instantiations

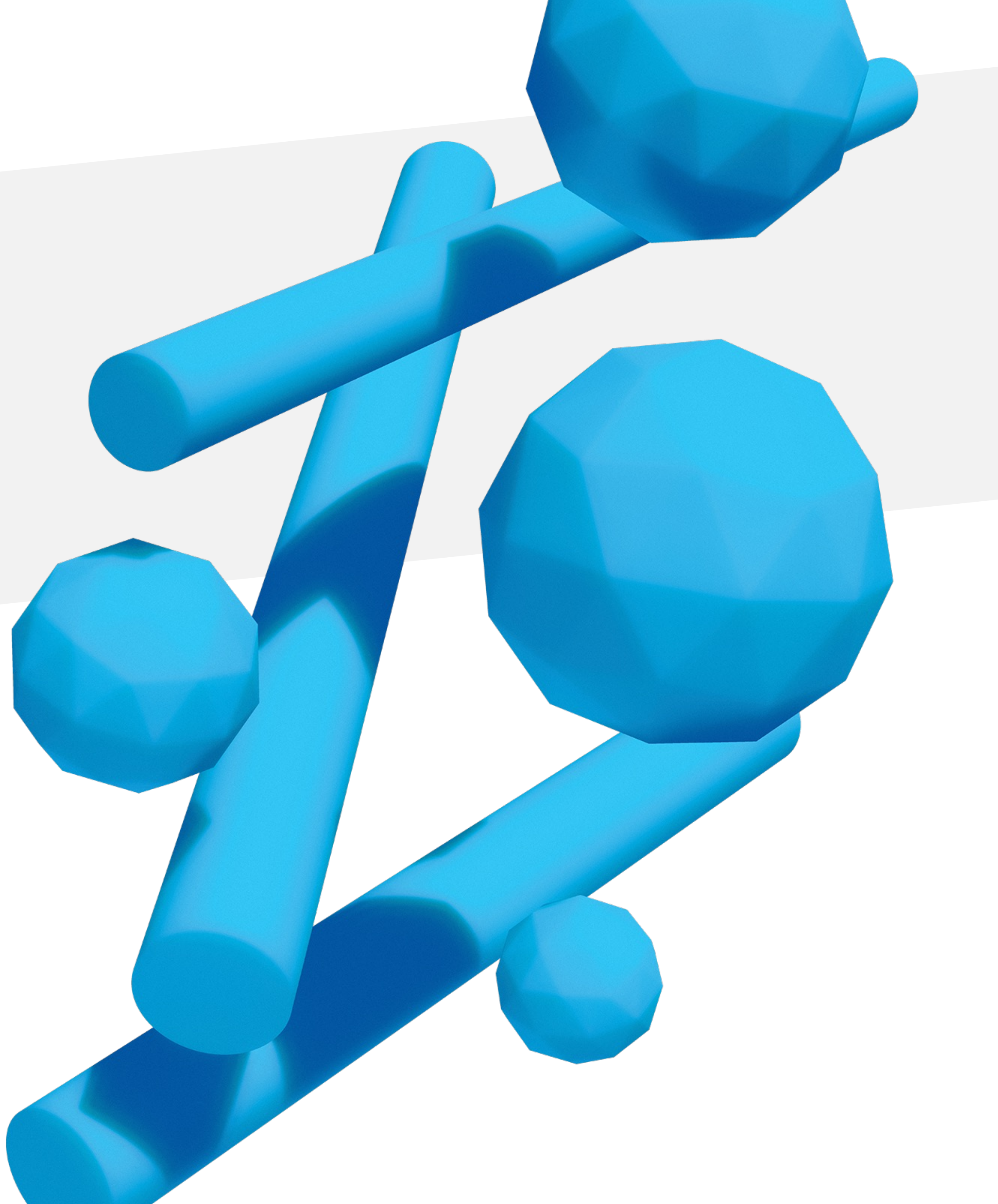
Behind the Scenes: The Making of VAST

Mariano Martinez Peck
Team Lead & Sr. Software Engineer

mpeck@instantiations.com

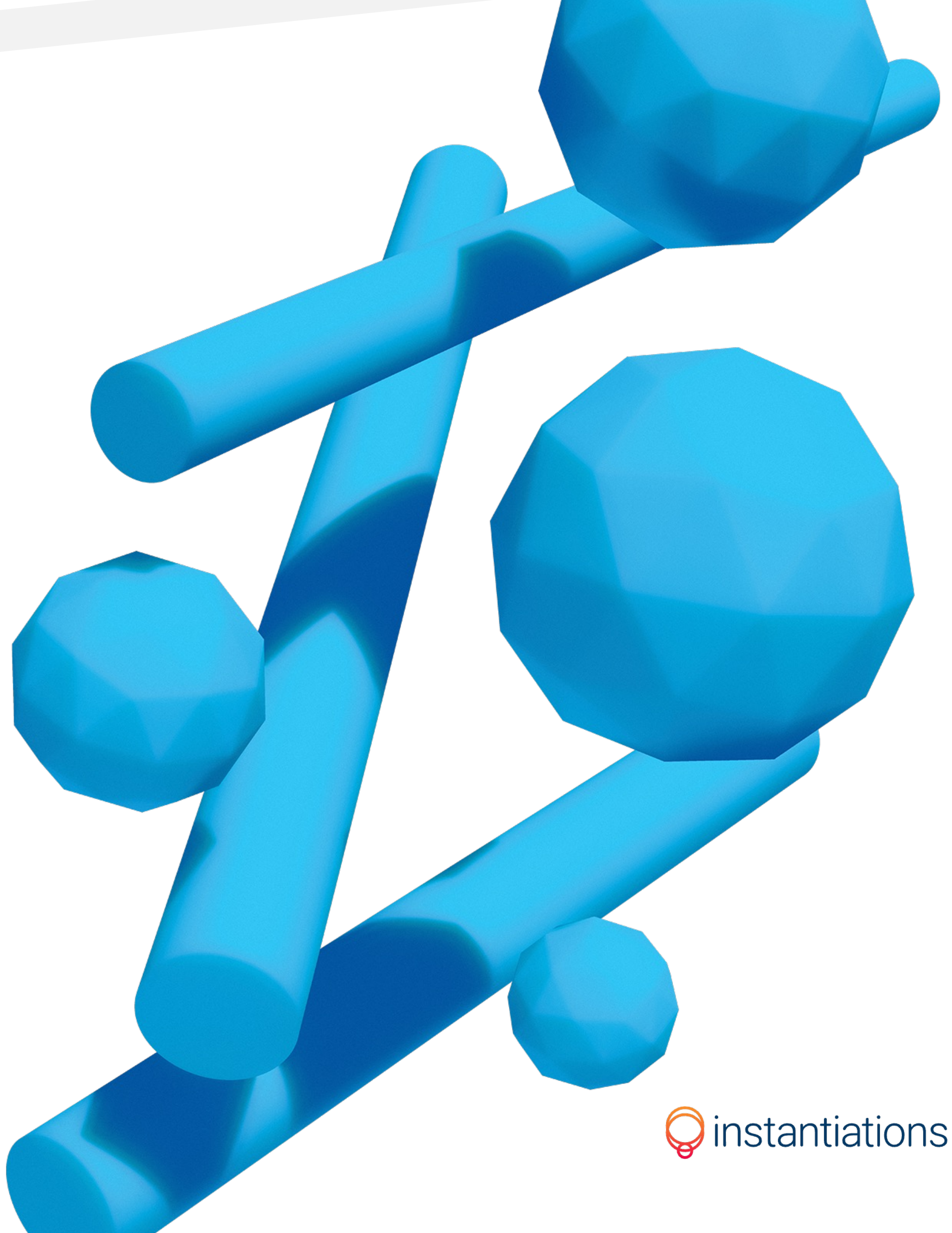
@MartinezPeck

/in/mariano-martinez-peck



Agenda

- VAST Platform Overview
- Coding
- Testing
- Building
- Documenting
- Conclusion
- Q&A



instantiations

VAST Platform Overview

instantiations

VAST Platform Components

- Smalltalk
- Virtual Machine (VM)
- Internal tests
- Many build scripts
- Installers
