Tastydoc

A documentation tool for Dotty using TASTy files

Bryan Abate

19th June 2019

Table of Contents

Table of Contents

What is Tastydoc ?

What is Tastydoc ?

- Documentation tool for Dotty

What is Tastydoc ?

- Documentation tool for Dotty
- Uses TASTy files

What is Tastydoc ?

- Documentation tool for Dotty
- Uses TASTy files
- Outputs Markdown

Table of Contents

Accessible information

Accessible information

- Annotations, modifiers (including scope modifiers), parameters, type parameters, and return types

Accessible information

- Annotations, modifiers (including scope modifiers), parameters, type parameters, and return types
- Members, parents, constructors, known subclasses and companion

Accessible information

- Annotations, modifiers (including scope modifiers), parameters, type parameters, and return types
- Members, parents, constructors, known subclasses and companion
- User documentation (Wiki-style & Markdown)

TASTy

TASTy

- Extract information from them

TASTy

- Extract information from them
- Independent from the compiler

Linking

Linking

- To types

Linking

- To types
- Companion

Linking

- To types
- Companion
- Annotations

Linking

- To types
- Companion
- Annotations
- Scope modifiers

Linking

- To types
- Companion
- Annotations
- Scope modifiers
- Parents

Markdown

Markdown

- Easy to edit by hand & preview

Markdown

- Easy to edit by hand & preview
- Easy to add own files

Markdown

- Easy to edit by hand & preview
- Easy to add own files
- Easy for the user to make links

Markdown

- Easy to edit by hand & preview
- Easy to add own files
- Easy for the user to make links
- Git hosting services have built-in preview

Markdown

- Easy to edit by hand & preview
- Easy to add own files
- Easy for the user to make links
- Git hosting services have built-in preview
- Easy to convert to another format (HTML, PDF, etc.)

Table of Contents

Example: scala.tasty.Reflection

scala.tasty

class Reflection

Companion object Reflection

```
class Reflection extends Core with ConstantOps with ContextOps with CommentOps with FlagsOps with IdOps
```

Constructors:

```
Reflection(kernel: Kernel)
```

Concrete Type Members:

typing

```
final object typing
```

Concrete Value Members:

error

```
def error(msg: => String, source: SourceFile, start: Int, end: Int)(implicit ctx: Context): Unit
```

error

```
def error(msg: => String, pos: Position)(implicit ctx: Context): Unit
```

rootContext

```
implicit def rootContext: Context
```

Context of the macro expansion

Example: dotty.DottyPredef

object DottyPredef

```
final object DottyPredef extends Serializable
```

Concrete Value Members:

assert

```
@forceInline final inline def assert(assertion: => Boolean): Unit
```

assert

```
@forceInline final inline def assert(assertion: => Boolean, message: => Any): Unit
```

assertFail

```
def assertFail(message: => Any): Unit
```

assertFail

```
def assertFail(): Unit
```

implicitly

```
@forceInline final inline def implicitly[T](implicit ev: T): T
```

locally

```
@forceInline inline def locally[T](body: => T): T
```

the

```
inline def the[T](x: T): x
```

Example: scalaShadowing.language

scalaShadowing

object language

```
final object language extends Serializable
```

The `scala.language` object controls the language features available to the programmer, as proposed in the **SIP-18 document**. Each of these features has to be explicitly imported into the current scope to become available:

```
import language.postfixOps // or language._
List(1, 2, 3) reverse
```

The language features are:

- `dynamics` enables defining calls rewriting using the `Dynamic` trait
- `postfixOps` enables postfix operators
- `reflectiveCalls` enables using structural types
- `implicitConversions` enables defining implicit methods and members
- `higherKinds` enables writing higher-kinded types
- `existentials` enables writing existential types
- `experimental` contains newer features that have not yet been tested in production

and, for dotty:

- `Scala2`] backwards compatibility mode for Scala2
- `noAutoTupling` disable auto-tupling
- `strictEquality` enable strick equality

production Language Features

experimental Experimental Language Features

experimental 10 Dotty-specific features come at the end. Note: Due to the more restricted language import mechanism in dotty (only imports count, implicits are disregarded) we don't need the constructions of the inherited language features. A simple object for each feature is sufficient.

