getFieldCount() {
                return $v { fieldCount }
            }
        }
    }
}
```

AST Matching

❖ AST Matching:

- Selective transformations, filtering, testing
- Supports placeholders

```
Expression transform(Expression exp) {
    Expression ref = macro { 1 + 1 }

    if (ASTMatcher.matches(ref, exp)) {
        return macro { 2 }
    }

    return super.transform(exp)
}
```

Macro method examples: match

See: https://github.com/touchez-du-bois/akatsuki

```
def fact(num) {
    return match(num) {
        when String then fact(num.toInteger())
        when(0 | 1) then 1
        when 2 then 2
        orElse num * fact(num - 1)
    }
}

assert fact("5") == 120
```

Macro method examples: doWithData

Spock inspired

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class MathSpec extends Specification {
    def "maximum of two numbers"(int a, int b, int c) {
        expect:
        Math.max(a, b) == c

        where:
        a | b | c
        1 | 3 | 3
        7 | 4 | 7
        0 | 0 | 0
    }
}
```

Macro method examples: doWithData

See: https://github.com/touchez-du-bois/akatsuki

```
doWithData {
    dowith:
        assert a + b == c
    where:
        a | b || c
        1 | 2 || 3
        4 | 5 || 9
        7 | 8 || 15
}
```

Misc features

- Repeated annotations
- Method parameter names
- Annotations in more places (JSR-308)
- :grab in groovysh
- tap in addition to with
- CliBuilder supports annotations
- JAXB marshalling shortcuts
- Customizable JSON serializer

Repeated annotations

```java
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Repeatable(MyAnnotationArray)
@interface MyAnnotation {
    String value() default "val0"
}

@Retention(RetentionPolicy.RUNTIME)
@interface MyAnnotationArray {
    MyAnnotation[] value()
}
```

Repeated annotations

```groovy
class MyClass {
    private static final expected =
      '@MyAnnotationArray(value=[@MyAnnotation(value=val1), @MyAnnotation(value=val2)])'

    // control
    @MyAnnotationArray([@MyAnnotation("val1"), @MyAnnotation("val2")])
    String method1() { 'method1' }

    // duplicate candidate for auto collection
    @MyAnnotation(value = "val1")
    @MyAnnotation(value = "val2")
    String method2() { 'method2' }

    static void main(String... args) {
        MyClass myc = new MyClass()
        assert 'method1' == myc.method1()
        assert 'method2' == myc.method2()
        assert checkAnnos(myc, "method1") == expected
        assert checkAnnos(myc, "method2") == expected
    }

    private static String checkAnnos(MyClass myc, String name) {
        def m = myc.getClass().getMethod(name)
        List annos = m.getAnnotations()
        assert annos.size() == 1
        annos[0].toString()
    }
```

Method Parameter names

```
import org.codehaus.groovy.control.CompilerConfiguration

def fooClass = new GroovyClassLoader().parseClass('''
class Foo {
  def foo(String one, Integer two, Date three) {}
}
''')
println fooClass.methods.find{ it.name == 'foo' }.parameters


def cc = new CompilerConfiguration(parameters: true)
def barClass = new GroovyClassLoader(getClass().classLoader, cc).parseClass('''
class Bar {
  def bar(String one, Integer two, Date three) {}
}
''')
println barClass.methods.find{ it.name == 'bar' }.parameters
```

Method Parameter names

```
import org.codehaus.groovy.control.CompilerConfiguration

def fooClass = new GroovyClassLoader().parseClass('''
class Foo {
  def foo(String one, Integer two, Date three) {}
}
''')
println fooClass.methods.find{ it.name == 'foo' }.parameters


def cc = new CompilerConfiguration(parameters: true)
def barClass = new GroovyClassLoader(getClass().classLoader, cc).parseClass('''
class Bar {
  def bar(String one, Integer two, Date three) {}
}
''')
println barClass.methods.find{ it.name == 'bar' }.parameters
```

```
[java.lang.String arg0, java.lang.Integer arg1, java.util.Date arg2]
[java.lang.String one, java.lang.Integer two, java.util.Date three]
```