- Documentation
- Migration Guide
- Other supplementary materials…

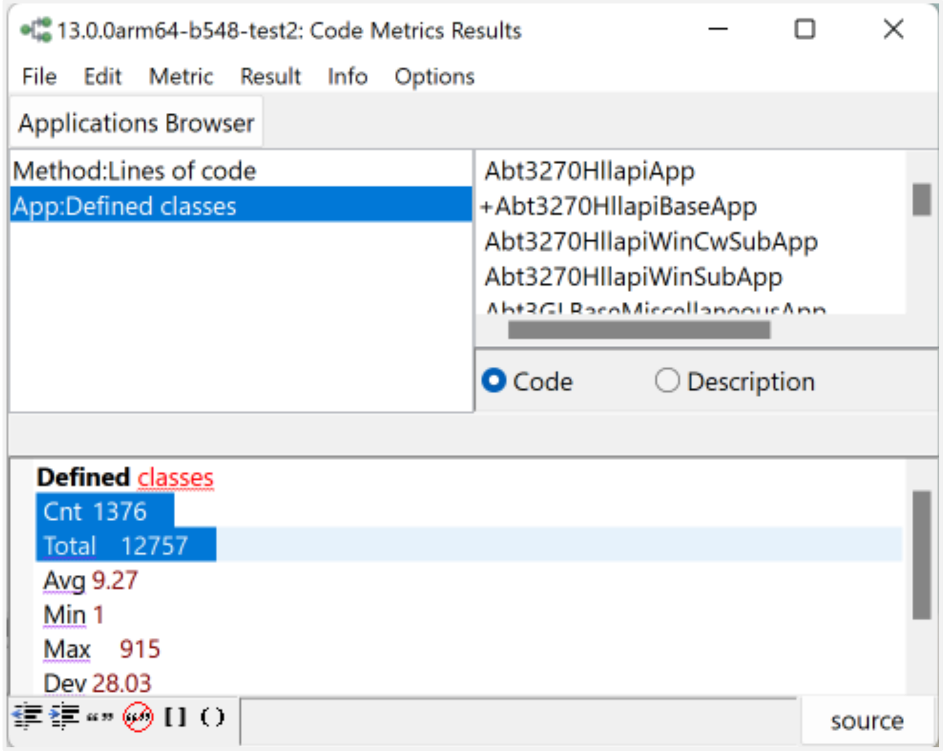
Smalltalk

- Base Smalltalk language implementation
- Additional libraries and frameworks
- Integrated development environment (IDE)
- Graphical user interface (GUI)
- Version control system
- Tools

instantiations

Smalltalk – Some Numbers

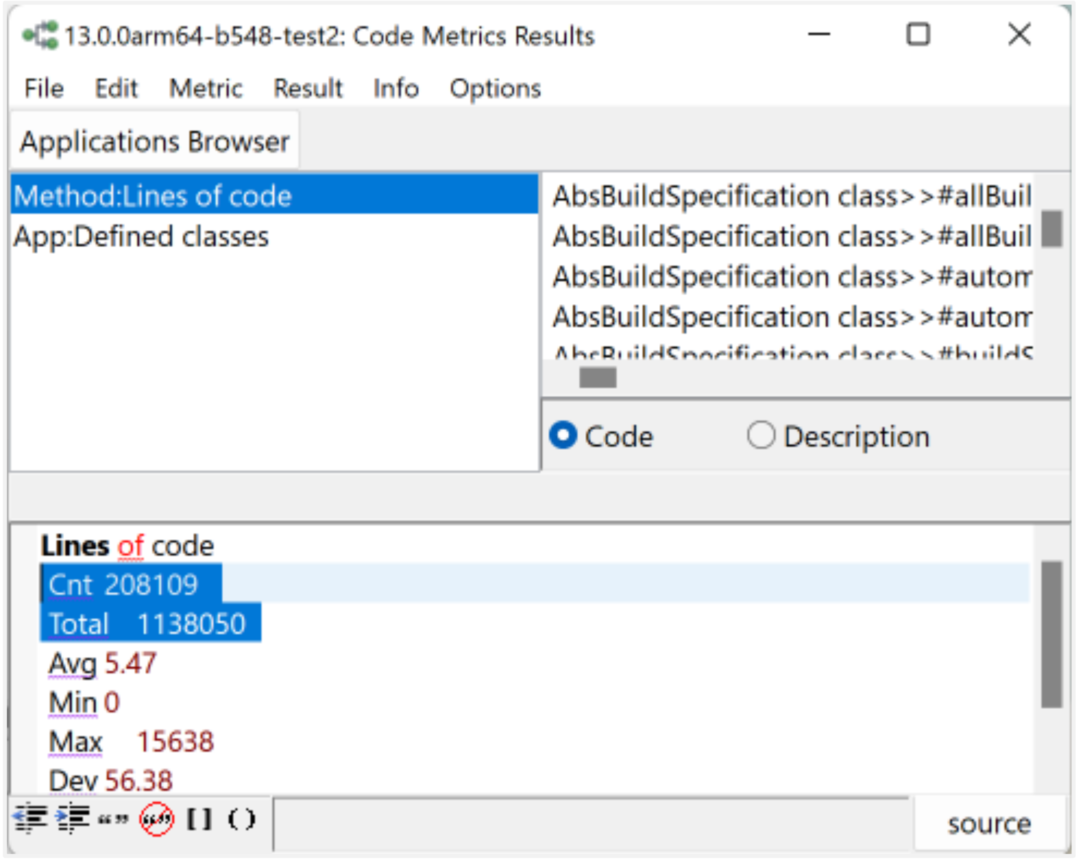
- Number of Applications ~= **1,400**
- Number of Classes ~= **13,000**
- Number of Methods ~= **213,000**
- Lines of Code ~= **1,150,000**


