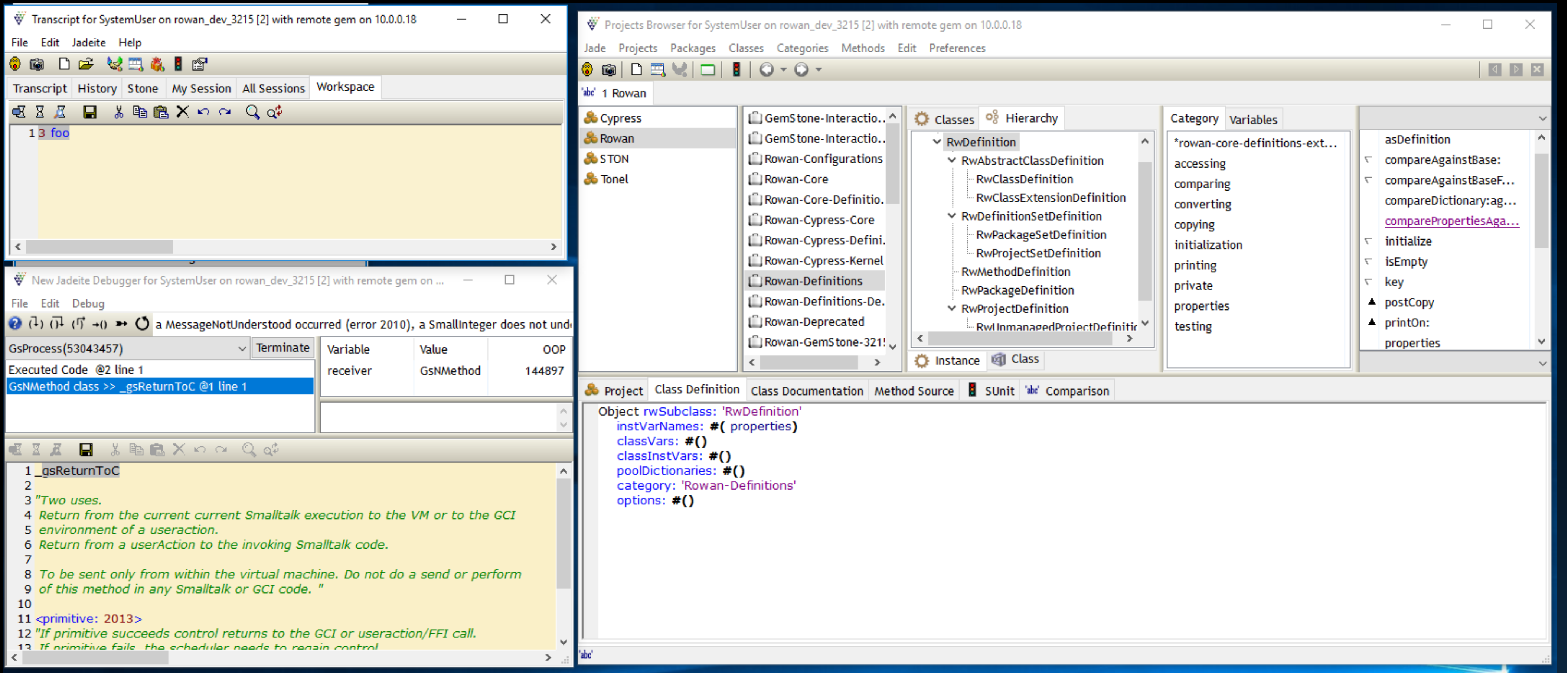
Rowan: A new project/package manager

Jadeite: A Development client for Rowan and GemStone/S

Dale Henrichs
GemTalk Systems
ESUG 2018

Jadeite



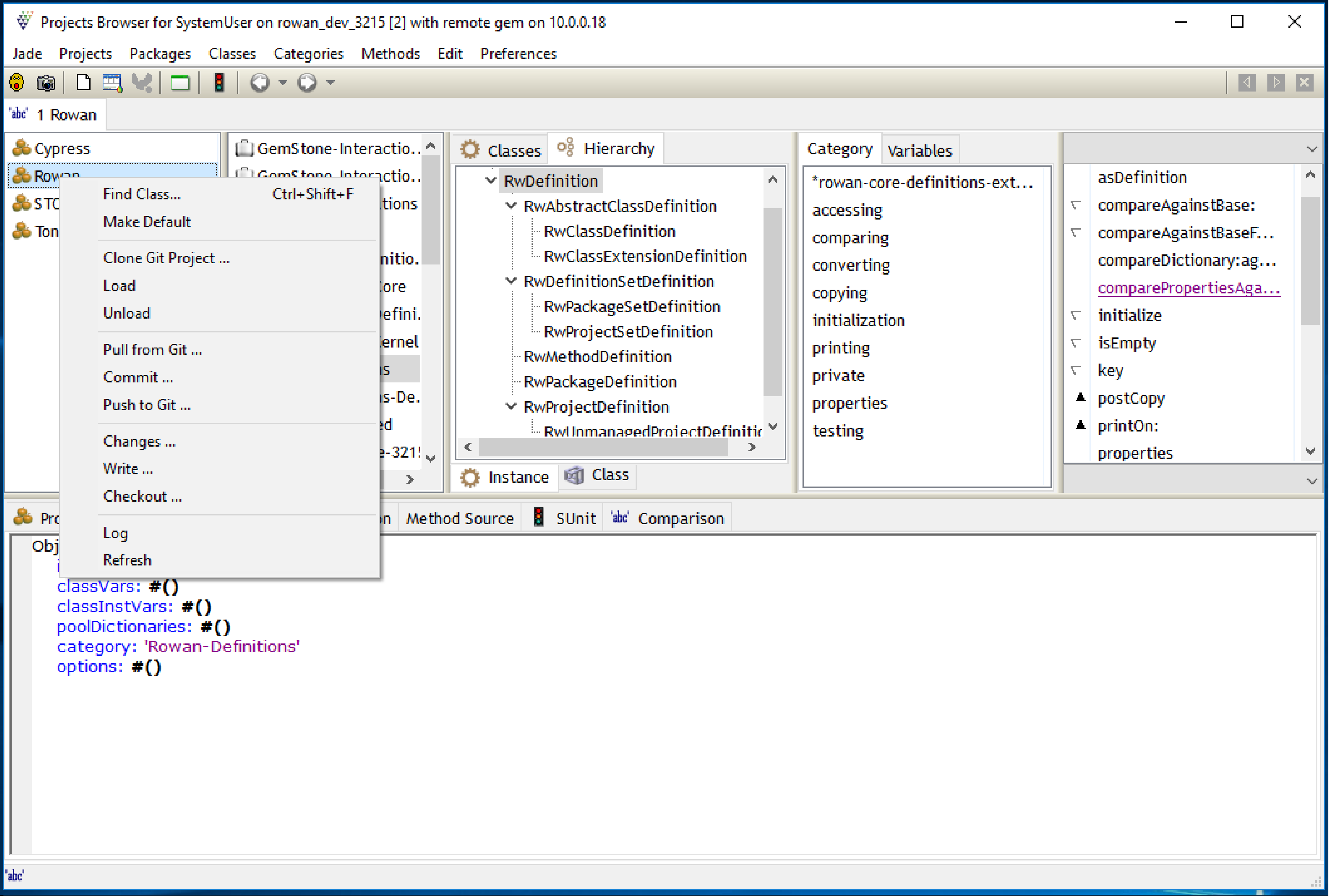
```
3 foo
```

a MessageNotUnderstood occurred (error 2010), a SmallInteger does not und

Variable	Value	OOP
receiver	GsNMethod	144897

```
_gsReturnToC

"Two uses.
 Return from the current current Smalltalk execution to the VM or to the GCI
 environment of a useraction.
 Return from a userAction to the invoking Smalltalk code.

 To be sent only from within the virtual machine. Do not do a send or perform
 of this method in any Smalltalk or GCI code. "

<primitive: 2013>
"If primitive succeeds control returns to the GCI or useraction/FFI call.
 If primitive fails, the scheduler needs to regain control
```

```
Object rwSubclass: 'RwDefinition'
   instVarNames: #( properties)
   classVars: #()
   classInstVars: #()
   poolDictionaries: #()
   category: 'Rowan-Definitions'
   options: #()
```



```
Obj
  i
  classVars: #()
  classInstVars: #()
  poolDictionaries: #()
  category: 'Rowan-Definitions'
  options: #()
```

Current Rowan Status

- In production at a GemTalk customer site
- Running on Gemstone/S 3.2.15
- Still not quite feature complete

Current Jadeite Status

- Customer is currently using an early Alpha Jadeite for development
- Plan to release to customer a new Jadeite Alpha
- Jadeite runs on Windows, but can be used on a Mac with Wine installed.

Rowan

Rowan is an updated implementation of Monticello and a wholesale replacement for Metacello.

Rowan Structure

Project
- Configurations
  - Configurations
  - External Project References
  - Packages
    - Classes
      - Methods
    - Class Extensions
      - Methods

Fine print: External project references not yet implemented

Rowan project (in image)

- Named
- Associated with a set of directories on disk
- SCM neutral, but Git aware
- Loaded using a `project load specification`

Rowan project load specification (1 of 3)

- Named
- Specifies disk structure:
  - src directory for packages
  - configs directory for configurations
  - specs directory for load specs
- Specifies load parameters
  - List of configuration names and group names
  - Remote repository specifics

Rowan project load specification (2 of 3)

- Instance-based (declarative)
  - Stored as STON on disk
  - Potential for multiple implementations
    - depending upon needs of project
    - Support future expansion
- Located anywhere
  - Meant to be copied and customized
  - Typically reference via an url (`file:` or `https:` or ???)

Rowan project load specification (3 of 3)

```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’, ‘tests’, ‘examples’ ],
    #comment : 'Sample project load specification'
}
```

Rowan configurations (1 of 3)

- Named
- The minimum loadable unit for Rowan
- Located in the `configs` directory
- Declares the set of packages to be loaded
  - Expected to define loadable subsystems
  - Should include ‘core’ and ‘tests’

Rowan configurations (2 of 3)

- Specifies nested configurations and external project references
- May be recursive
- Instance-based (declarative)
  - Stored as STON on disk
  - Potential for multiple implementations
    - depending upon needs of project
    - Support future expansion

Fine print: RwProjectReferenceConfiguration not yet implemented

Rowan configurations (3 of 3)

```
RwProjectCompoundConfiguration{
    #name : 'Core',
    #packageNames : [
        'RowanSample1-Core',
        'RowanSample1-Extensions',
        'RowanSample1-Tests'
    ],
    #comment : 'Project packages'
}
```

Rowan External Project References (1 of 2)

- Named
- References a `project load specification` url
- Optionally specifies overrides for referenced `project load specification`
- A “kind of” configuration
  - Resolves to project(s) and packages added to the load list

Fine print: RwProjectReferenceConfiguration not yet implemented

Rowan External Project References (2 of 2)

```
RwProjectReferenceConfiguration{
    #name : 'FileSystem',
    #projectSpecUrl : 'https://raw.githubusercontent.com/GemTalk/FileSystem/master/rowan/specs/FileSystem.ston',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames : [‘core’, ‘examples’ ],
    #comment : 'Reference to the FileSystem project' }
```

Fine print: RwProjectReferenceConfiguration not yet implemented

…
Updated Monticello (1 of 2)

- Definition-based package reader/writer focused exclusively on Filetree and Tonel
- Set of definitions expanded to include package and project definitions
- Atomic **project** loading

… *Updated Monticello (2 of 2)*

- First class objects representing loaded project/package/class/method
- Loaded “things” directly updated by the **project** loader
- Every class and method “knows” which package/project it is a member of without expensive lookup

Rowan definitions API create/load (1 of 3)

```
projectName := ‘Project’.
packageName1 := ‘Project-Core’.
packageName2 := ‘Project-Extensions’.

"create project"
projectDefinition := Rowan projectTools create
    createDiskBasedProjectDefinition: projectName
    packageNames: { packageName1. packageName2 }
    format: ‘tonel’
    root: ‘$ROWAN_PROJECTS_HOME’.

“create project content”

    “...Following slides ...”

"load project definition"
Rowan projectTools load
    loadProjectDefinition: projectDefinition.
```

Rowan definitions API class/method creation (2 of 3)

```
"create classes and methods"
classDefinition := (RwClassDefinition
    newForClassNamed: ’MyClass’
        super: 'Object'
        instvars: #()
        classinstvars: #()
        classvars: #()
        category: ‘MyCategory’
        comment: ''
        pools: #()
        type: 'normal')
    addInstanceMethodDefinition:
            (RwMethodDefinition
                newForSource: 'method1 ^1'
                protocol: 'accessing').

(projectDefinition packageNamed: packageName1)
    addClassDefinition: classDefinition.
```