Table of Contents

Representation

Representation

- Contain information about an entity

Representation

- Contain information about an entity
- Easy to use, no knowledge of TASTy required

Representation

- Contain information about an entity
- Easy to use, no knowledge of TASTy required
- Code is easy to maintain

Representation

- Contain information about an entity
- Easy to use, no knowledge of TASTy required
- Code is easy to maintain
- Similar to Dottydoc Entity $\rightarrow$can reuse Dottydoc code

Reference

Reference

- Contain information about types

Reference

- Contain information about types
- Necessary for linking

Reference

- Contain information about types
- Necessary for linking
- Inspired by Dottydoc

User documentation

User documentation

- Access to all @ except `@usecase` and `@define`

User documentation

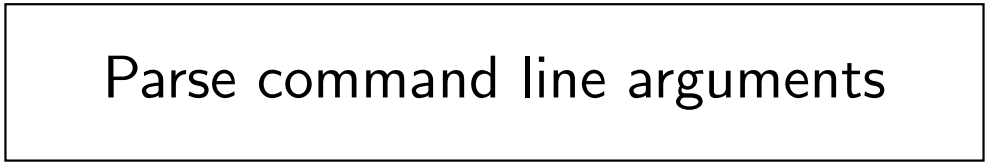
- Access to all `@` except `@usecase` and `@define`
- Support Wiki-style and Markdown

User documentation

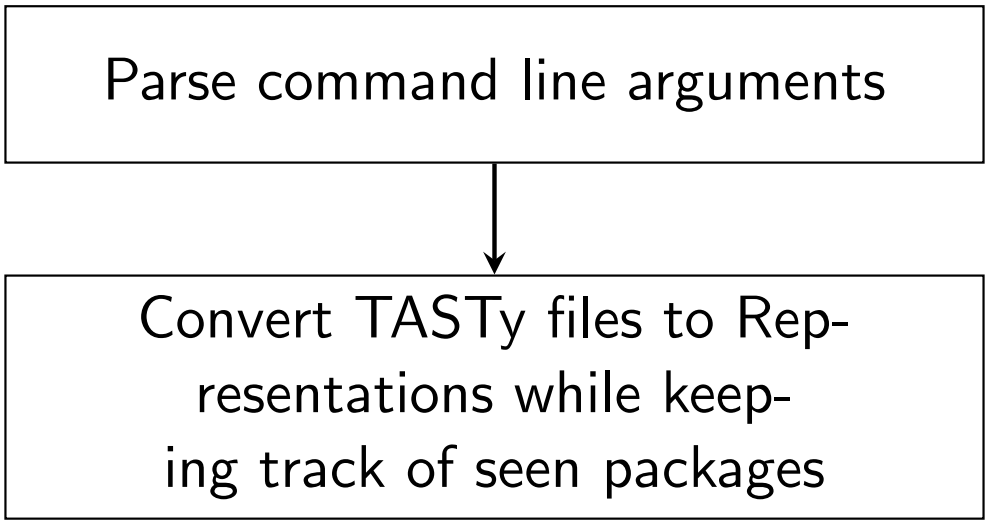
- Access to all @ except `@usecase` and `@define`
- Support Wiki-style and Markdown
- Uses Dottydoc code modified for:
  - Markdown output
  - Small changes in structure

Workflow

Workflow



Workflow


Parse command line arguments
Convert TASTy files to Representations while keeping track of seen packages