13.0.0arm64-b548-test2: Code Metrics Results
File Edit Metric Result Info Options
Applications Browser
Code
Description
source

Method:Lines of code
App:Defined classes

Abt3270HllapiApp
+Abt3270HllapiBaseApp
Abt3270HllapiWinCwSubApp
Abt3270HllapiWinSubApp

```
Defined classes
Cnt 1376
Total   12757
Avg 9.27
Min 1
Max    915
Dev 28.03
```


13.0.0arm64-b548-test2: Code Metrics Results
File Edit Metric Result Info Options
Applications Browser
Code
Description
source

Method:Lines of code
App:Defined classes

AbsBuildSpecification class>>#allBuil
AbsBuildSpecification class>>#allBuil
AbsBuildSpecification class>>#autom
AbsBuildSpecification class>>#autom

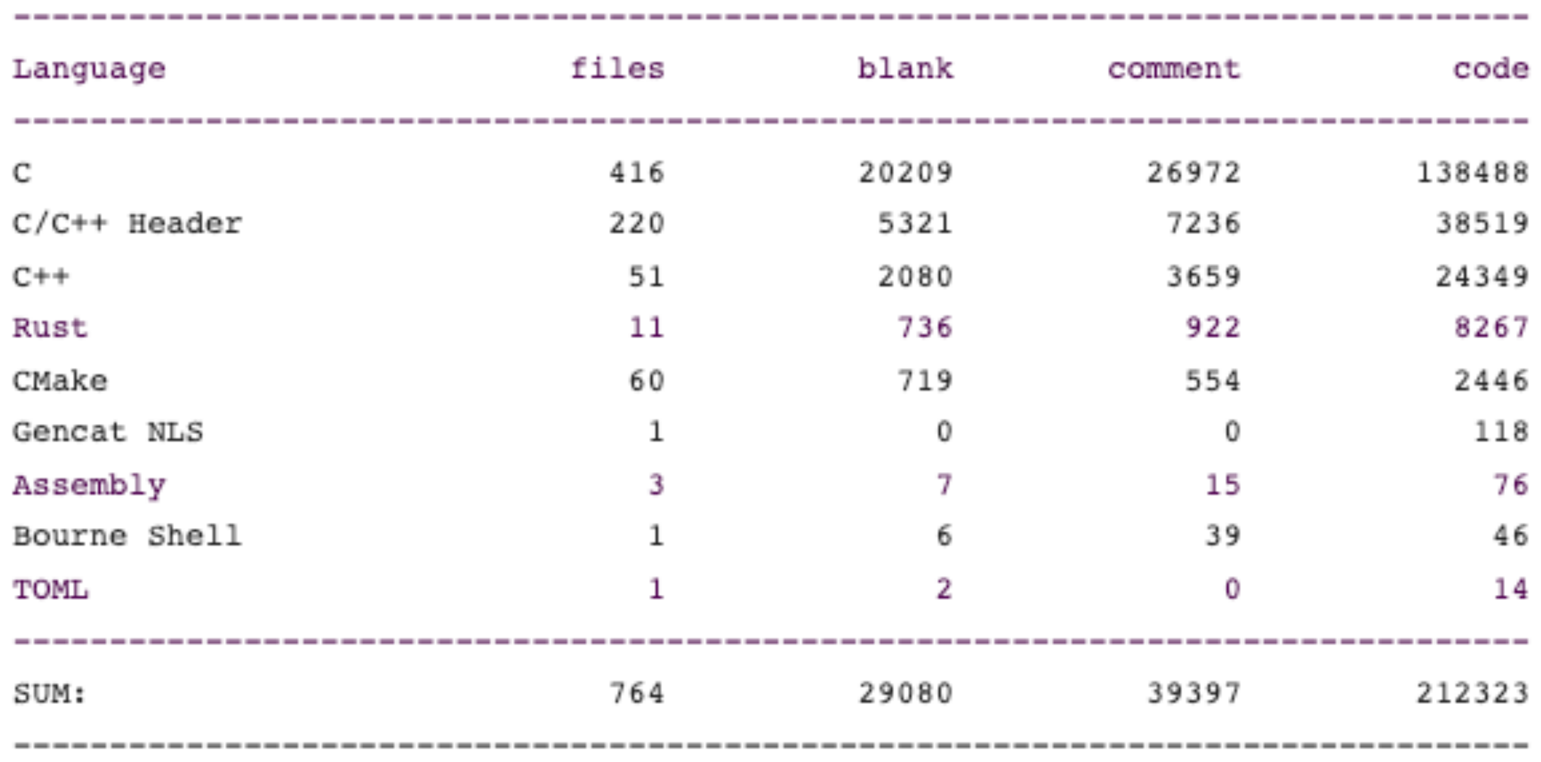
```
Lines of code
Cnt 208109
Total   1138050
Avg 5.47
Min 0
Max    15638
Dev 56.38
```

Virtual Machine (VM)

- Interpreter
- Memory manager & GC
- JIT and PIC
- Unicode support
- OsProcess
- And many other parts!