Rowan definitions API extension method creation (3 of 3)

```
"create classes extension methods"

classExtensionDefinition :=
    (RwClassExtensionDefinition newForClassNamed: 'Object')
        addInstanceMethodDefinition:
            (RwMethodDefinition
                newForSource: 'extension1 ^1'
                protocol: 'accessing');
        yourself.

(projectDefinition packageNamed: packageName2)
    addClassExtension: classExtensionDefinition.
```

...
Replacement for Metacello

Comparison between Metacello and Rowan

Rowan project loading

```
Monticello new
 baseline: 'RowanSample1';
 repository: 'github://dalehenrich/RowanSample1:sample/rowan/src';
 load: #( ‘core’ )
```

```
url := ‘file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston’.

Rowan projectTools load
    loadProjectFromSpecUrl: url.
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

https://raw.githubusercontent.com/GemTalk/Rowan/master/samples/RowanSample1.ston

RwSimpleProjectSpecification

```
Monticello new
 baseline: 'RowanSample1';
 repository: 'github://dalehenrich/RowanSample1:sample/rowan/src';
 load: #( ‘core’ )
```

```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’ ],
    #comment : 'Sample project load specification'
}
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

… RwSimpleProjectSpecification

```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’ ],
    #comment : 'Sample project load specification'
}
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

… RwSimpleProjectSpecification

```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’ ],
    #comment : 'Sample project load specification'
}
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

BaselineOfRowanSample1 packages and groups

```
baseline: spec
  <baseline>
  spec
    for: #'common'
    do: [
      spec
        package: 'RowanSample4-Core';
        package: 'RowanSample4-Extensions'
          with: [ spec requires: 'RowanSample4-Core' ];
        package: 'RowanSample1-Tests'
          with: [ spec requires: 'RowanSample4-Extensions' ].
      spec
        group: 'Core' with: #('RowanSample4-Core' 'RowanSample4-Extensions');
        group: 'Tests' with: #('RowanSample1-Tests') ]
```

…
Rowan Configurations

```
RwProjectLoadConfiguration{
    #name : 'Core',
    #definedGroupNames : {
        'Core’ : [],
        'Tests’ : [ 'Core’ ] },
    #conditionalPackages: {
        [ 'common' ] :
            { 'Core': {
                #packageNames : [
                    'RowanSample4-Core',
                    'RowanSample4-Extensions' ] } },
            { 'Tests': {
                #packageNames : [
                    'RowanSample4-Tests' ] } } },
    #comment : 'conditional packages'
}
```

file:$ROWAN_PROJECTS_HOME/Project/rowan/configs/Core.ston

…
Rowan Configurations

```
RwProjectLoadConfiguration{
    #name : 'Core',
    #configurationNames : [ 'Test' ],
    #comment : ‘Platform independent configuration for RowanSample1.' }
```

```
RwProjectLoadConfiguration{
    #name : 'Common',
    #conditionalPackages: {
        [ 'common' ] : { 'Core': {
            #packageNames : [
                'RowanSample1-Core',
                'RowanSample1-Extensions' ] } } },
    #configurationNames : [ ],
    #comment : 'Platform independent packages' }
```

```
RwProjectLoadConfiguration{
    #name : 'Test',
    #conditionalPackages: {
        [ 'common' ] : { 'Tests': {
            #packageNames : [
                'RowanSample1-Tests’ ] } } },
    #configurationNames : [ ‘Common' ],
    #comment : 'Platform independent Test packages.' }
```

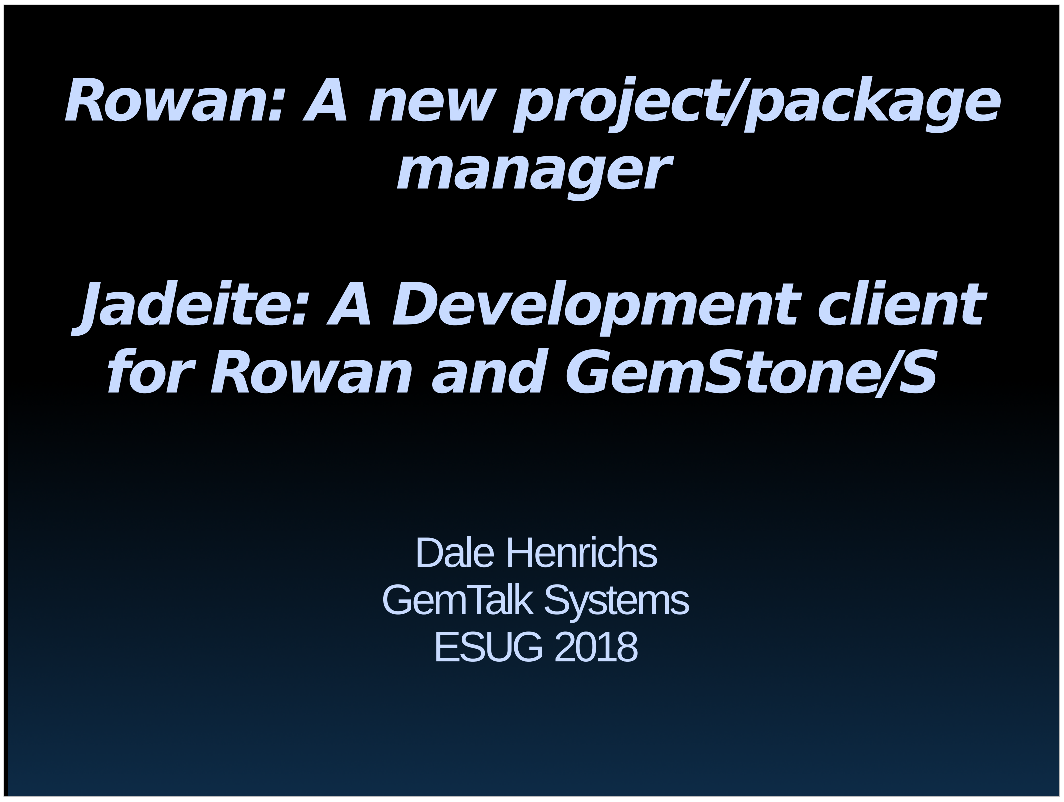
file:$ROWAN_PROJECTS_HOME/Project/rowan/configs/*

Rowan/Jadeite Plans (near term)

- Keep our production customer happy
- Implement missing features in both Rowan and Jadeite
- Rowan support in SmalltalkCI
- Port FileSystem and parts of Zinc to GemStone kernel
- Port Rowan to 3.4
  - Upgrade path from 3.2.15 to 3.4

Rowan/Jadeite Plans (long term)

- Port Rowan to GemStone 3.5
- Package kernel classes for 3.5/3.6
- Create a Pharo-based GUI using services API created for Jadeite
- Integrate Rowan with tODE, GLASS/GsDevKit, GsDevKit_home
- Review/Refactor for dialect portability



Rowan: A new project/package manager

Jadeite: A Development client for Rowan and GemStone/S

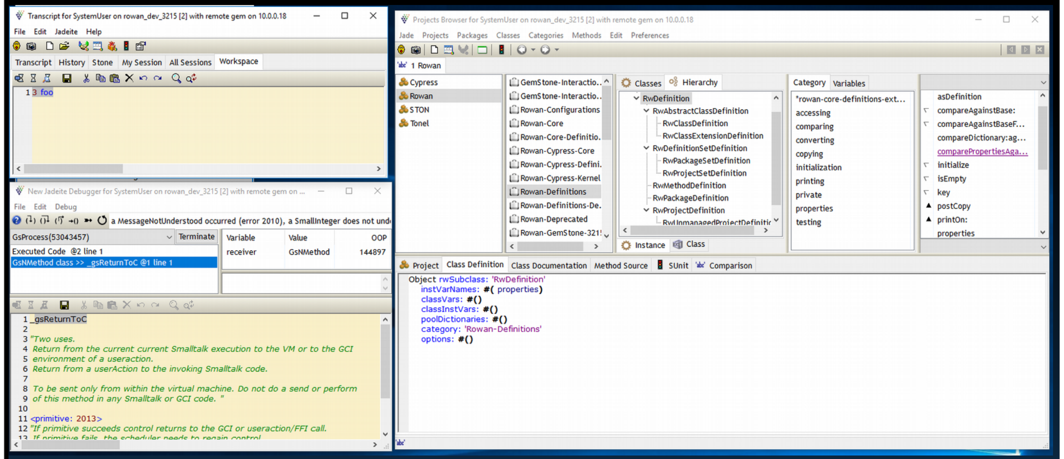
Dale Henrichs
GemTalk Systems
ESUG 2018

Rowan and Jadeite are part of GemTalk's effort to provide a supported development environment for GemStone/S.

Rowan and Jadeite were developed in cooperation with one of our commercial customers over the better part of this year and is currently in production at the customer's site.

Rowan is designed to be support multiple Smalltalk dialects

Jadeite



```
_gsReturnToC

"Two uses.
 Return from the current current Smalltalk execution to the VM or to the GCI
 environment of a useraction.
 Return from a userAction to the invoking Smalltalk code.

 To be sent only from within the virtual machine. Do not do a send or perform
 of this method in any Smalltalk or GCI code. "

<primitive: 2013>
"If primitive succeeds control returns to the GCI or useraction/FFI call.
 If primitive fails, the scheduler needs to regain control.
```

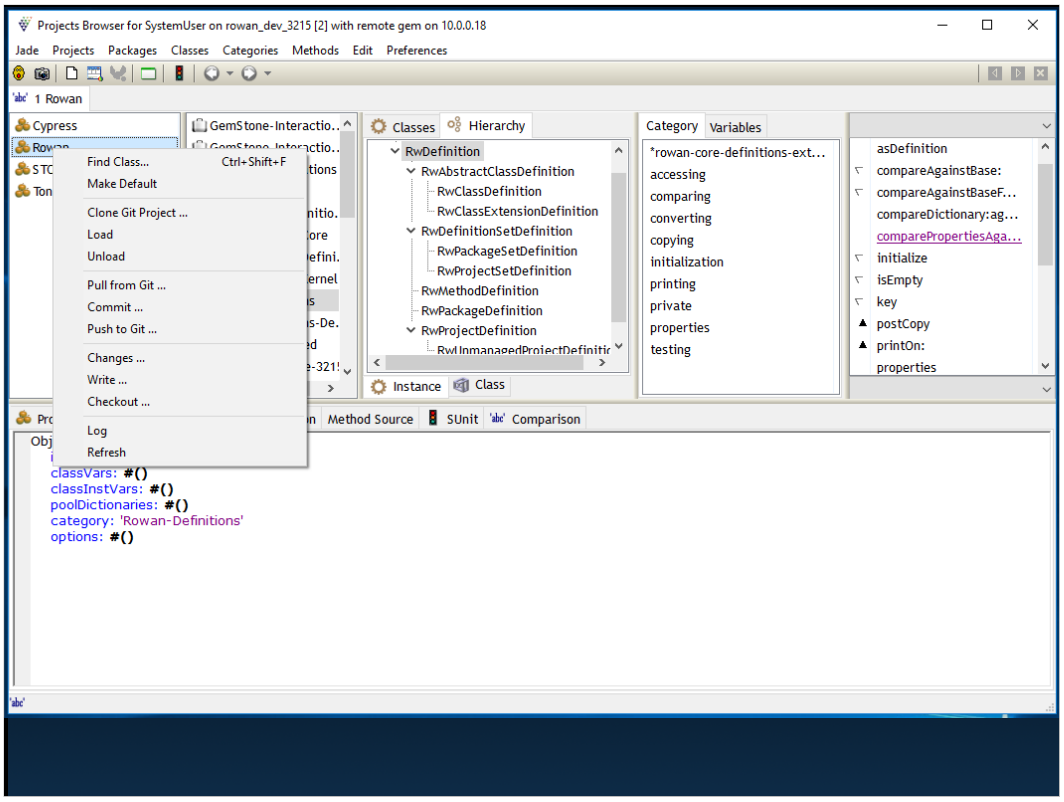
```
Object rwSubclass: 'RwDefinition'
    instVarNames: #( properties)
    classVars: #()
    classInstVars: #()
    poolDictionaries: #()
    category: 'Rowan-Definitions'
    options: #()
```

Jadeite is a Dolphin-based client that implements a browser, inspector, debugger, etc. that supports the Rowan project/package model.