Work in progress…annotations in more places (jsr308)

```
class MyList extends @NonEmpty ArrayList<@NonNegative Integer> implements @NonNull Closeable {
    void close() throws IOException {}
}

@Readonly Object x
if (x instanceof Date) { /*...*/ }               // error: incompatible annotations
if (x instanceof @Readonly Date) { /*...*/ }  // OK

Object y
if (y instanceof Date) { ... }                   // OK
if (y instanceof @NonNull Date) { /*...*/ }   // error: incompatible annotations

class Folder<F extends @Existing File> { /*...*/ }
Collection<? super @Existing File> files

@Existing String @NonNull [] @NonEmpty [] @ReadOnly [] items
```

Partial support in 2.5, aiming for full support in Parrot parser

:grab in groovysh

```
Groovy Shell (3.0.0-SNAPSHOT, JVM: 1.8.0_161)
Type ':help' or ':h' for help.
---------------------------------------------------
groovy:000> :grab 'com.google.guava:guava:24.1-jre'
groovy:000> import com.google.common.collect.ImmutableBiMap
===> com.google.common.collect.ImmutableBiMap
groovy:000> m = ImmutableBiMap.of('foo', 'bar')
===> [foo:bar]
groovy:000> m.inverse()
===> [bar:foo]
groovy:000>
```

With vs Tap

```
class Person {
    String first, last, honorific
    boolean friend
}

def p = new Person(last: 'Gaga', honorific: 'Lady', friend: false)
def greeting = 'Dear ' + p.with{ friend ? first : "$honorific $last" }
assert greeting == 'Dear Lady Gaga'

new Person().tap {
    friend = true
    first = 'Bob'
}.tap {
    assert friend && first || !friend && last
}.tap {
    if (friend) {
        println "Dear $first"
    } else {
        println "Dear $honorific $last"
    }
}
```

With vs Tap

```
class Person {
    String first, last, honorific
    boolean friend
}

def p = new Person(last: 'Gaga', honorific: 'Lady', friend: false)
def greeting = 'Dear ' + p.with{ friend ? first : "$honorific $last" }
assert greeting == 'Dear Lady Gaga'

new Person().tap {
    friend = true
    first = 'Bob'
}.tap {
    assert friend && first || !friend && last
}.tap {
    if (friend) {
        println "Dear $first"
    } else {
        println "Dear $honorific $last"
    }
}
```

CliBuilder supports annotations

```
interface GreeterI {
    @Option(shortName='h', description='display usage’)
    Boolean help()
    @Option(shortName='a', description='greeting audience’)
    String audience()
    @Unparsed
    List remaining()
}
```

```
def cli = new CliBuilder(usage: 'groovy Greeter [option]')
def argz = '--audience Groovologist'.split()
def options = cli.parseFromSpec(GreeterI, argz)
assert options.audience() == 'Groovologist'
```

```
@OptionField String audience
@OptionField Boolean help
@UnparsedField List remaining
new CliBuilder().parseFromInstance(this, args)
assert audience == 'Groovologist'
```

JAXB marshalling shortcuts

```
@EqualsAndHashCode
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement
public class Person {
    String name
    int age
}
```

More concise without having to call createMarshaller() and createUnmarshaller()

```
JAXBContext jaxbContext = JAXBContext.newInstance(Person)
Person p = new Person(name: 'JT', age: 20)

String xml = jaxbContext.marshal(p)
assert jaxbContext.unmarshal(xml, Person) == p
```

A customizable JSON serializer

```groovy
def generator = new JsonGenerator.Options()
        .addConverter(URL) { URL u, String key ->
    if (key == 'favoriteUrl') {
        u.getHost()
    } else {
        u
    }
}
.build()
```