Workflow

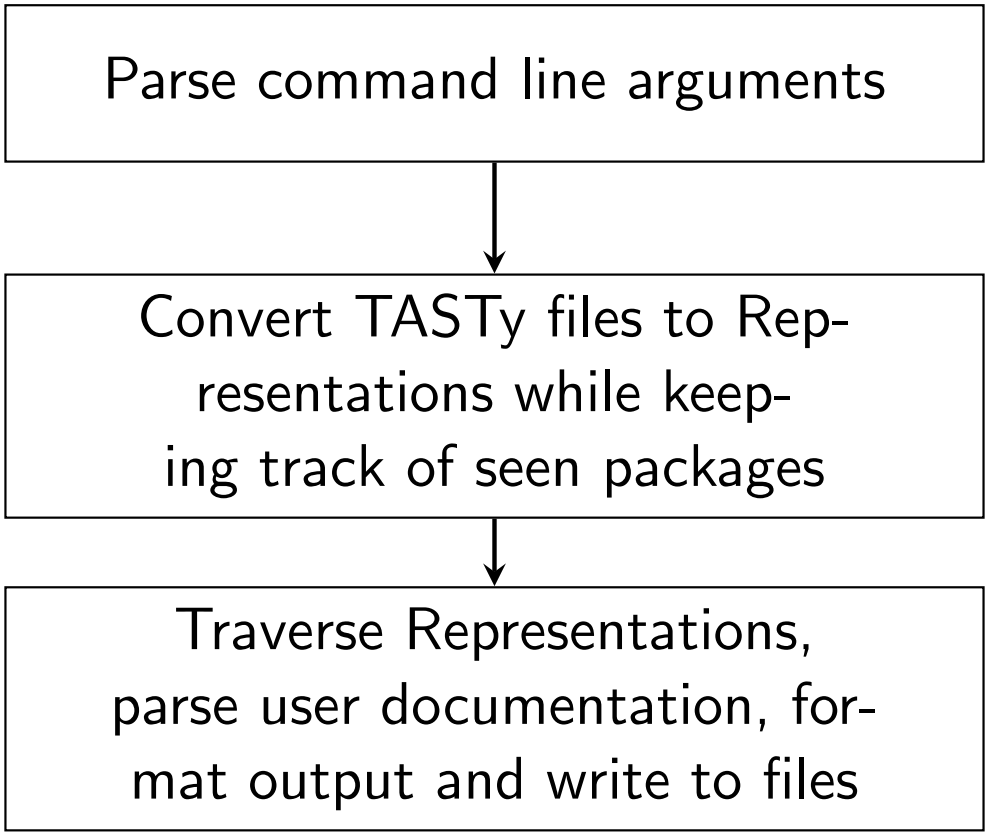

Parse command line arguments
Convert TASTy files to Representations while keeping track of seen packages
Traverse Representations, parse user documentation, format output and write to files

Table of Contents

General comparison

General comparison

- Compiler internals

General comparison

- Compiler internals
- Markdown vs HTML/CSS

Extra features

Extra features

- Scope modifiers

Extra features

- Scope modifiers
- Known subclasses

Extra features

- Scope modifiers
- Known subclasses
- Refined types

Bugs fixed

Bugs fixed

- Buggy output

```
final val BITS_PER_LAZY_VAL : [31m2L[0m
```

Bugs fixed

- Buggy output

```
final val BITS_PER_LAZY_VAL : [31m2L[0m
```

- Wrong parents

Bugs fixed

- Buggy output

```
final val BITS_PER_LAZY_VAL : [31m2L[0m
```

- Wrong parents
- Annotations

Bugs fixed

- Buggy output

```
final val BITS_PER_LAZY_VAL : [31m2L[0m
```

- Wrong parents
- Annotations
- Compiler artifacts

Bugs fixed

- Buggy output

```
final val BITS_PER_LAZY_VAL : [31m2L[0m
```

- Wrong parents
- Annotations
- Compiler artifacts
- potentially program breaking code

```
def parents: List[Entity] = this :: this.parents
```

Table of Contents

Problems

Problems

- Markdown escaping

Problems

- Markdown escaping
- Linking inside code blocks

Problems

- Markdown escaping
- Linking inside code blocks
- Sections

Problems

- Markdown escaping
- Linking inside code blocks
- Sections
- IDs for linking

Further work

Further work

- Markdown escaping

Further work

- Markdown escaping
- Type lambdas

Further work

- Markdown escaping
- Type lambdas
- Complex types

```
class Graph {
    type Node = Int
}
def linkingGraph(g: Graph): g.Node = ???
```

Further work

- Markdown escaping
- Type lambdas
- Complex types

```
class Graph {
    type Node = Int
}
def linkingGraph(g: Graph): g.Node = ???
```

- Default values

Further work

- Markdown escaping
- Type lambdas
- Complex types

```
class Graph {
    type Node = Int
}
def linkingGraph(g: Graph): g.Node = ???
```

- Default values
- Extra user-documentation parsing

Further work

- Markdown escaping
- Type lambdas
- Complex types

```
class Graph {
    type Node = Int
}
def linkingGraph(g: Graph): g.Node = ???
```

- Default values
- Extra user-documentation parsing
- HTML/CSS

Questions ?

?