Jadeite is derived from Jade by James Foster

Our client is a windows shop, so Rowanizing Jade was the fastest route to a functional client for Rowan.

Jadeite is a thin client, with most of the business logic for browsers and debuggers running on the server

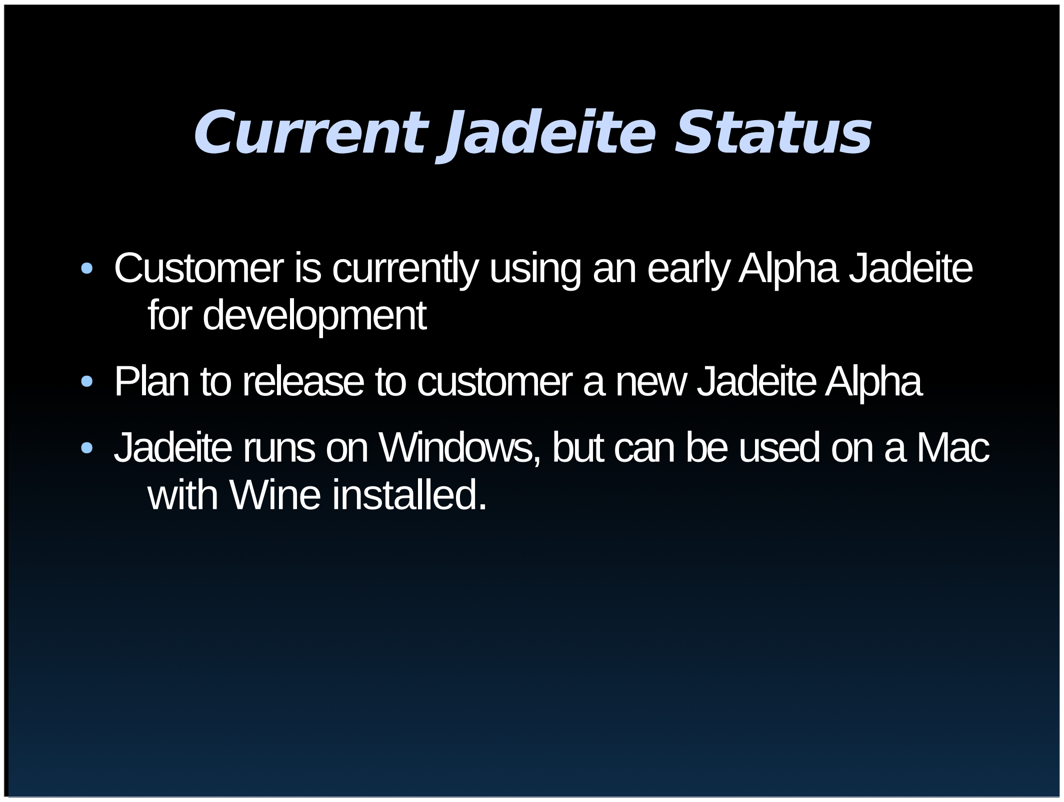


```
Obj
    i
    classVars: #()
    classInstVars: #()
    poolDictionaries: #()
    category: 'Rowan-Definitions'
    options: #()
```

Jadeite basic support for git-based projects.

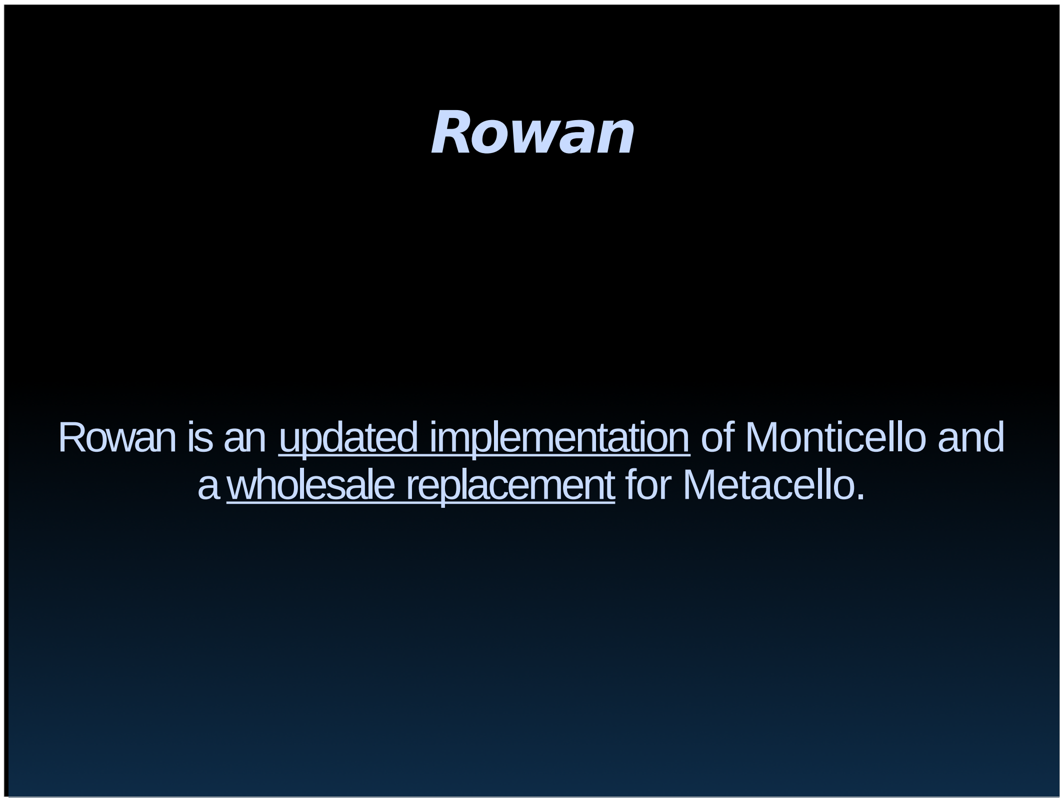
Current Rowan Status

- In production at a GemTalk customer site
- Running on Gemstone/S 3.2.15
- Still not quite feature complete



Current Jadeite Status

- Customer is currently using an early Alpha Jadeite for development
- Plan to release to customer a new Jadeite Alpha
- Jadeite runs on Windows, but can be used on a Mac with Wine installed.

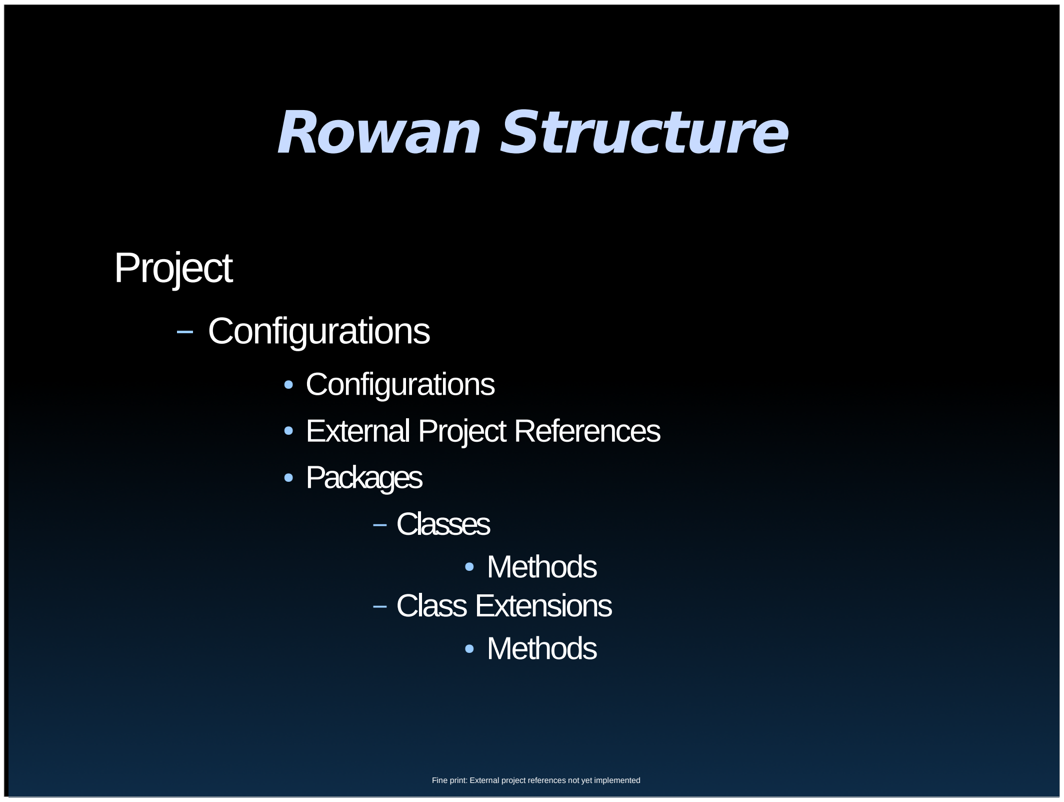


Rowan

Rowan is an updated implementation of Monticello and a wholesale replacement for Metacello.

Monticello is 15 years old, Metacello is 10 years old and the software management world for Smalltalk has changed significantly since they were both created.

FileTree, Tonel, Git support, and STON did not exist when Monticello and Metacello were first created.



Rowan Structure

Project

- Configurations
  - Configurations
  - External Project References
  - Packages
    - Classes
      - Methods
    - Class Extensions
      - Methods

Fine print: External project references not yet implemented

Multiple projects per image

Configurations provide organization for packages

Packages are similar to what you are used to with Monticello

Now I’ll talk a bit more some of these newer structural elements

Rowan project (in image)

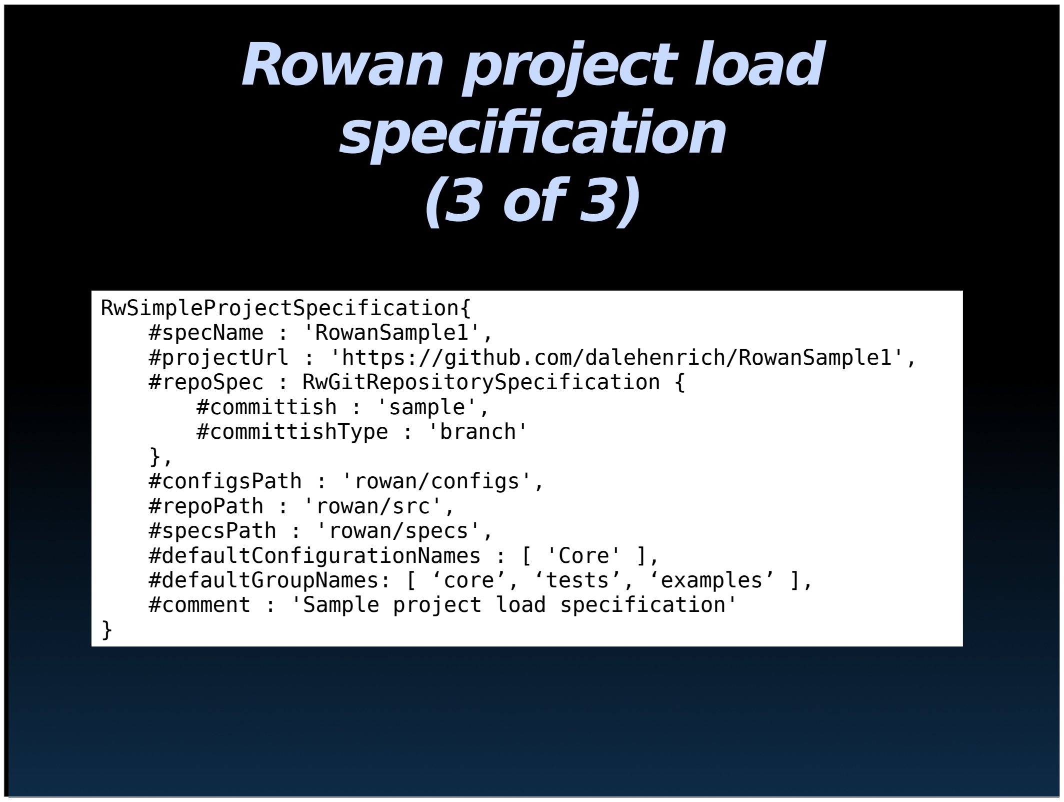
- Named
- Associated with a set of directories on disk
- SCM neutral, but Git aware
- Loaded using a `project load specification`

Rowan project load specification (1 of 3)

- Named
- Specifies disk structure:
  - src directory for packages
  - configs directory for configurations
  - specs directory for load specs
- Specifies load parameters
  - List of configuration names and group names
  - Remote repository specifics

Rowan project load specification (2 of 3)

- Instance-based (declarative)
  - Stored as STON on disk
  - Potential for multiple implementations
    - depending upon needs of project
    - Support future expansion
- Located anywhere
  - Meant to be copied and customized
  - Typically reference via an url (\`file:\` or \`https:\` or ???)