Virtual Machine (VM) – Some Numbers

- Lines of Code = **212,323**



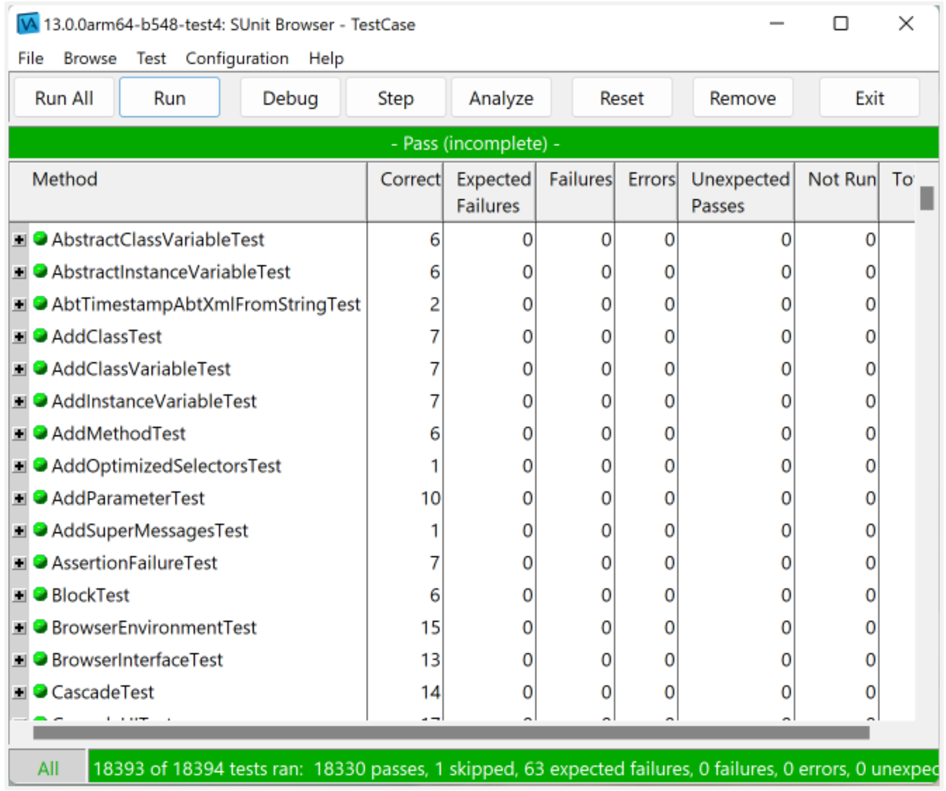
```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                              416          20209          26972         138488
C/C++ Header                   220           5321           7236          38519
C++                             51           2080           3659          24349
Rust                            11            736            922           8267
CMake                           60            719            554           2446
Gencat NLS                       1              0              0            118
Assembly                         3              7             15             76
Bourne Shell                     1              6             39             46
TOML                             1              2              0             14
-------------------------------------------------------------------------------
SUM:                           764          29080          39397         212323
-------------------------------------------------------------------------------
```

instantiations

Internal Tests – Some Numbers

- ~= **19,600** unit tests
- ~= **162,000** lines of code

…and even more tests I'll mention later!



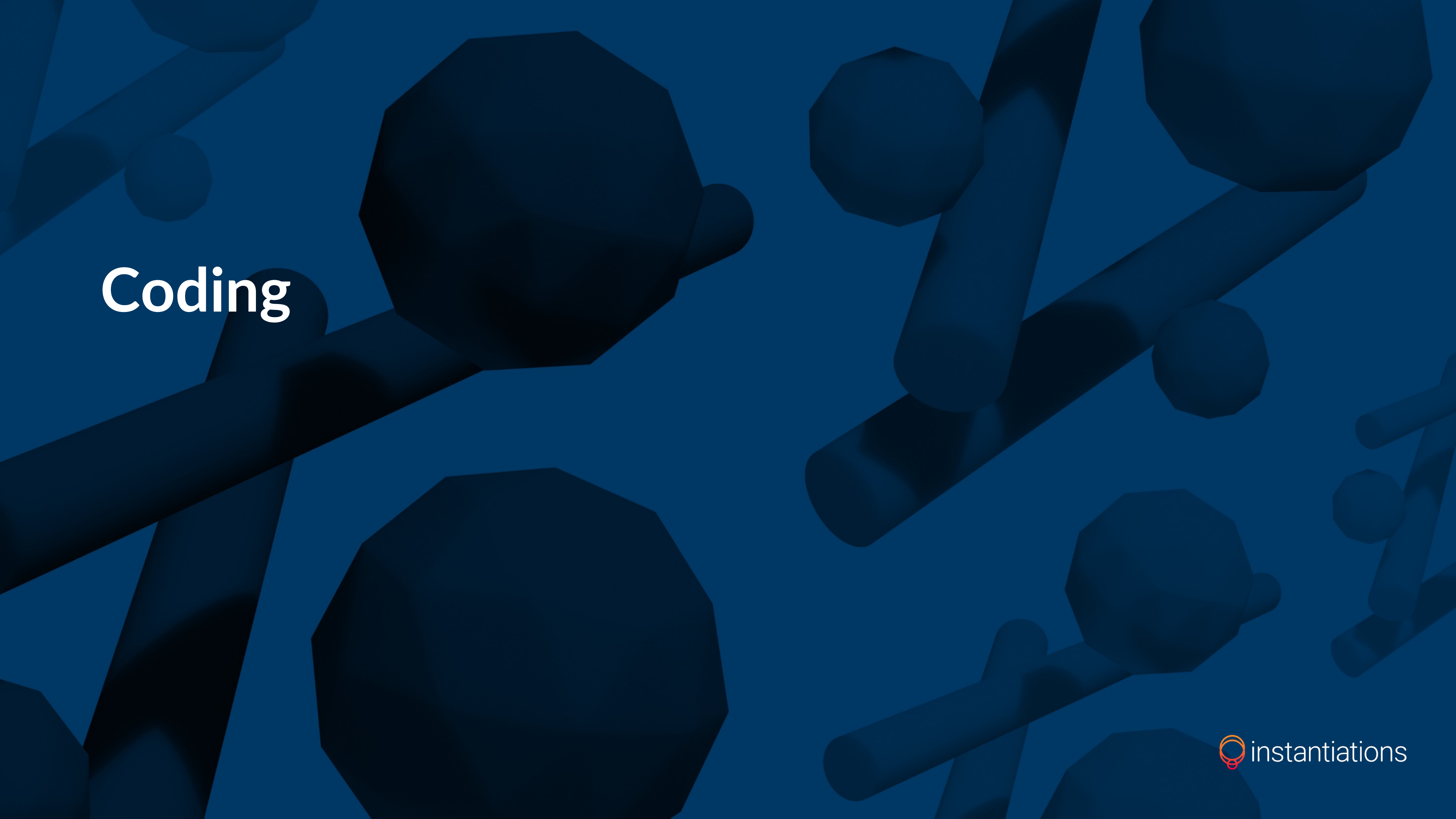
Method	Correct	Expected Failures	Failures	Errors	Unexpected Passes	Not Run	To
AbstractClassVariableTest	6	0	0	0	0	0	
AbstractInstanceVariableTest	6	0	0	0	0	0	
AbtTimestampAbtXmlFromStringTest	2	0	0	0	0	0	
AddClassTest	7	0	0	0	0	0	
AddClassVariableTest	7	0	0	0	0	0	
AddInstanceVariableTest	7	0	0	0	0	0	
AddMethodTest	6	0	0	0	0	0	
AddOptimizedSelectorsTest	1	0	0	0	0	0	
AddParameterTest	10	0	0	0	0	0	
AddSuperMessagesTest	1	0	0	0	0	0	
AssertionFailureTest	7	0	0	0	0	0	
BlockTest	6	0	0	0	0	0	
BrowserEnvironmentTest	15	0	0	0	0	0	
BrowserInterfaceTest	13	0	0	0	0	0	
CascadeTest	14	0	0	0	0	0	

18393 of 18394 tests ran: 18330 passes, 1 skipped, 63 expected failures, 0 failures, 0 errors, 0 unexpec

instantiations

1.5 million lines of code just for Smalltalk, the VM, and unit tests!