Groovy 2.6/3.0

- Parrot parser
  - 2.6 JDK 7 backport of 3.0
  - 3.0 JDK 8 requirement
- More JDK 9/10 friendly
- Misc improvements

Parrot looping

```
// classic Java-style do..while loop
def count = 5
def fact = 1
do {
    fact *= count--
} while(count > 1)
assert fact == 120
```

Parrot looping

```
// classic for loop but now with extra commas
def facts = []
def count = 5
for (int fact = 1, i = 1; i <= count; i++, fact *= i) {
    facts << fact
}
assert facts == [1, 2, 6, 24, 120]
```

Parrot looping

```
// multi-assignment
def (String x, int y) = ['foo', 42]
assert "$x $y" == 'foo 42'

// multi-assignment goes loopy
def baNums = []
for (def (String u, int v) = ['bar', 42]; v < 45; u++, v++) {
    baNums << "$u $v"
}
assert baNums == ['bar 42', 'bas 43', 'bat 44']
```

Java-style array initialization

```
def primes = new int[] {2, 3, 5, 7, 11}
assert primes.size() == 5 && primes.sum() == 28
assert primes.class.name == '[I'

def pets = new String[] {'cat', 'dog'}
assert pets.size() == 2 && pets.sum() == 'catdog'
assert pets.class.name == '[Ljava.lang.String;'

// traditional Groovy alternative still supported
String[] groovyBooks = [ 'Groovy in Action', 'Making Java Groovy' ]
assert groovyBooks.every{ it.contains('Groovy') }
```

New operators: identity

```
import groovy.transform.EqualsAndHashCode

@EqualsAndHashCode
class Creature { String type }

def cat = new Creature(type: 'cat')
def copyCat = cat
def lion = new Creature(type: 'cat')

assert cat.equals(lion) // Java logical equality
assert cat == lion      // Groovy shorthand operator

assert cat.is(copyCat)  // Groovy identity
assert cat === copyCat  // operator shorthand
assert cat !== lion     // negated operator shorthand
```

New operators: not variations

```
assert 45 !instanceof Date

assert 4 !in [1, 3, 5, 7]
```

New operators: Elvis assignment

```
import groovy.transform.ToString

@ToString
class Element {
    String name
    int atomicNumber
}
def he = new Element(name: 'Helium')
he.with {
    name = name ?: 'Hydrogen'   // existing Elvis operator
    atomicNumber ?= 2           // new Elvis assignment shorthand
}
assert he.toString() == 'Element(Helium, 2)'
```

Safe indexing

```
String[] array = ['a', 'b']
assert 'b' == array?[1]      // get using normal array index
array?[1] = 'c'              // set using normal array index
assert 'c' == array?[1]

array = null
assert null == array?[1]     // return null for all index values
array?[1] = 'c'              // quietly ignore attempt to set value
assert array == null
```

Better Java syntax support: try with resources

```
class FromResource extends ByteArrayInputStream {
    boolean closed = false

    @Override
    void close() throws IOException {
        super.close()
        closed = true
    }

    FromResource(String input) {
        super(input.bytes)
    }
}

class ToResource extends ByteArrayOutputStream {
    boolean closed = false

    @Override
    void close() throws IOException {
        super.close()
        closed = true
    }
}
```

Better Java syntax support: try with resources

```
// Java style
try(
    FromResource from = new FromResource("ARM was here!")
    ToResource to = new ToResource()
) {
    to << from
} finally {
    assert from.closed
    assert to.closed
    assert to.toString() == 'ARM was here!'
}
```

Better Java syntax support: try with resources

```groovy
// some Groovy friendliness without explicit types
try(
    from = new FromResource("ARM was here!")
    to = new ToResource()
) {
    to << from
} finally {
    assert from.closed
    assert to.closed
    assert to.toString() == 'ARM was here!'
}
```

But remember Groovy’s IO/Resource extension methods may be better

Better Java syntax support: nested blocks

```
{
    def a = 1
    a++
    assert 2 == a
}
try {
    a++ // not defined at this point
} catch(MissingPropertyException ex) {
    println ex.message
}
{
    {
        // inner nesting is another scope
        def a = 'banana'
        assert a.size() == 6
    }
    def a = 1
    assert a == 1
}
```