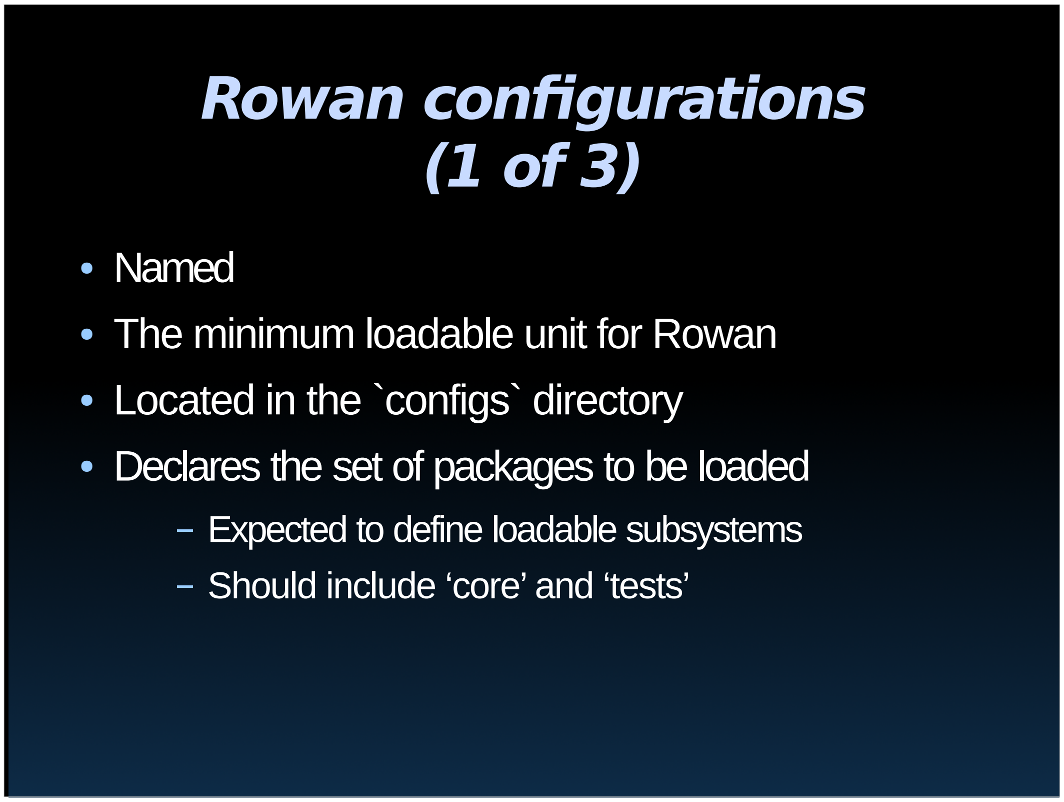


Rowan project load specification (3 of 3)

```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’, ‘tests’, ‘examples’ ],
    #comment : 'Sample project load specification'
}
```

This is a (obviously) a STON file and as such is pretty readable …

I will cover the specific fields in more detail later.

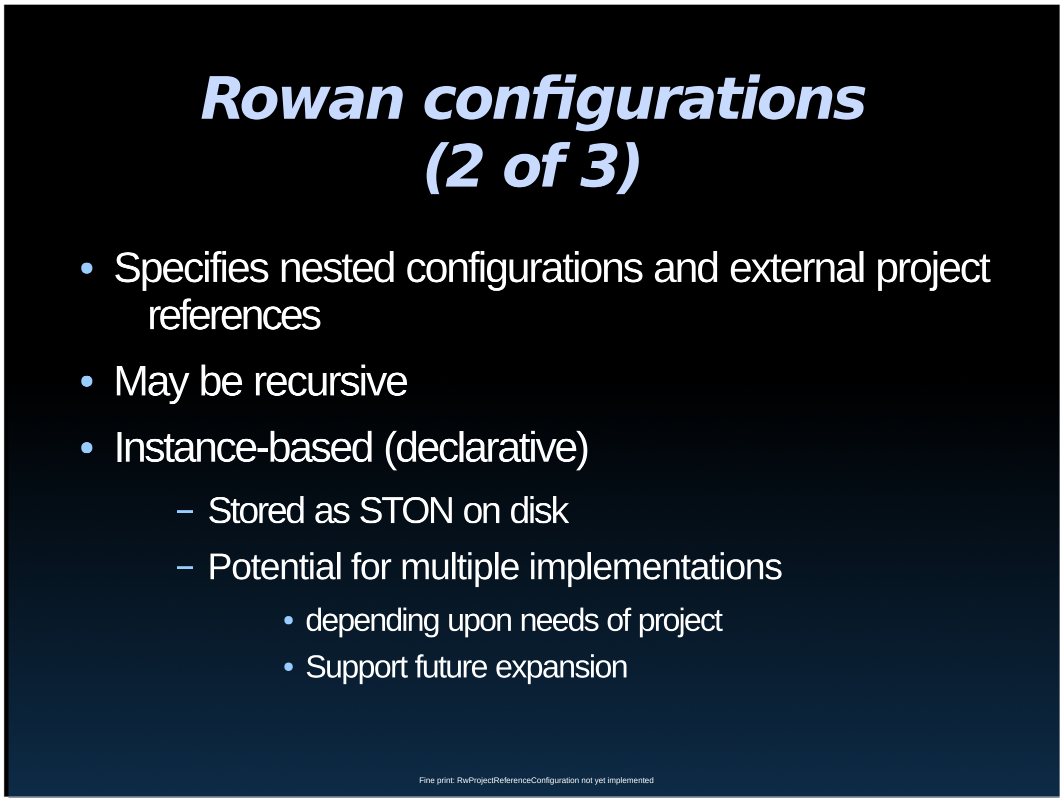


Rowan configurations (1 of 3)

- Named
- The minimum loadable unit for Rowan
- Located in the `configs` directory
- Declares the set of packages to be loaded
  - Expected to define loadable subsystems
  - Should include ‘core’ and ‘tests’

With Metacello you are allowed to load a single package from a baseline … but there is no guarantee that things will work.

With a configuration there is an explicit contract between project owners and project users about what combinations of packages are supported

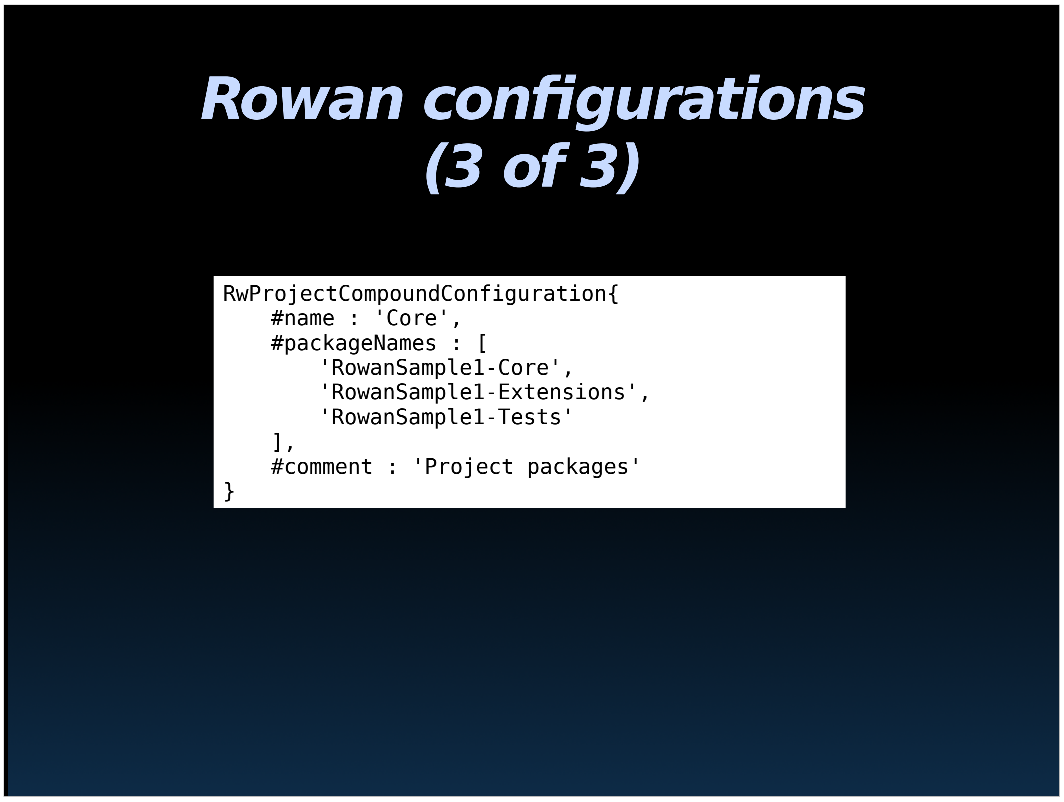


Rowan configurations (2 of 3)

- Specifies nested configurations and external project references
- May be recursive
- Instance-based (declarative)
  - Stored as STON on disk
  - Potential for multiple implementations
    - depending upon needs of project
    - Support future expansion

Fine print: RwProjectReferenceConfiguration not yet implemented

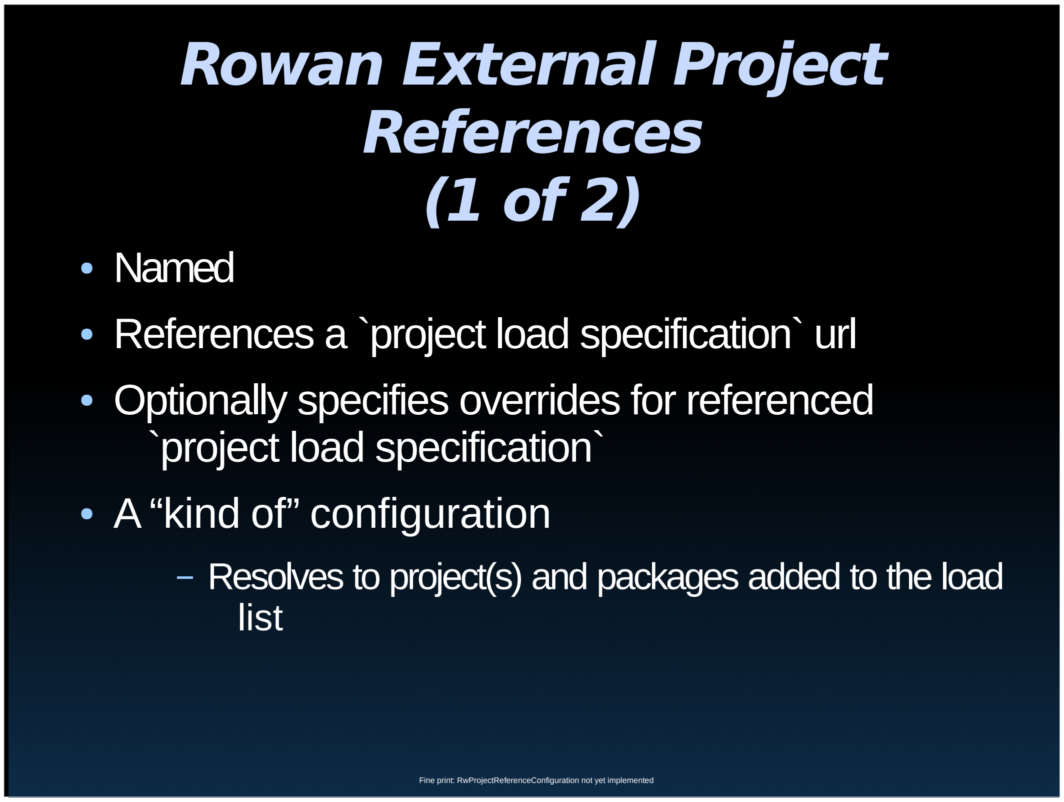
FINE PRINT



Rowan configurations (3 of 3)

```
RwProjectCompoundConfiguration{
    #name : 'Core',
    #packageNames : [
        'RowanSample1-Core',
        'RowanSample1-Extensions',
        'RowanSample1-Tests'
    ],
    #comment : 'Project packages'
}
```

Example of simplest configuration possible
…. a list of 3 packages.
I will show a few more complicated configration options later on...

