# GEP 8 - Static type checking

## Metadata

| | |
|---|---|
| Number: | GEP-8 |
| Title: | Static Type Checking |
| Version: | 8 |
| Type: | Feature |
| Status: | Draft |
| Leader: | Cédric Champeau |
| Created: | 2011-10-08 |
| Last modification: | 2012-02-21 |

## Abstract: Static Type Checking

This GEP introduces a new feature in the language known as *static type checking*. It is often disturbing for developers coming from a statically typed language (say Java) to discover that the Groovy compiler will not complain at compile time:

- when assignments are made on different types
- when a method doesn't exist
- when a property or variable doesn't exist
- when returned object type doesn't match the method signature
- ...

All those are silent because the dynamic nature of the Groovy language makes such code perfectly valid. However, in some situations, a developper may want Groovy to behave like a statically typed language and have the compiler give hints about such "errors". To do this, Groovy must introduce static type checking.

## Rationale: Static Type Checking vs Static compilation

It is important to make the difference between static type checking and static compilation. The goal of this GEP is to have an *option* to turn static type checking (STC) on. If STC is activated, the compiler will be more verbose (you will also see the term *"grumpy"*), but in the end, the generated bytecode and runtime behaviour will be exactly the same as if you did not activate this mode. This is a major difference from an alternate compiler like Groovy++ which will perform STC then produce a different bytecode and therefore produce **different runtime semantics**. The scope of this GEP is only a *static type checker*, and therefore should only be considered as a feature which allows developers to write statically checked code, so is an elegant way for example to leverage the Groovy syntax to reduce verbosity of Java code while still getting strongly checked code. Eventually, IDE could support the STC mode and provide information to the developper.

# Implementation details

## Development branch

Since Groovy 2.0-beta-2, code has been merged into *master* branch. However, if heavy developments are done on the type checker, it is advisable to work on the *grumpy* branch. It adds an AST transformation named *TypeChecked*. If set, then the AST transformation will perform type inference and store type information in AST nodes metadata. Eventually, if errors are found, it will add errors to the compiler through a dedicated *addStaticTypeError* method which basically does the same as the traditional *addError* method but prefixes the messages with a "Static type checking" message. This is done to help the developer determine whether the error he is seeing is a "plain Groovy" error, or an error thrown by the STC mode.

## The StaticTypeCheckingTestCase class

Static type checking behaviour must be tested. As there are tons of possible checks to be done, a base test class provides a framework for testing this mode. Unit tests for static type checking should override this class.

# Decisions made

## About this section

The goal of this section is to provide code samples which demonstrates in what case the STC transformation will actually complain and what is the expected error message, and serves as a basis to future STC documentation. This section may not be up-to-date, and one should always take a look at the STC unit tests found in the *src/test/groovy/transform/stc* directory.

## Features

| Feature | Example | Behaviour | Status |
|---|---|---|---|
| Method does not exist | `def method() { ... } methode() // typo` | Complains about undefined method | Implemented |
| Property does not exist | `class A { int x } A obj = new A() a.y = 2` | Complains about undefined property "y" | Implemented |

| | | | |
|---|---|---|---|
| Assignment type checking | `int x=2`<br>`x = 'String'` | Assigning a String to an int is forbidden | Implemented |
| Incompatible binary expressions | `1+'string'` | Checks that arguments of a binary expression are compatible (here, no 'plus' method is available | Implemented |
| Possible loss of precision (1/2) | `long myLong = ...`<br>`int myInt = myLong` | Complains about possible loss of precision | Implemented |
| Possible loss of precision (2/2) | `int myInt = 2L` | Will not complain because '2' can be represented as an int | Implemented |
| Arrays components | `String[] arr = { '1','2','3' }`<br>`arr[2] = 200` | Cannot assign an int value in an array of type String[] | Implemented |
| Method return type check | `String method() { 'Hello' }`<br>`int x = method()`<br>`// return types don't match` | Ensures that assignments are compatible with method return type | Implemented |

| | | | |
|---|---|---|---|
| Explicit return type checking | `int method() { return 'String' // return type is not compatible }` | Ensures that returned value is compatible with declared return type | Implemented |
| Implicit return type checking | `int method() { 'String' // return type is not compatible }` | Ensures that returned value is compatible with declared return type | Implemented |
| Implicit toString() | `String method(String name) { StringBuilder sb = new StringBuilder() sb 'Hi' << name << '!' }` | Implicit call to toString() | Implemented |