Lambdas

```
import static java.util.stream.Collectors.toList

(1..10).forEach(e -> { println e })

assert (1..10).stream()
        .filter(e -> e % 2 == 0)
        .map(e -> e * 2)
        .collect(toList()) == [4, 8, 12, 16, 20]
```

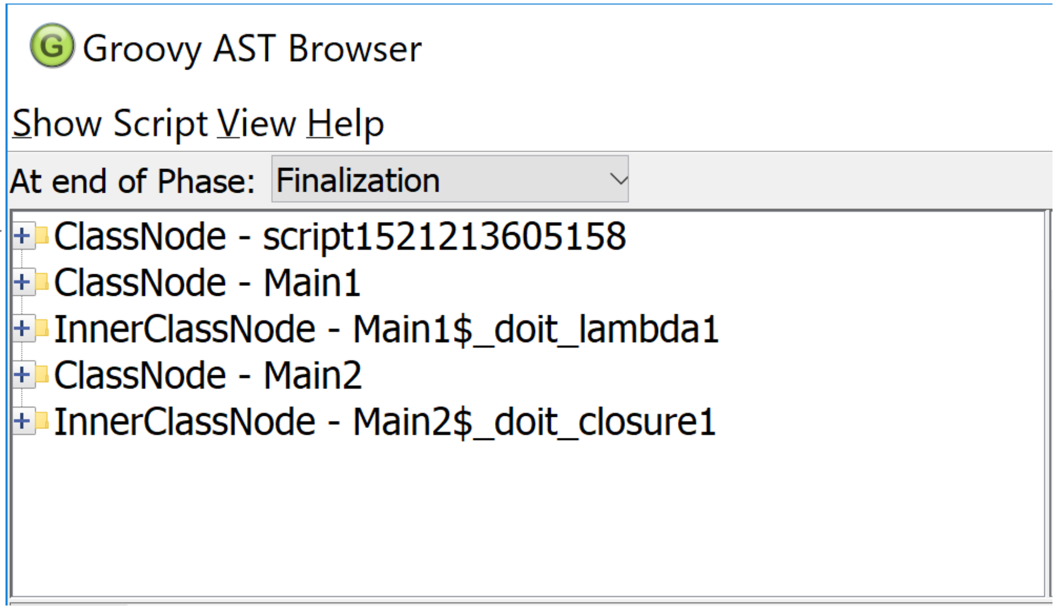
Lambdas – native lambdas with @CompileStatic

```
import groovy.transform.CompileStatic
import static java.util.stream.Collectors.toList
import java.util.stream.Stream

class Main1 {
    @CompileStatic
    static void doit() {
        assert [2] == Stream.of(1).map( e -> e.plus 1 ).collect(toList())
    }
}

class Main2 {
    static void doit() {
        assert [2] == Stream.of(1).map( e -> e.plus 1 ).collect(toList())
    }
}

Main1.doit()
Main2.doit()
```



At end of Phase: Finalization

```
+ ClassNode - script1521213605158
+ ClassNode - Main1
+ InnerClassNode - Main1$_doit_lambda1
+ ClassNode - Main2
+ InnerClassNode - Main2$_doit_closure1
```

Lambdas – all the shapes

```
// general form
def add = (int x, int y) -> { def z = y; return x + z }
assert add(3, 4) == 7

// curly braces are optional for a single expression
def sub = (int x, int y) -> x - y
assert sub(4, 3) == 1

// parameter types and
// explicit return are optional
def mult = (x, y) -> { x * y }
assert mult(3, 4) == 12
```

```
// no parentheses required for a single parameter with no type
def isEven = n -> n % 2 == 0
assert isEven(6)
assert !isEven(7)

// no arguments case
def theAnswer = () -> 42
assert theAnswer() == 42

// any statement requires braces
def checkMath = () -> { assert 1 + 1 == 2 }
checkMath()

// example showing default parameter values (no Java equivalent)
def addWithDefault = (int x, int y = 100) -> x + y
assert addWithDefault(1, 200) == 201
assert addWithDefault(1) == 101
```