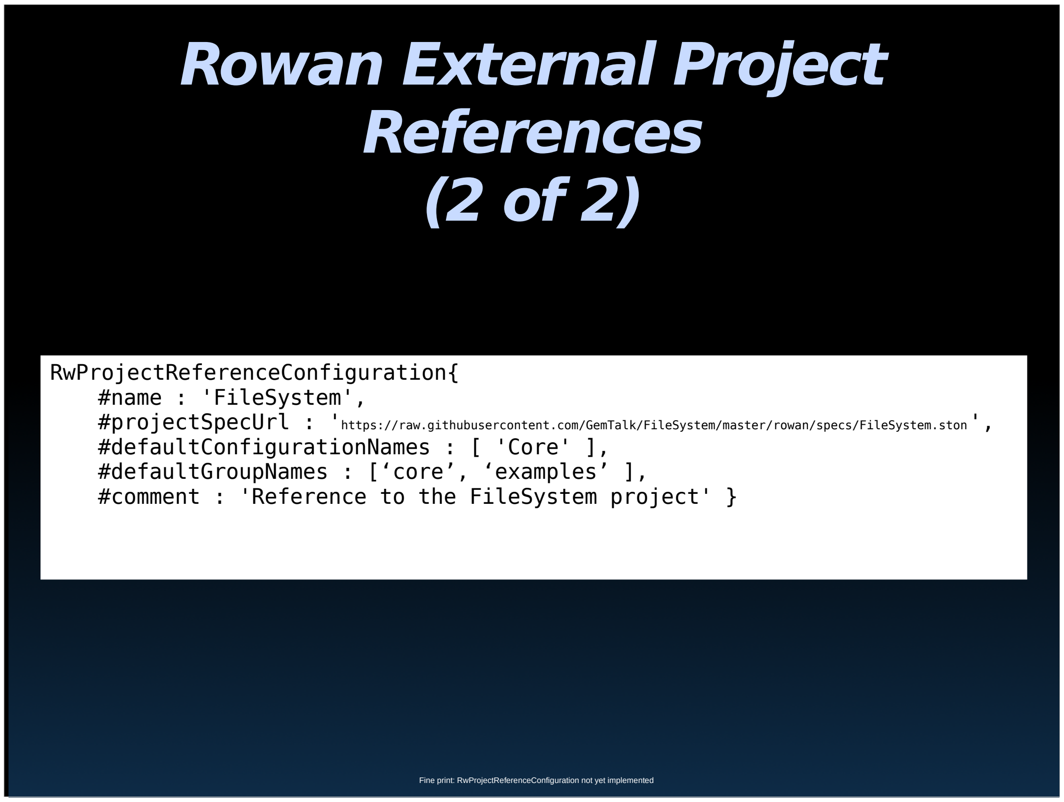


Rowan External Project References (1 of 2)

- Named
- References a `project load specification` url
- Optionally specifies overrides for referenced `project load specification`
- A “kind of” configuration
  - Resolves to project(s) and packages added to the load list

Fine print: RwProjectReferenceConfiguration not yet implemented

FINE PRINT!



Rowan External Project References (2 of 2)

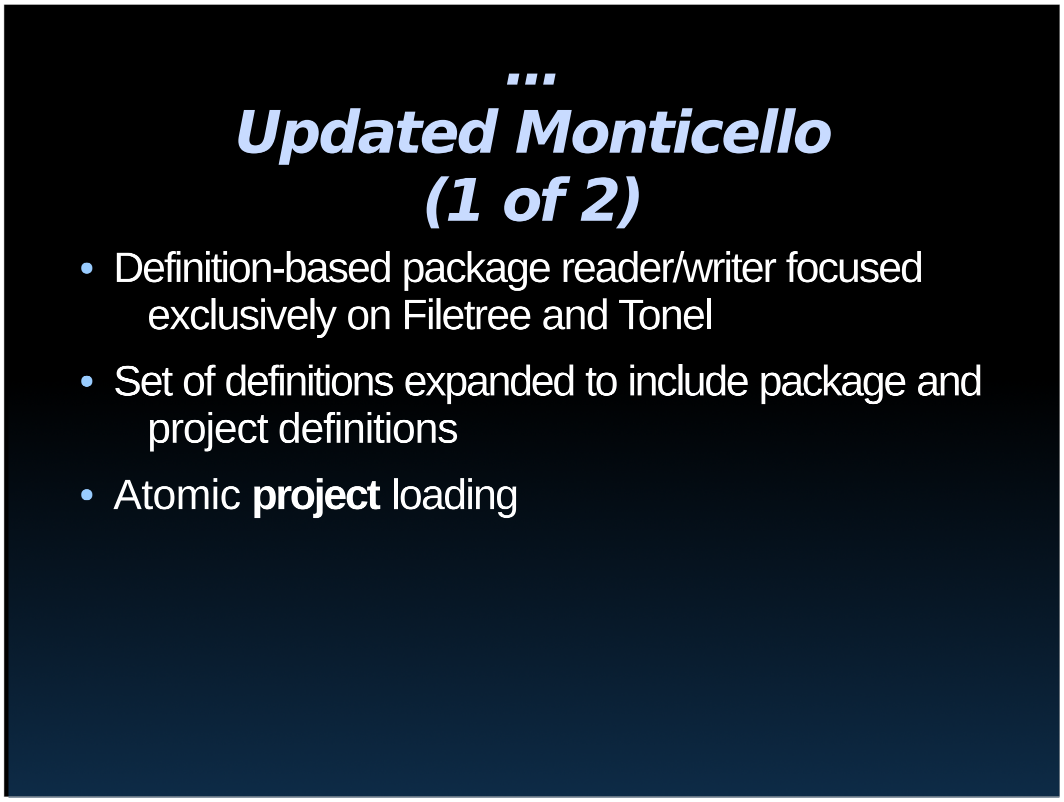
```
RwProjectReferenceConfiguration{
    #name : 'FileSystem',
    #projectSpecUrl : 'https://raw.githubusercontent.com/GemTalk/FileSystem/master/rowan/specs/FileSystem.ston',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames : [‘core’, ‘examples’ ],
    #comment : 'Reference to the FileSystem project' }
```

Fine print: RwProjectReferenceConfiguration not yet implemented

FINE PRINT

The tiny url is a direct https: reference to the STON file in the specs directory of a GitHub project.

The reference will load the Core configuration using the core and examples groups (presumably not loading the tests group)



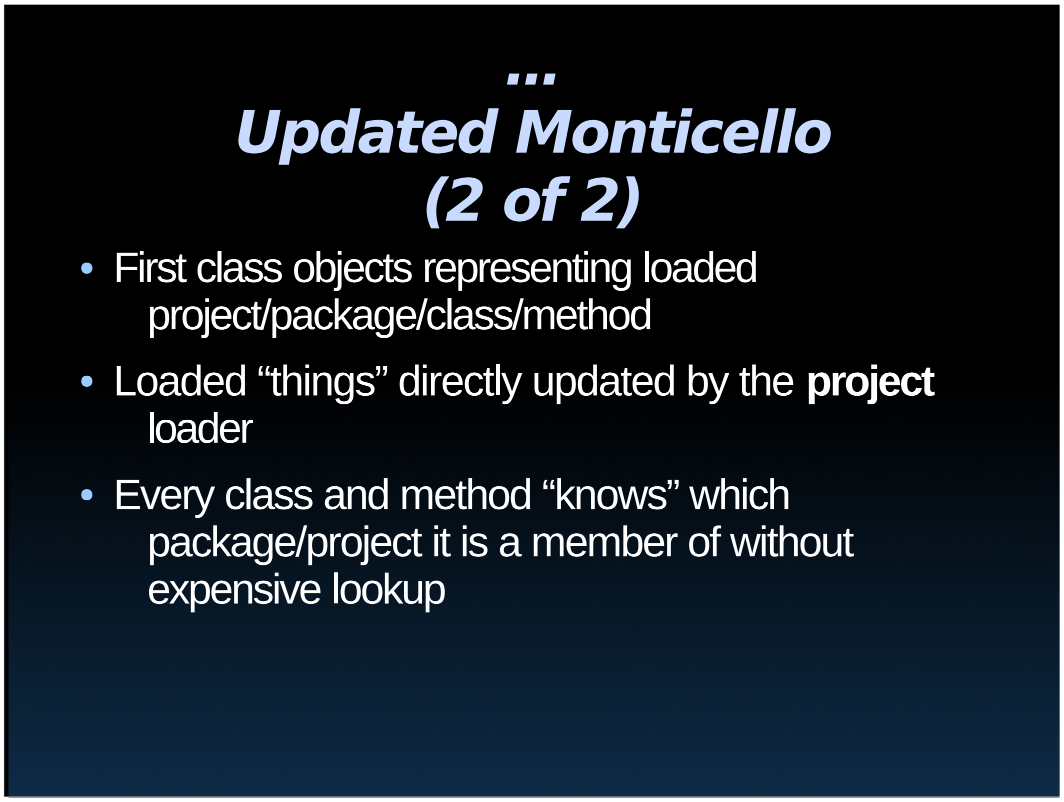
…
Updated Monticello (1 of 2)

- Definition-based package reader/writer focused exclusively on Filetree and Tonel
- Set of definitions expanded to include package and project definitions
- Atomic **project** loading

Done with Structure …

This is for the Monticello nerds in th audience:)

… yes Rowan is really a project loader and a collection of projects can be loaded in one atomic operation

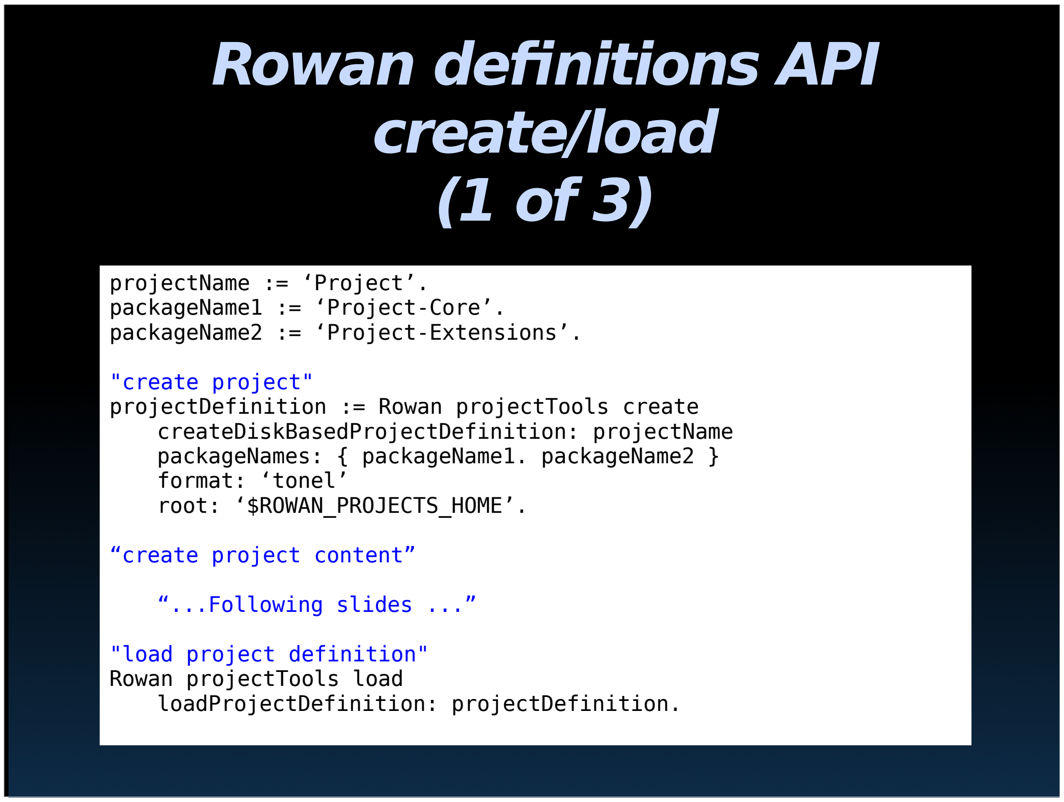


…
Updated Monticello (2 of 2)

- First class objects representing loaded project/package/class/method
- Loaded "things" directly updated by the **project** loader
- Every class and method "knows" which package/project it is a member of without expensive lookup

For the very large projects we see in GemStone, it isn't practical to scan all classes and methods in the image to determine package membership