instantiations



Coding

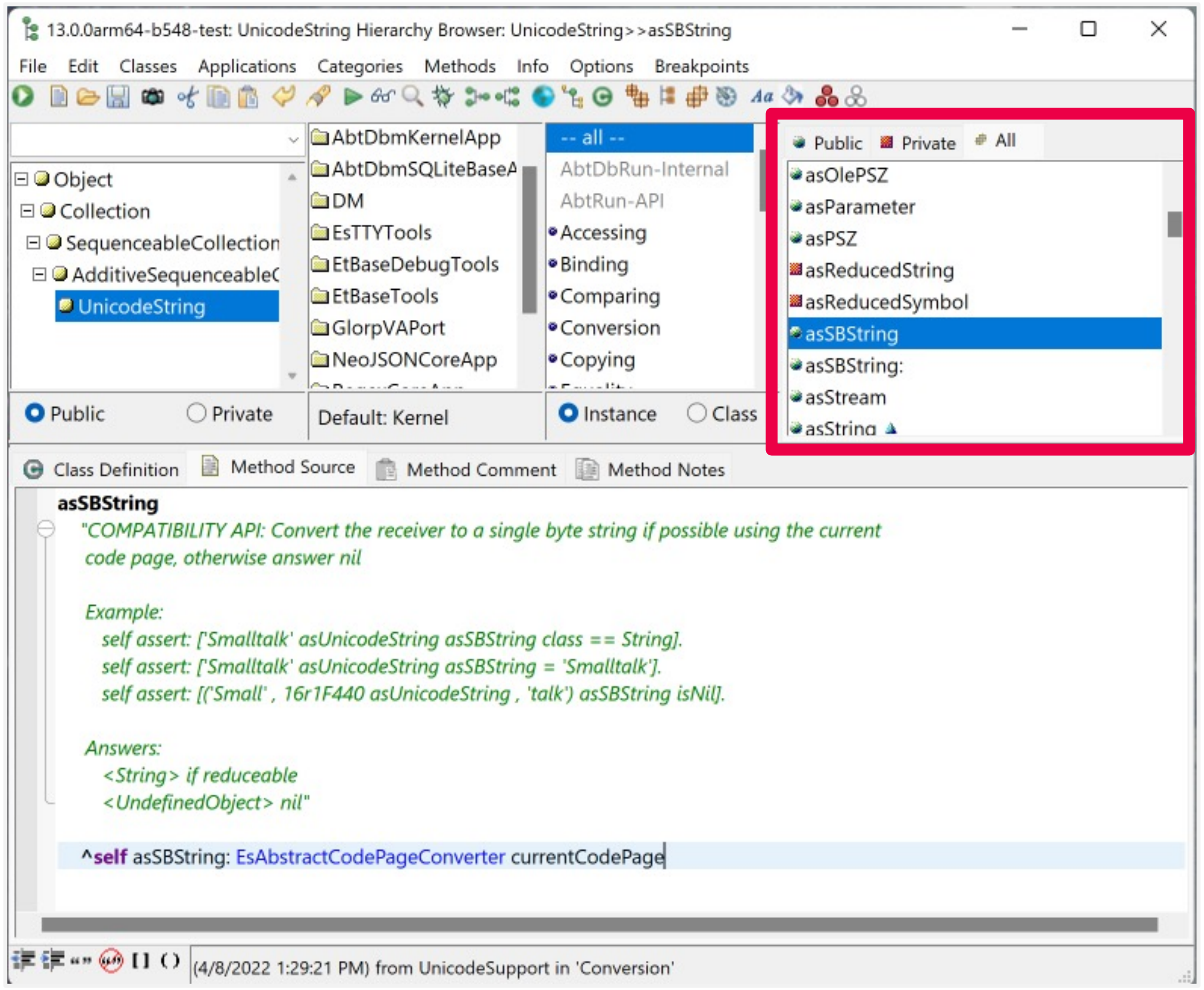
instantiations

Method Visibility: Private vs. Public

- Important consideration to create an easy upgrade path for customers. (Avoid or minimize "public" method changes.)

- When changes do occur to "public" methods, they are included in our Migration Guide.

- Balance changes to "public" methods between necessary evolution and business reality.



13.0.0arm64-b548-test: UnicodeString Hierarchy Browser: UnicodeString>>asSBString

File Edit Classes Applications Categories Methods Info Options Breakpoints

- Object
  - Collection
    - SequenceableCollection
      - AdditiveSequenceableC
        - UnicodeString

AbtDbmKernelApp, AbtDbmSQLiteBaseA, DM, EsTTYTools, EtBaseDebugTools, EtBaseTools, GlorpVAPort, NeoJSONCoreApp

-- all --, AbtDbRun-Internal, AbtRun-API, Accessing, Binding, Comparing, Conversion, Copying

Public | Private | All

asOlePSZ, asParameter, asPSZ, asReducedString, asReducedSymbol, asSBString, asSBString:, asStream, asString

[✓ Public] Private | Default: Kernel | [✓ Instance] Class

Class Definition | Method Source | Method Comment | Method Notes

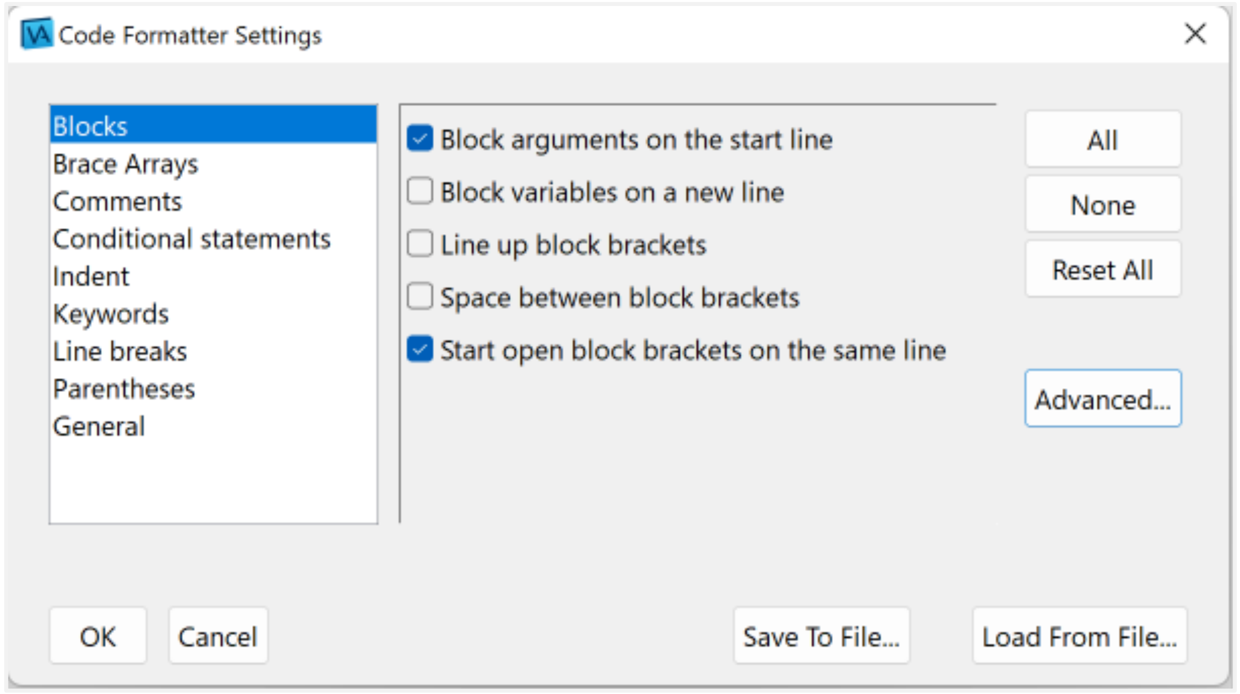
```
asSBString
	"COMPATIBILITY API: Convert the receiver to a single byte string if possible using the current
	 code page, otherwise answer nil

	 Example:
	   self assert: ['Smalltalk' asUnicodeString asSBString class == String].
	   self assert: ['Smalltalk' asUnicodeString asSBString = 'Smalltalk'].
	   self assert: [('Small' , 16r1F440 asUnicodeString , 'talk') asSBString isNil].

	 Answers:
	   <String> if reduceable
	   <UndefinedObject> nil"

	^self asSBString: EsAbstractCodePageConverter currentCodePage
```