Method references: classes

```
import java.util.stream.Stream
import static java.util.stream.Collectors.toList

// class::staticMethod
assert ['1', '2', '3'] ==
        Stream.of(1, 2, 3)
                .map(String::valueOf)
                .collect(toList())

// class::instanceMethod
assert ['A', 'B', 'C'] ==
        ['a', 'b', 'c'].stream()
                .map(String::toUpperCase)
                .collect(toList())
```

Method references: instances

```
// instance::instanceMethod
def sizeAlphabet =
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'::length
assert sizeAlphabet() == 26

// instance::staticMethod
def hexer = 42::toHexString
assert hexer(127) == '7f'
```

Method references: constructors

```
// normal constructor
def r = Random::new
assert r().nextInt(10) in 0..9

// array constructor is handy when working with various Java libraries, e.g. streams
assert [1, 2, 3].stream().toArray().class.name == '[Ljava.lang.Object;'
assert [1, 2, 3].stream().toArray(Integer[]::new).class.name == '[Ljava.lang.Integer;'

// works with multi-dimensional arrays too
def make2d = String[][]::new
def tictac = make2d(3, 3)
tictac[0] = ['X', 'O', 'X']
tictac[1] = ['X', 'X', 'O']
tictac[2] = ['O', 'X', 'O']
assert tictac*.join().join('\n') == '''
XOX
XXO
OXO
'''.trim()
```

Method references: constructors

```
// also useful for your own classes
import groovy.transform.Canonical
import java.util.stream.Collectors

@Canonical
class Animal {
    String kind
}

def a = Animal::new
assert a('lion').kind == 'lion'

def c = Animal
assert c::new('cat').kind == 'cat'

def pets = ['cat', 'dog'].stream().map(Animal::new)
def names = pets.map(Animal::toString).collect(Collectors.joining( "," ))
assert names == 'Animal(cat),Animal(dog)'
```

Default methods in interfaces

```
interface Greetable {
    String target()

    default String salutation() {
        'Greetings'
    }

    default String greet() {
        "${salutation()}, ${target()}"
    }
}

class Greetee implements Greetable {
    String name
    @Override
    String target() { name }
}

def daniel = new Greetee(name: 'Daniel')
assert 'Greetings, Daniel' == "${daniel.salutation()}, ${daniel.target()}"
assert 'Greetings, Daniel' == daniel.greet()
```

Currently implemented using traits

GroovyDoc comments as metadata

```
import org.codehaus.groovy.control.*

import static groovy.lang.groovydoc.GroovydocHolder.DOC_COMMENT

def ast = new CompilationUnit().tap {
    addSource 'myScript.groovy', '''
        /** class doco */
        class MyClass {
            /** method doco */
            def myMethod() {}
        }
    '''
    compile Phases.SEMANTIC_ANALYSIS
}.ast
def getDoc(node) { node.nodeMetaData[DOC_COMMENT].content }
def myClass = ast.classes[0]
def myClassDoc = getDoc(myClass)
def myMethodDoc = getDoc(myClass.methods[0])
assert myClassDoc.contains('class doco')
assert myMethodDoc.contains('method doco')
```

Groovydoc comments: runtime embedding

```
class Foo {
    /** @Groovydoc fo fum */
    def bar() { }
    @Groovydoc('Hard-coded')
    def baz() { }
}

def docForMethod(String name) {
//    Foo.methods.find{ it.name == name }.getAnnotation(Groovydoc).value()
    Foo.methods.find{ it.name == name }.groovydoc.content
}
assert docForMethod('bar').contains('@Groovydoc fo fum')
assert docForMethod('baz').contains('Hard-coded')
```