And this is one of the reasons we felt it necessary to update Monticello



Rowan definitions API create/load (1 of 3)

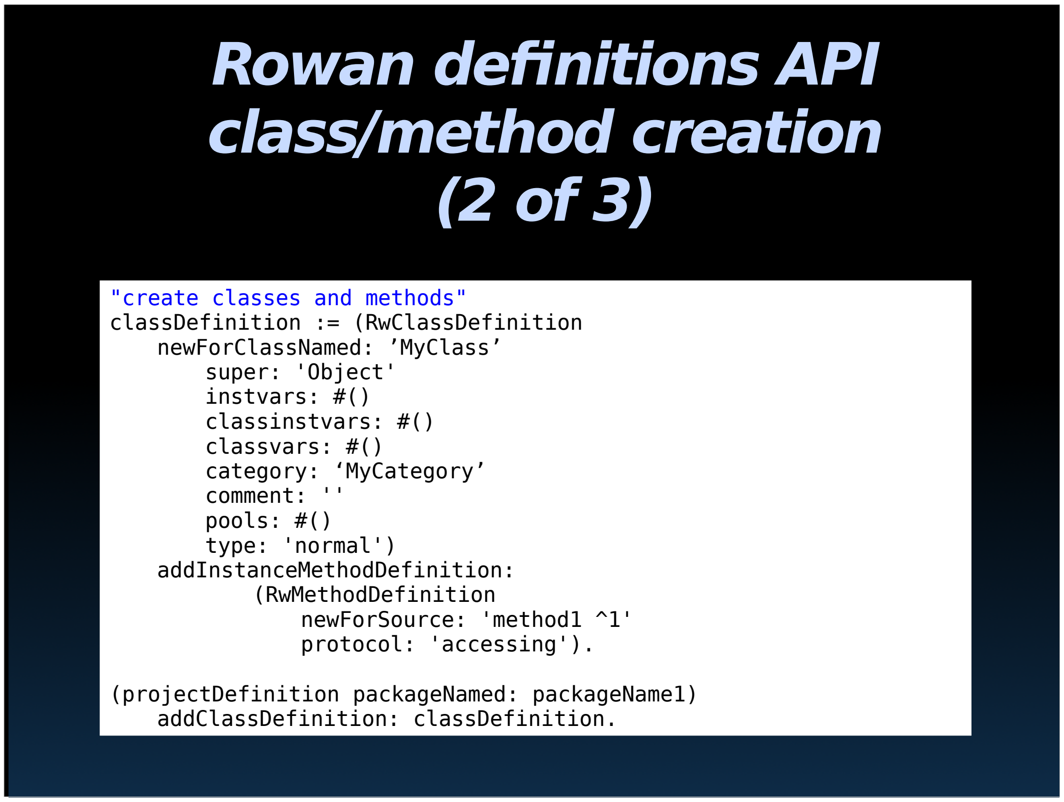
```
projectName := ‘Project’.
packageName1 := ‘Project-Core’.
packageName2 := ‘Project-Extensions’.

"create project"
projectDefinition := Rowan projectTools create
    createDiskBasedProjectDefinition: projectName
    packageNames: { packageName1. packageName2 }
    format: ‘tonel’
    root: ‘$ROWAN_PROJECTS_HOME’.

“create project content”

    “...Following slides ...”

"load project definition"
Rowan projectTools load
    loadProjectDefinition: projectDefinition.
```

Here we are creating a project definition of a certain type (disk base -– no SCM)

$ROWAN_PROJECTS_HOME is an evironment variable that I’m using to indicate the home directory for git clones …

#loadedProjectDefinition: is a fundamental operation and the important thing to note here is that when doing a load, there is no expectation of where the project definition has come from …

It could be disk, it could be created programatically as in this example.

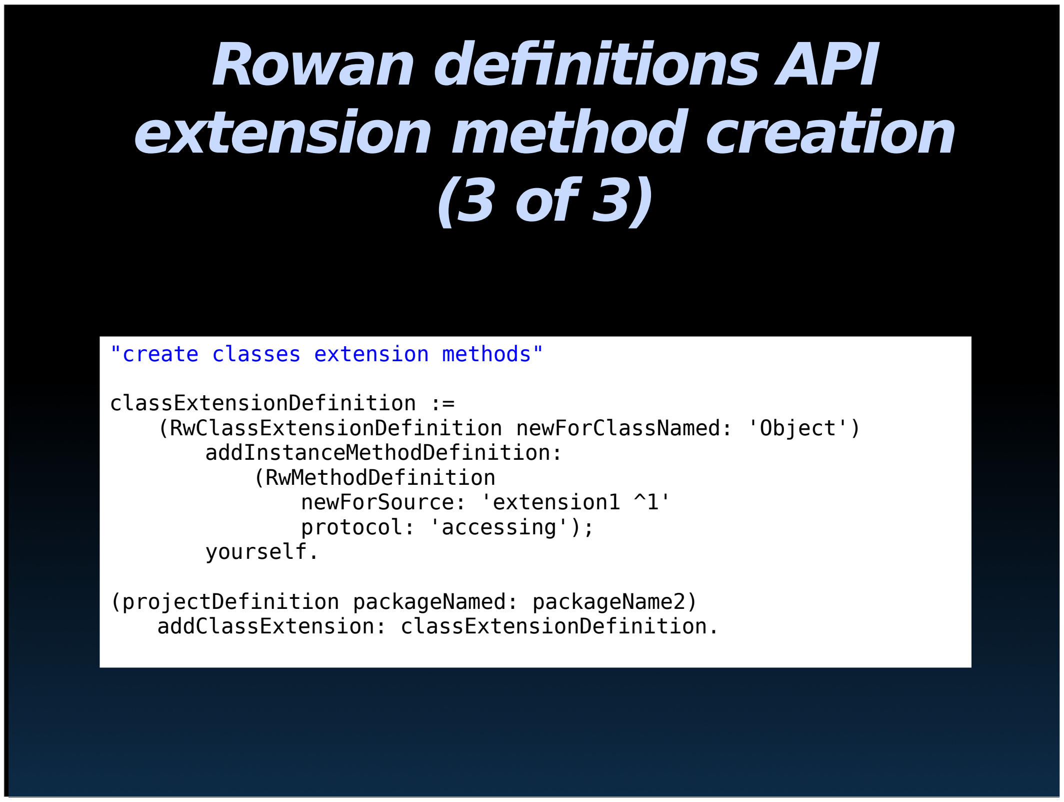


Rowan definitions API class/method creation (2 of 3)

```
"create classes and methods"
classDefinition := (RwClassDefinition
    newForClassNamed: ’MyClass’
        super: 'Object'
        instvars: #()
        classinstvars: #()
        classvars: #()
        category: ‘MyCategory’
        comment: ''
        pools: #()
        type: 'normal')
    addInstanceMethodDefinition:
            (RwMethodDefinition
                newForSource: 'method1 ^1'
                protocol: 'accessing').

(projectDefinition packageNamed: packageName1)
    addClassDefinition: classDefinition.
```

Another slide for the Monticello nerds …

It’s actually fun to work with Rowan definitions ...



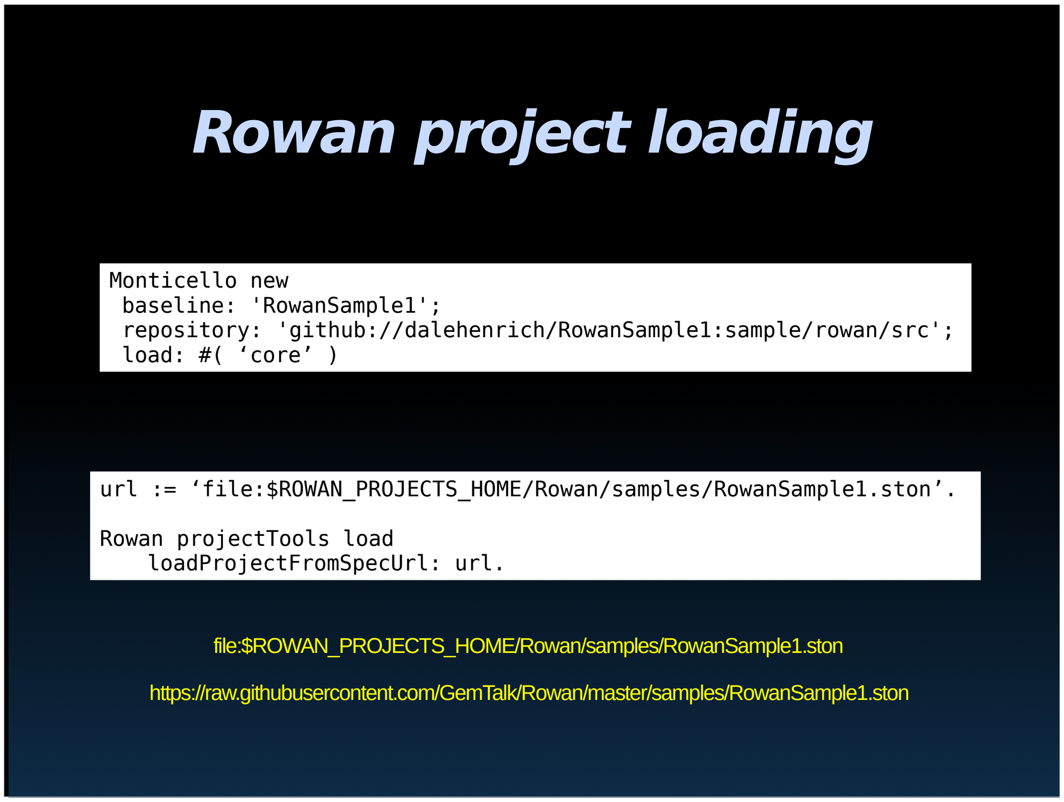
Rowan definitions API extension method creation (3 of 3)

```
"create classes extension methods"

classExtensionDefinition :=
    (RwClassExtensionDefinition newForClassNamed: 'Object')
        addInstanceMethodDefinition:
            (RwMethodDefinition
                newForSource: 'extension1 ^1'
                protocol: 'accessing');
        yourself.

(projectDefinition packageNamed: packageName2)
    addClassExtension: classExtensionDefinition.
```

There is a corresponding #addClassMethodDefinition: method ...

...
Replacement for Metacello

Comparison between Metacello and Rowan



Rowan project loading

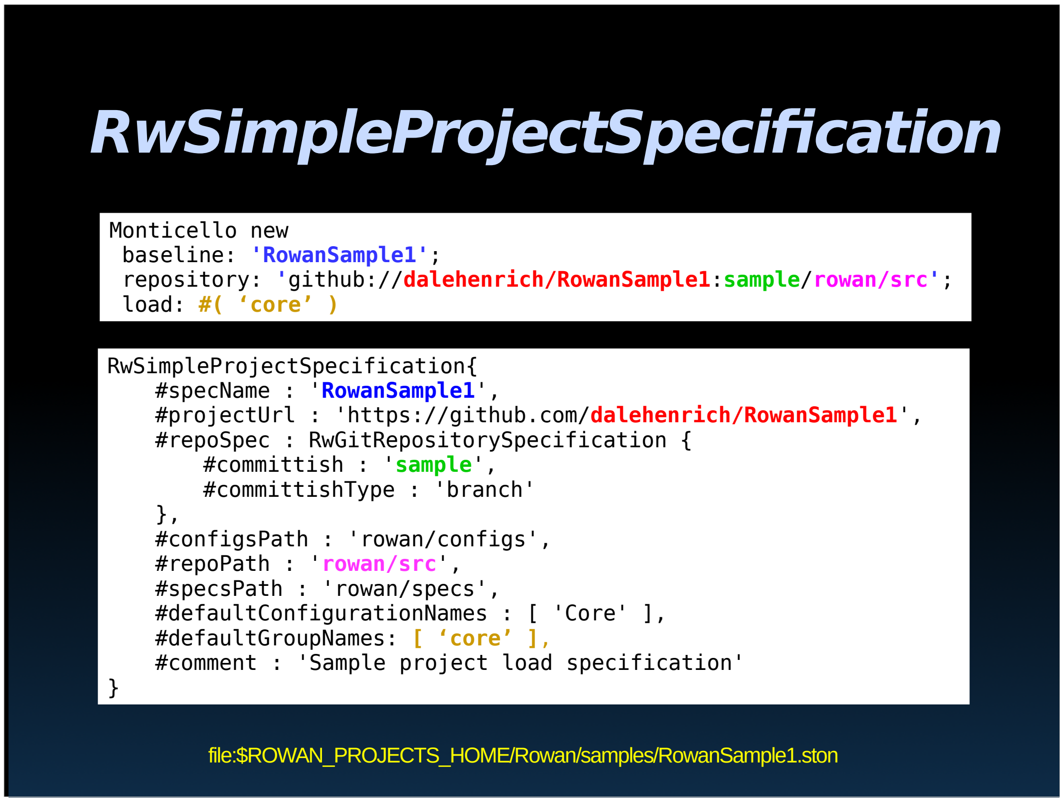
```
Monticello new
 baseline: 'RowanSample1';
 repository: 'github://dalehenrich/RowanSample1:sample/rowan/src';
 load: #( ‘core’ )
```

```
url := ‘file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston’.

Rowan projectTools load
    loadProjectFromSpecUrl: url.
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

https://raw.githubusercontent.com/GemTalk/Rowan/master/samples/RowanSample1.ston

The first change with Rowan is how projects are loaded.
The functionality of Metacello has been absorbed into Rowan, so the need for a meta-load expression is gone.
The (familiar) Metacello load expression is replaced by a project load specification URL…
The url may reference a STON file on disk, or a STON file on the network.
The STON file itself is project load specification...



RwSimpleProjectSpecification