(4/8/2022 1:29:21 PM) from UnicodeSupport in 'Conversion'

instantiations

Code Formatting

- Unified formatting layout
- We stick to the default format
- Be able to compare each others code and see not formatting difference


Code Formatter Settings

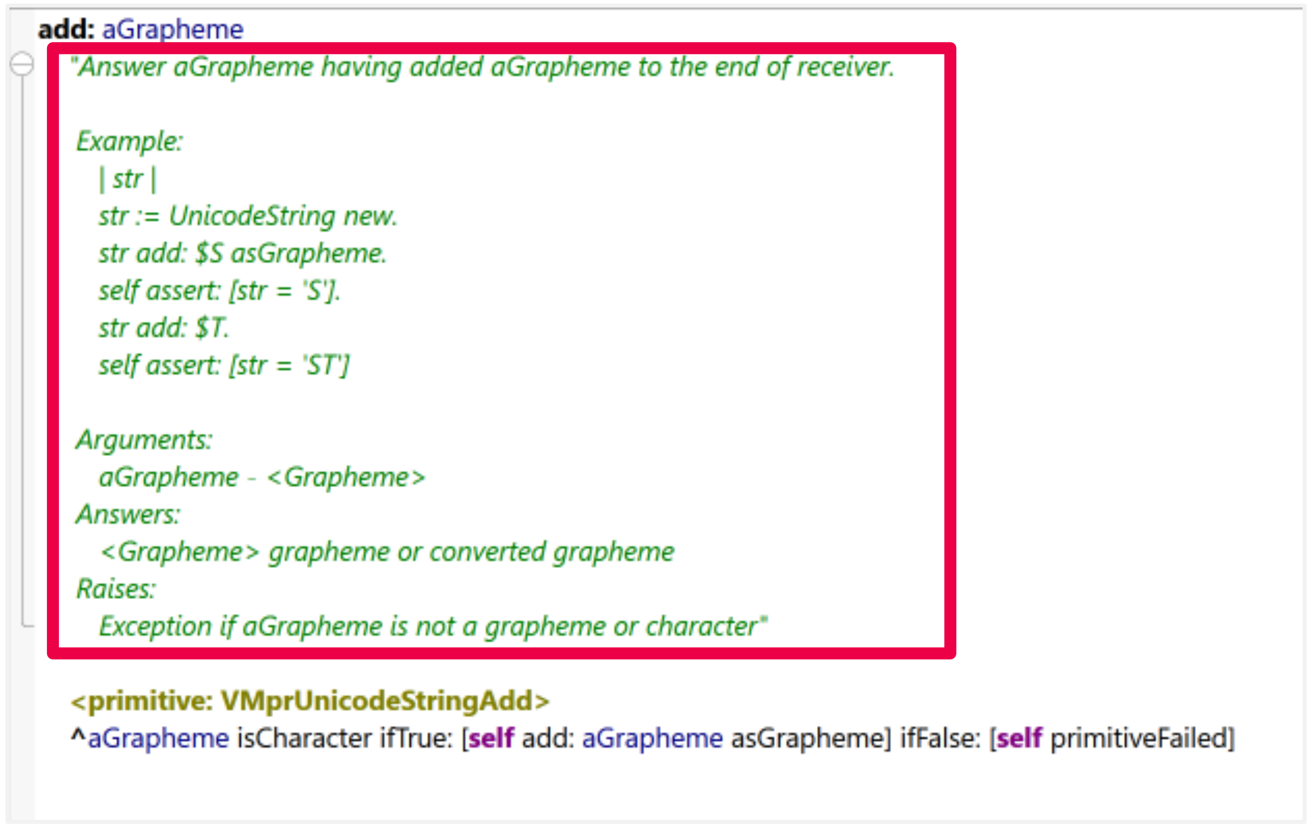
Blocks
Brace Arrays
Comments
Conditional statements
Indent
Keywords
Line breaks
Parentheses
General

- [✓] Block arguments on the start line
- [ ] Block variables on a new line
- [ ] Line up block brackets
- [ ] Space between block brackets
- [✓] Start open block brackets on the same line

Method Comments

- At least, for public methods
- They include a brief description, arguments, answer, exception raised, examples, etc.

Bonus:
Create tests that run the examples located in the comments!