| | | | |
|---|---|---|---|
| Basic type inference | `def str = 'My string' str.toUpperCase() // type of 'str' is inferred` | Method calls as well as property access are checked against inferred type | Implemented |
| Basic flow analysis | `def o = new Object() o = 1 o = 'Hello' o.toUpperCase()` | Last method call will not complain because type of 'o' at this point can be inferred | Implemented |
| Instance of | `def o ... if (o instanceof String) { o.toUpperCase() // no explicit cast required }` | Casts should not be necessary when type can be inferred from a previous instanceof check | Implemented |
| DefaultGroovyMethods support | `'123'.toInteger() // toInteger() is a Groovy extension method` | Method calls can be resolved against Groovy extension methods | Implemented |

| | | | |
|---|---|---|---|
| with | `class A {`<br>`  int x`<br>`}`<br>`def a =`<br>`new A()`<br>`a.with {`<br>`  x = 1`<br>`}` | Static type checking should be aware of the "with" structure | Implemented |
| Categories | `use`<br>`(MyString`<br>`Category)`<br>`{`<br><br>`'string'.`<br>`methodInS`<br>`tringCate`<br>`gory()`<br>`}` | Compiler should be aware that extension method is found in a category | **N/A (support will be limited as category support is inherently dynamic)** |
| Groovy list constructor | `Dimension`<br>`d =`<br>`[100,200]` | Type checks the arguments and the number of arguments | Implemented |
| Groovy map constructor | `Bean`<br>`myBean =`<br>`[x:100,y:`<br>`200]` | Type checks the properties and checks for wrong property names | Implemented |

| | | | |
|---|---|---|---|
| Closure parameter types | `def closure = { int x, int y -> x+y } closure(1,2) closure('1','2') // complains` | Type checking the arguments when calling a closure | Implemented |
| Closure return type inference | `def closure = { int x, int y -> x+y } int sum = closure(1,2)` | Closure return type can be inferred from block | Implemented |
| Method return type inference | `def method(int x, int y) { x+y } int sum = method(1, 2)` | Return type can be inferred from a method if the method is itself annotated with @TypeChecked (or class is annotated with @TypeChecked) | Implemented |
| Multiple assignments | `def (x,y) = [1,2]` | In case of inline declaration, type check arguments. | Implemented |
| Multiple assignments from a variable | `def (x,y) = list` | In case of inline declaration, type check arguments. | Unsupported |

| | | | |
|---|---|---|---|
| Generics | `List<String> list = []`<br>`List<String> list = ['a','b','c']`<br>`List<String> list = [1,2,3] // should throw error` | Type checking of generic parameters | Implemented |
| Spread operator | `def list = ['a','b','c']`<br>`list*.toUpperCase()` | Type checking against component type | Implemented |

| | | | |
|---|---|---|---|
| Closure shared variables | **Closure shared variables**<br>`def x = new Date()`<br>`def cl = { x = 'hello' }`<br>`cl()`<br>`x.toUpperCase() // should throw an error because the toUpperCase() method doesn't belong to both Date and String classes` | Type check assignments of closure shared variables. The type checker is required to perform a two-pass verification, in order to check that method calls on a closure shared variables belong to the lowest upper bound of all assignment types. | Implemented |

# Open discussions

## Closure parameter type inference

With the current version of the checker, idiomatic constructs like :

```
['a','b','c'].collect { it.toUpperCase() }
```

Are not properly recognized. You have to explicitly set the type of the "it" parameter inside the closure. It is because the expected parameter types of closures are unknown at compile time. There is a discussion about how to add this type information to source code so that the inference engine can deal with them properly. The implementation of closure parameter type inference requires a change to the method signatures. It will probably not belong to the initial release of the type checker.

## Unification Types

In cases of for example "x instanceof A || x instanceof B" with A and B being unrelated we could still make an

artificial union kind of type, that contains everthing present in A and B, to allow those kinds of method calls. The alternative to this is to allow only methods from Object here, which is less interesintg. This typing can also be used for multicatch, ensuring that a method call is only valid if it exists on each of the exceptions for the multicatch. In the current implementation (Okt-14-2011) the multicatch is already expanded at the point @TypeChecked will check. Meaning effectively this already represents a kind of union type, as the same code is in each catch block and thus the method call would fail, if the method is not available on each type. The proposed behaviour is therefore to align the instanceof case with multicatch.

# References

## Mailing-list discussions

- [groovy-dev] Static mode : An early discussion about the (anticipated) announcement of the "Grumpy" mode
- [groovy-user] What to do on assignment? : Discussion about the expected behaviour when STC detects a potential error on assignment (for example, possible loose of precision on implicit number casts)

## JIRA issues

- GROOVY-5073: GEP-8 - Static type checking
- GROOVY-3014: add an annotation that forces the compiler to check methods for their existence at compile time
- GROOVY-5081: Handle explicit and implicit returns

## Useful links

- Blackdrag's blog post about flow typing
- Cedric Champeau's blog post about Grumpy update