```
Monticello new
 baseline: 'RowanSample1';
 repository: 'github://dalehenrich/RowanSample1:sample/rowan/src';
 load: #( ‘core’ )
```

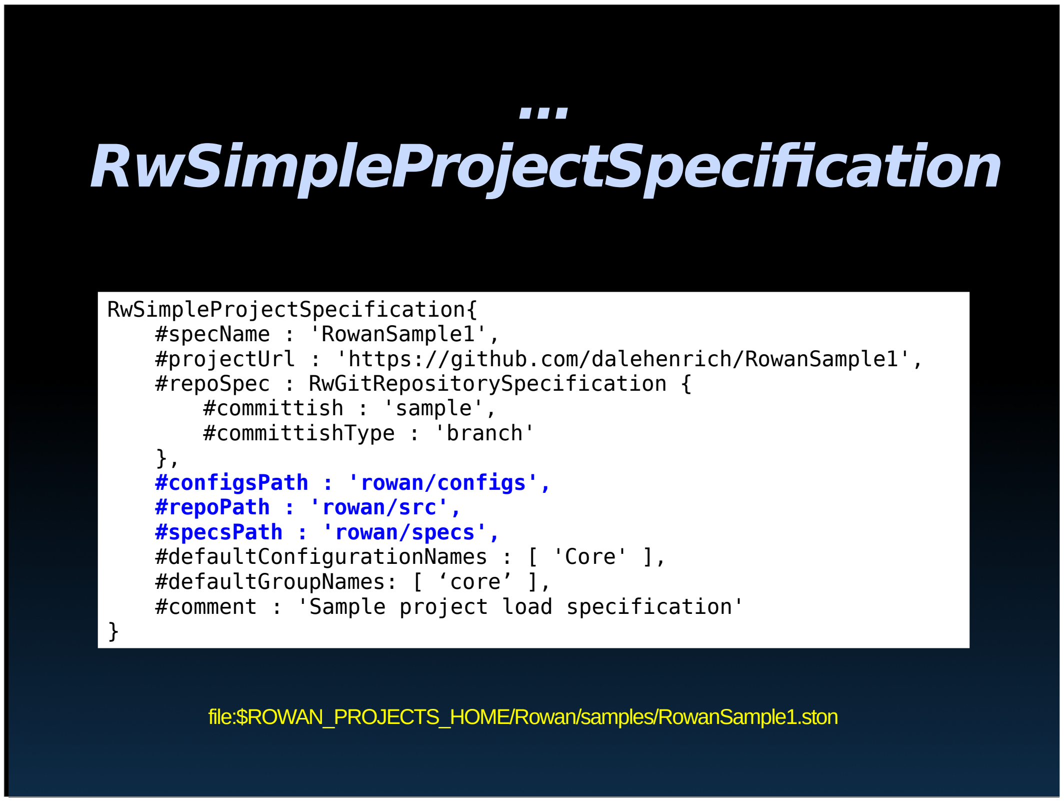
```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’ ],
    #comment : 'Sample project load specification'
}
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

Now we look into the guts of the project load specification.

The highlighted bits shows the correspondence to information provided in Metacello load expression and the Rowan load specification…

Rowan preserves the basic sematics of the Metacello load expression



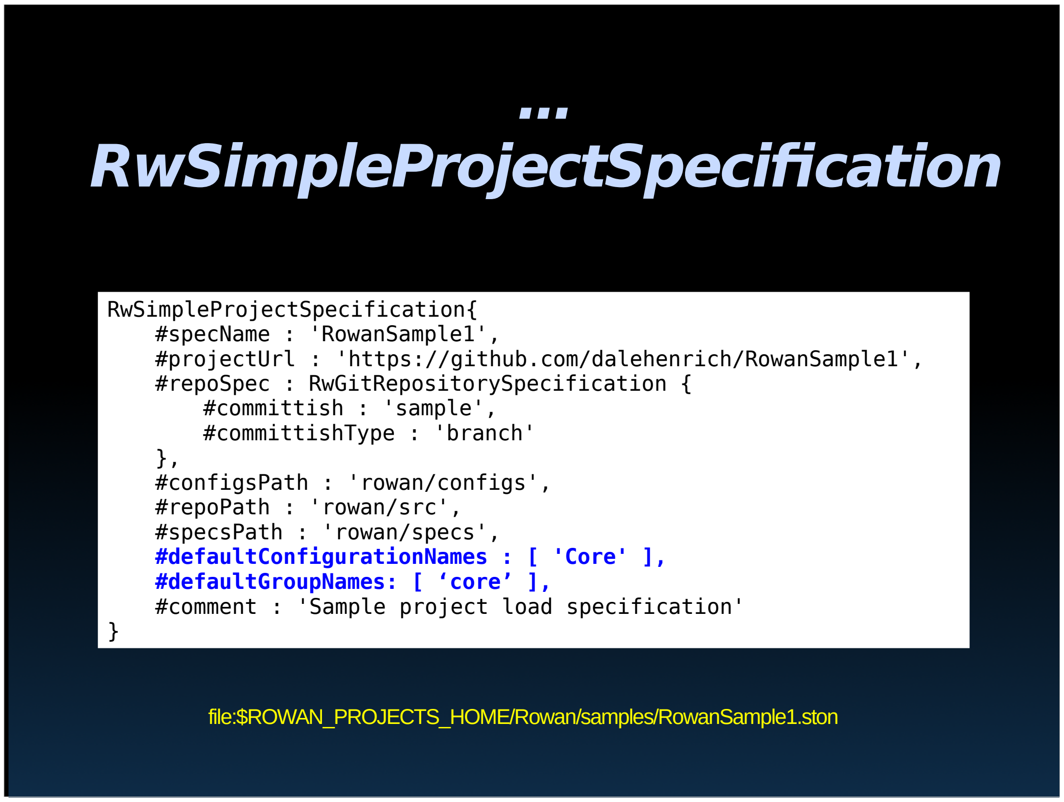
… RwSimpleProjectSpecification

```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’ ],
    #comment : 'Sample project load specification'
}
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

The params specify project directory structure.

The use of separate directories for the Rowan configs and specs means that a project can be Rowanized non-invasively…



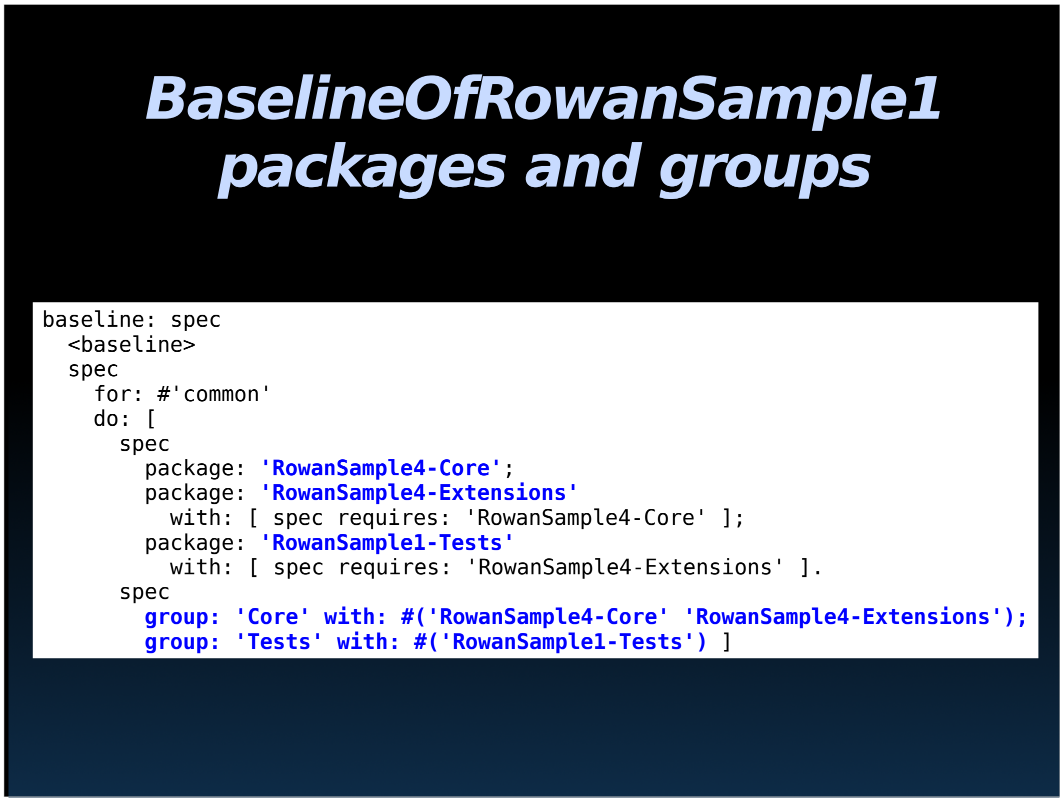
… RwSimpleProjectSpecification

```
RwSimpleProjectSpecification{
    #specName : 'RowanSample1',
    #projectUrl : 'https://github.com/dalehenrich/RowanSample1',
    #repoSpec : RwGitRepositorySpecification {
        #committish : 'sample',
        #committishType : 'branch'
    },
    #configsPath : 'rowan/configs',
    #repoPath : 'rowan/src',
    #specsPath : 'rowan/specs',
    #defaultConfigurationNames : [ 'Core' ],
    #defaultGroupNames: [ ‘core’ ],
    #comment : 'Sample project load specification'
}
```

file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston

The default config names and group names allow project maintainers to recommend a starting point …

It’s worth noting that since specs are in a directory, maintainers can recommend multiple specification/configurations.

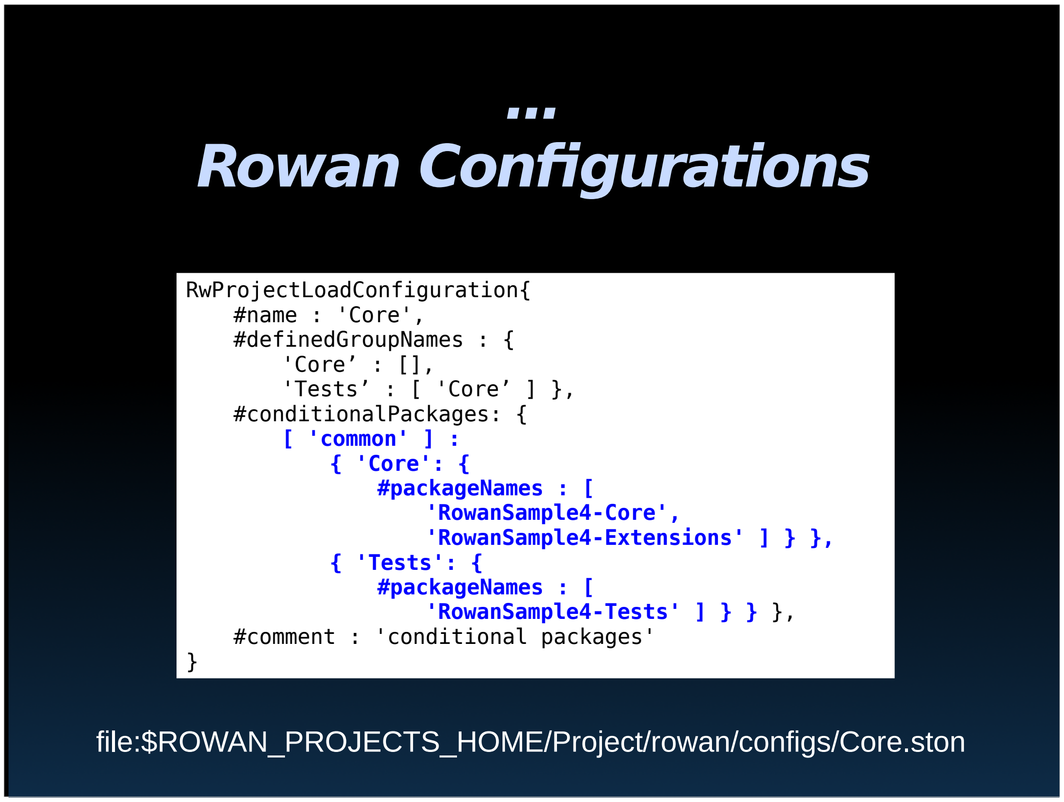


BaselineOfRowanSample1 packages and groups