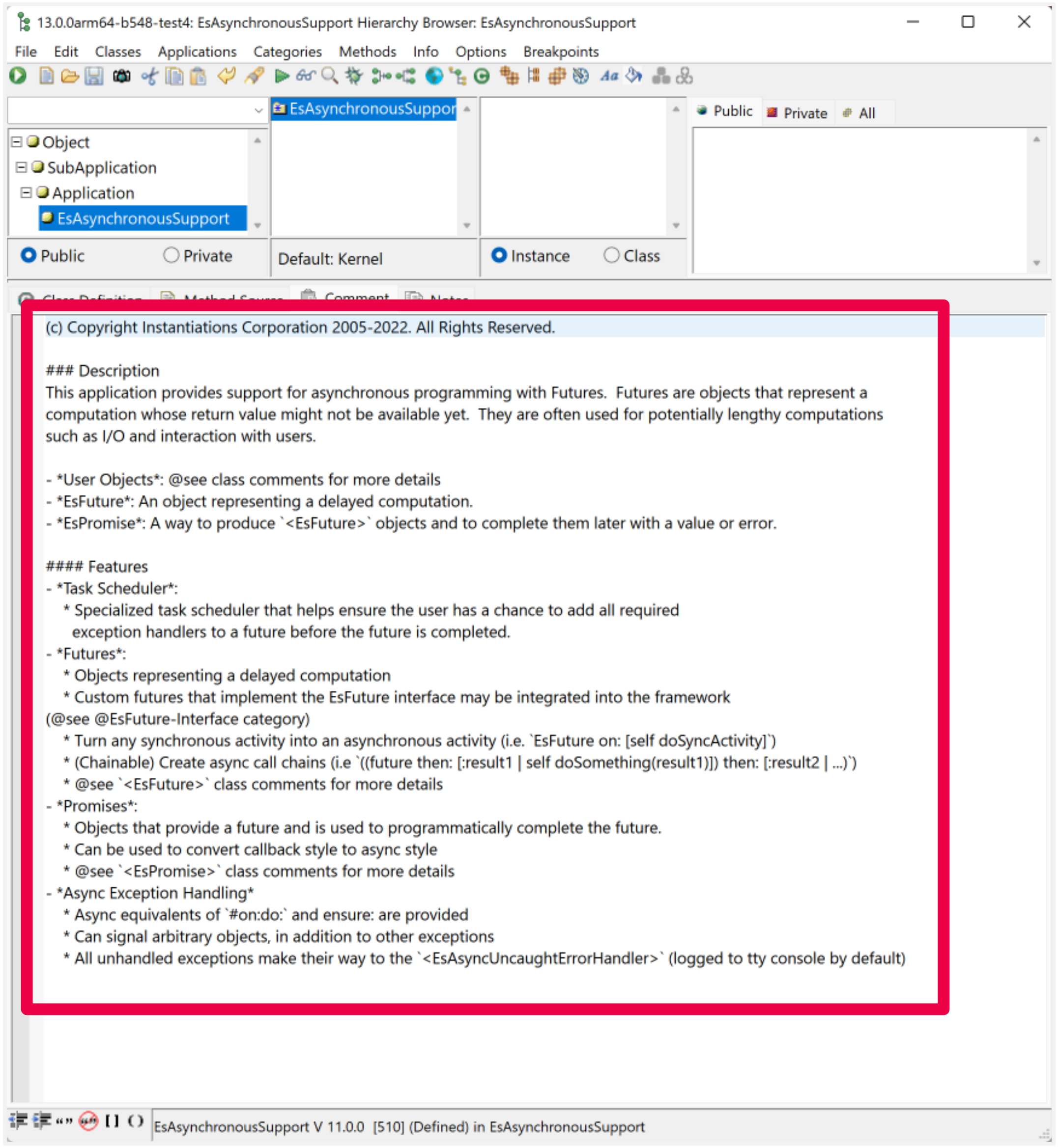
```
add: aGrapheme
    "Answer aGrapheme having added aGrapheme to the end of receiver.

    Example:
        | str |
        str := UnicodeString new.
        str add: $S asGrapheme.
        self assert: [str = 'S'].
        str add: $T.
        self assert: [str = 'ST']

    Arguments:
        aGrapheme - <Grapheme>
    Answers:
        <Grapheme> grapheme or converted grapheme
    Raises:
        Exception if aGrapheme is not a grapheme or character"

    <primitive: VMprUnicodeStringAdd>
    ^aGrapheme isCharacter ifTrue: [self add: aGrapheme asGrapheme] ifFalse: [self primitiveFailed]
```

instantiations

App/Class Comments

- Comments in methods are great, but its hard to give the bigger picture.
- Application comments give a sense of the cohesive properties that led all the classes to be grouped together.
- Class comments should describe the object's purpose and any other interesting details.

Bonus:
Create tests that run the examples located in the comments!


13.0.0arm64-b548-test4: EsAsynchronousSupport Hierarchy Browser: EsAsynchronousSupport
File Edit Classes Applications Categories Methods Info Options Breakpoints
Object
SubApplication
Application
EsAsynchronousSupport
EsAsynchronousSuppor
Public Private All
Class Definition Method Source Comment Notes

Public [✓] Private
Default: Kernel
Instance [✓] Class

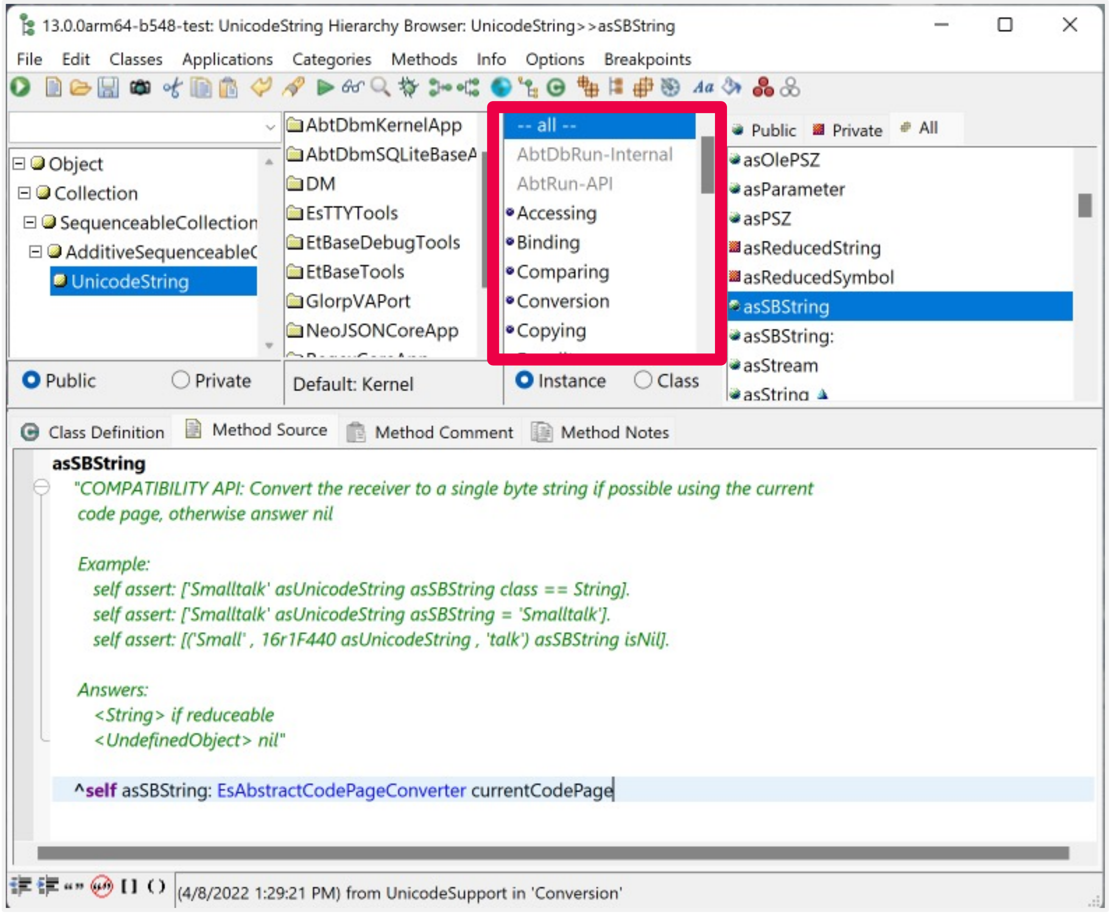
```
(c) Copyright Instantiations Corporation 2005-2022. All Rights Reserved.

### Description
This application provides support for asynchronous programming with Futures.  Futures are objects that represent a
computation whose return value might not be available yet.  They are often used for potentially lengthy computations
such as I/O and interaction with users.

- *User Objects*: @see class comments for more details
- *EsFuture*: An object representing a delayed computation.
- *EsPromise*: A way to produce `<EsFuture>` objects and to complete them later with a value or error.

#### Features
- *Task Scheduler*:
   * Specialized task scheduler that helps ensure the user has a chance to add all required
    exception handlers to a future before the future is completed.
- *Futures*:
   * Objects representing a delayed computation
   * Custom futures that implement the EsFuture interface may be integrated into the framework
(@see @EsFuture-Interface category)
   * Turn any synchronous activity into an asynchronous activity (i.e. `EsFuture on: [self doSyncActivity]`)
   * (Chainable) Create async call chains (i.e `((future then: [:result1 | self doSomething(result1)]) then: [:result2 | ...)`)
   * @see `<EsFuture>` class comments for more details
- *Promises*:
   * Objects that provide a future and is used to programmatically complete the future.
   * Can be used to convert callback style to async style
   * @see `<EsPromise>` class comments for more details
- *Async Exception Handling*
   * Async equivalents of `#on:do:` and ensure: are provided
   * Can signal arbitrary objects, in addition to other exceptions
   * All unhandled exceptions make their way to the `<EsAsyncUncaughtErrorHandler>` (logged to tty console by default)
```