```
baseline: spec
  <baseline>
  spec
    for: #'common'
    do: [
      spec
        package: 'RowanSample4-Core';
        package: 'RowanSample4-Extensions'
          with: [ spec requires: 'RowanSample4-Core' ];
        package: 'RowanSample1-Tests'
          with: [ spec requires: 'RowanSample4-Extensions' ].
      spec
        group: 'Core' with: #('RowanSample4-Core' 'RowanSample4-Extensions');
        group: 'Tests' with: #('RowanSample1-Tests') ]
```

Now we’ll look at the differences between a Metacello BaselineOf and a Rowan configuration

This baseline has 3 packages and 2 groups ...



…
Rowan Configurations

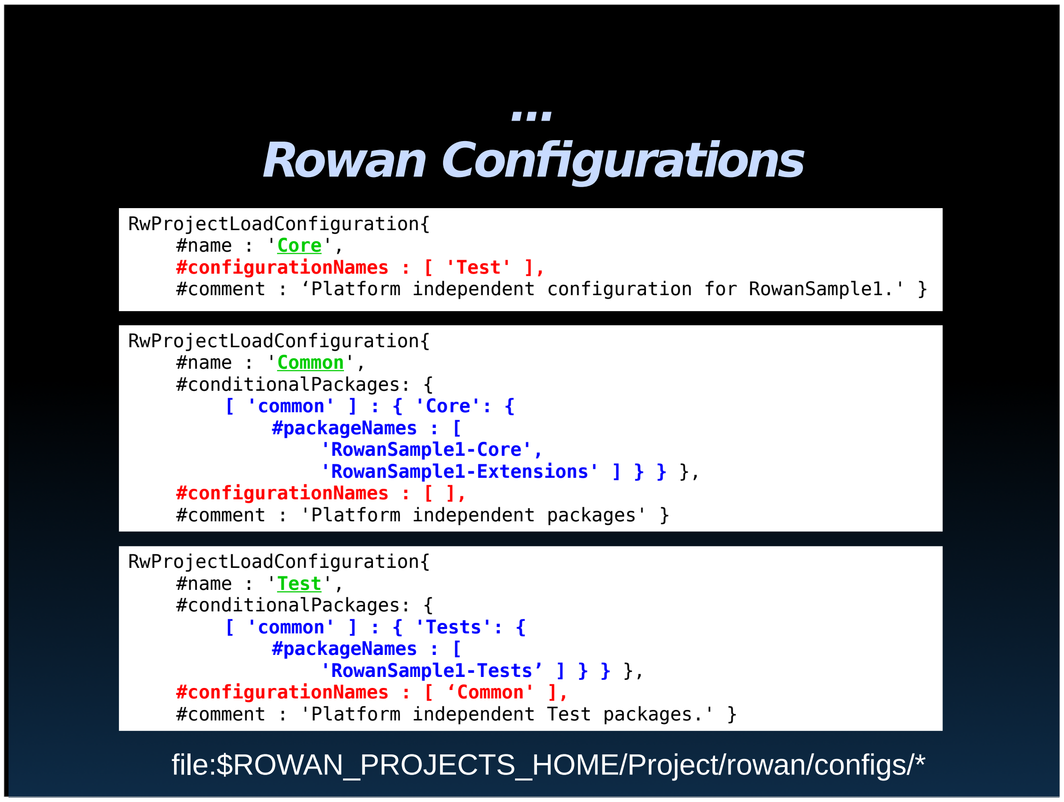
```
RwProjectLoadConfiguration{
    #name : 'Core',
    #definedGroupNames : {
        'Core’ : [],
        'Tests’ : [ 'Core’ ] },
    #conditionalPackages: {
        [ 'common' ] :
            { 'Core': {
                #packageNames : [
                    'RowanSample4-Core',
                    'RowanSample4-Extensions' ] } },
            { 'Tests': {
                #packageNames : [
                    'RowanSample4-Tests' ] } } },
    #comment : 'conditional packages'
}
```

file:$ROWAN_PROJECTS_HOME/Project/rowan/configs/Core.ston

This is a slightly more complicated configuration implementation than you saw earlier ..
In this case we have the platform defined as ‘common’ and packages are directly gathered together into groups.
What is missing from Metacello is package dependencies

With atomic loads, dependencies aren’t required
If package dependencies are needed, a more complicated specification class can be defined to permit it …

It’s worth mentioning at this point that an advantage to going with the instance-based approach is that it’s possible to analyze configurations with non-Rowan tools … and configuration implemantations can contain project/dialect specific attributes



…
Rowan Configurations

```
RwProjectLoadConfiguration{
    #name : 'Core',
    #configurationNames : [ 'Test' ],
    #comment : ‘Platform independent configuration for RowanSample1.' }
```

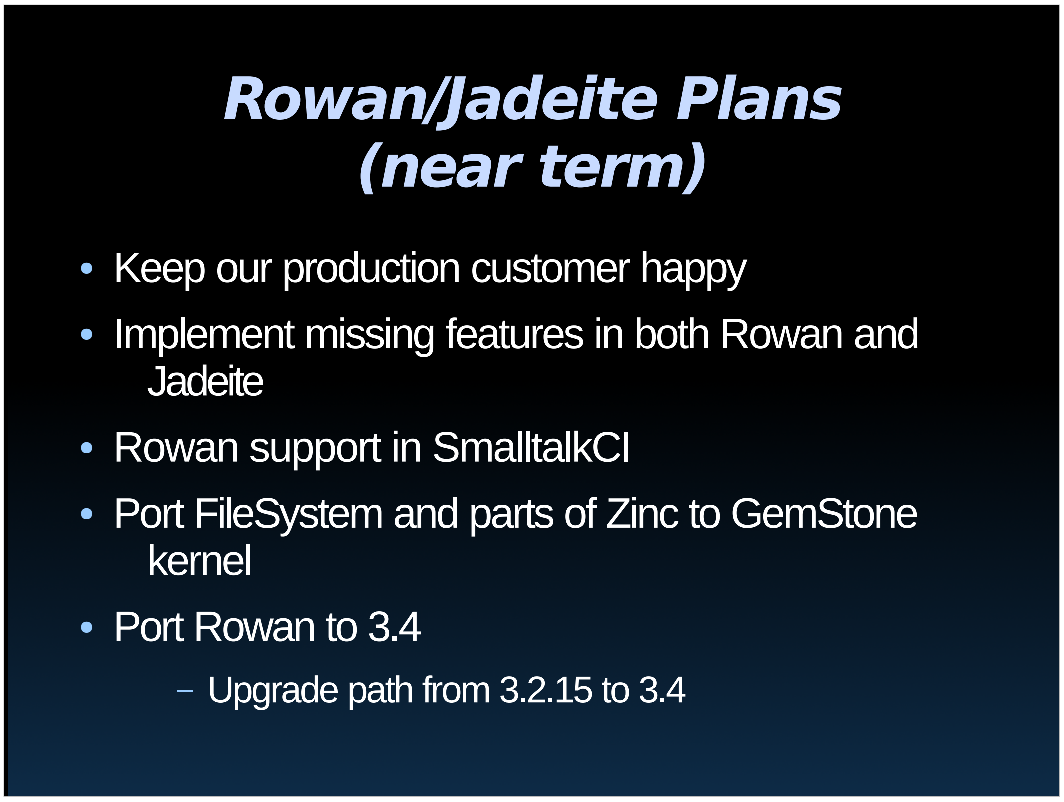
```
RwProjectLoadConfiguration{
    #name : 'Common',
    #conditionalPackages: {
        [ 'common' ] : { 'Core': {
            #packageNames : [
                'RowanSample1-Core',
                'RowanSample1-Extensions' ] } } },
    #configurationNames : [ ],
    #comment : 'Platform independent packages' }
```

```
RwProjectLoadConfiguration{
    #name : 'Test',
    #conditionalPackages: {
        [ 'common' ] : { 'Tests': {
            #packageNames : [
                'RowanSample1-Tests’ ] } } },
    #configurationNames : [ ‘Common' ],
    #comment : 'Platform independent Test packages.' }
```

file:$ROWAN_PROJECTS_HOME/Project/rowan/configs/*

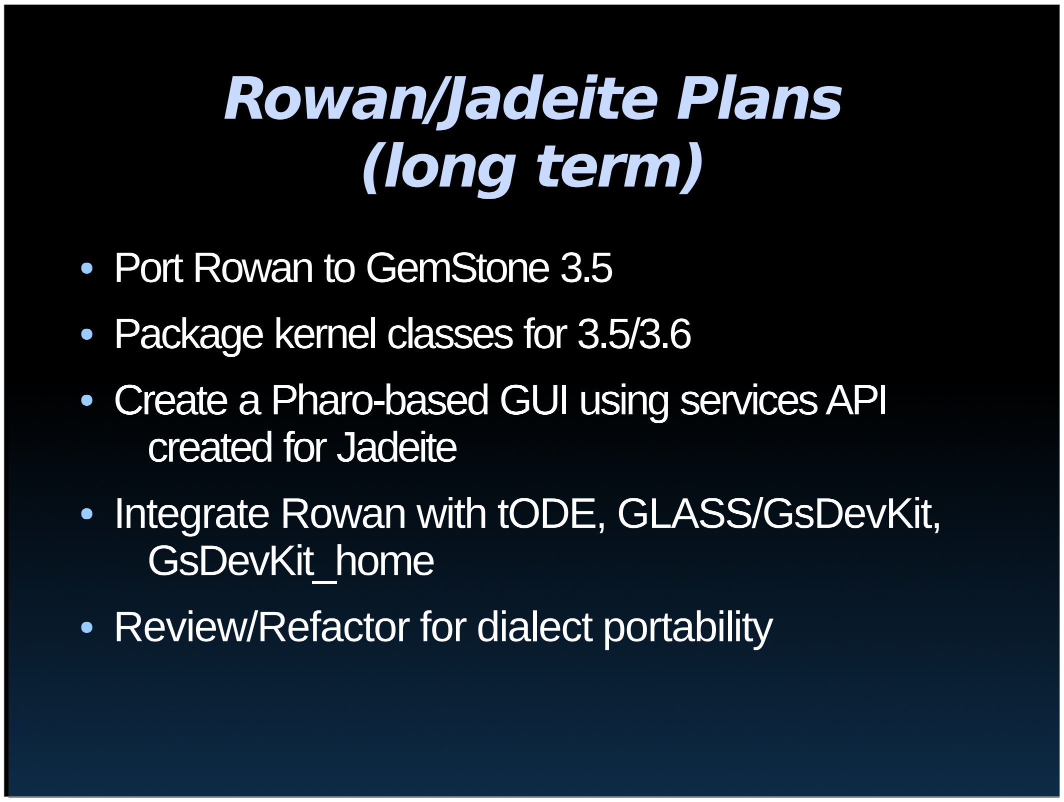
The final bit I want to show is that configurations can be nested to provide additional structure for projects that have a large number of packages…

Worth noting that that the plan is to display the configuration structures in Jadeite...



Rowan/Jadeite Plans (near term)

- Keep our production customer happy
- Implement missing features in both Rowan and Jadeite
- Rowan support in SmalltalkCI
- Port FileSystem and parts of Zinc to GemStone kernel
- Port Rowan to 3.4
  - Upgrade path from 3.2.15 to 3.4



Rowan/Jadeite Plans (long term)

- Port Rowan to GemStone 3.5
- Package kernel classes for 3.5/3.6
- Create a Pharo-based GUI using services API created for Jadeite
- Integrate Rowan with tODE, GLASS/GsDevKit, GsDevKit_home
- Review/Refactor for dialect portability