EsAsynchronousSupport V 11.0.0 [510] (Defined) in EsAsynchronousSupport

instantiations

Method Categories/Protocols

- Every method belongs to a category
- Methods can be in more than one category

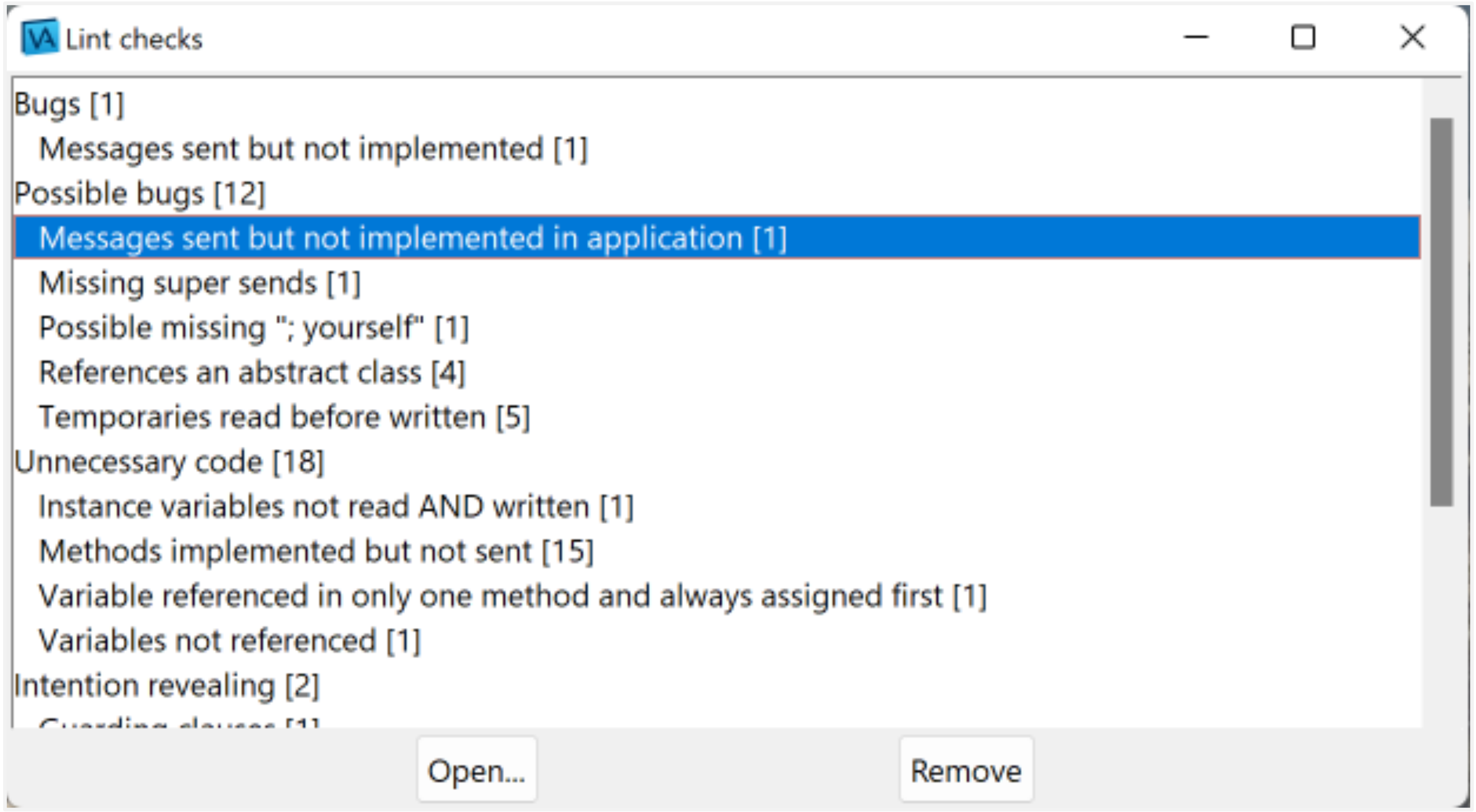


```
asSBString
    "COMPATIBILITY API: Convert the receiver to a single byte string if possible using the current
    code page, otherwise answer nil

    Example:
      self assert: ['Smalltalk' asUnicodeString asSBString class == String].
      self assert: ['Smalltalk' asUnicodeString asSBString = 'Smalltalk'].
      self assert: [('Small' , 16r1F440 asUnicodeString , 'talk') asSBString isNil].

    Answers:
      <String> if reduceable
      <UndefinedObject> nil"

    ^self asSBString: EsAbstractCodePageConverter currentCodePage
```

(4/8/2022 1:29:21 PM) from UnicodeSupport in 'Conversion'

Lint Checking

- Lint rules are run against the code
- Improve code quality

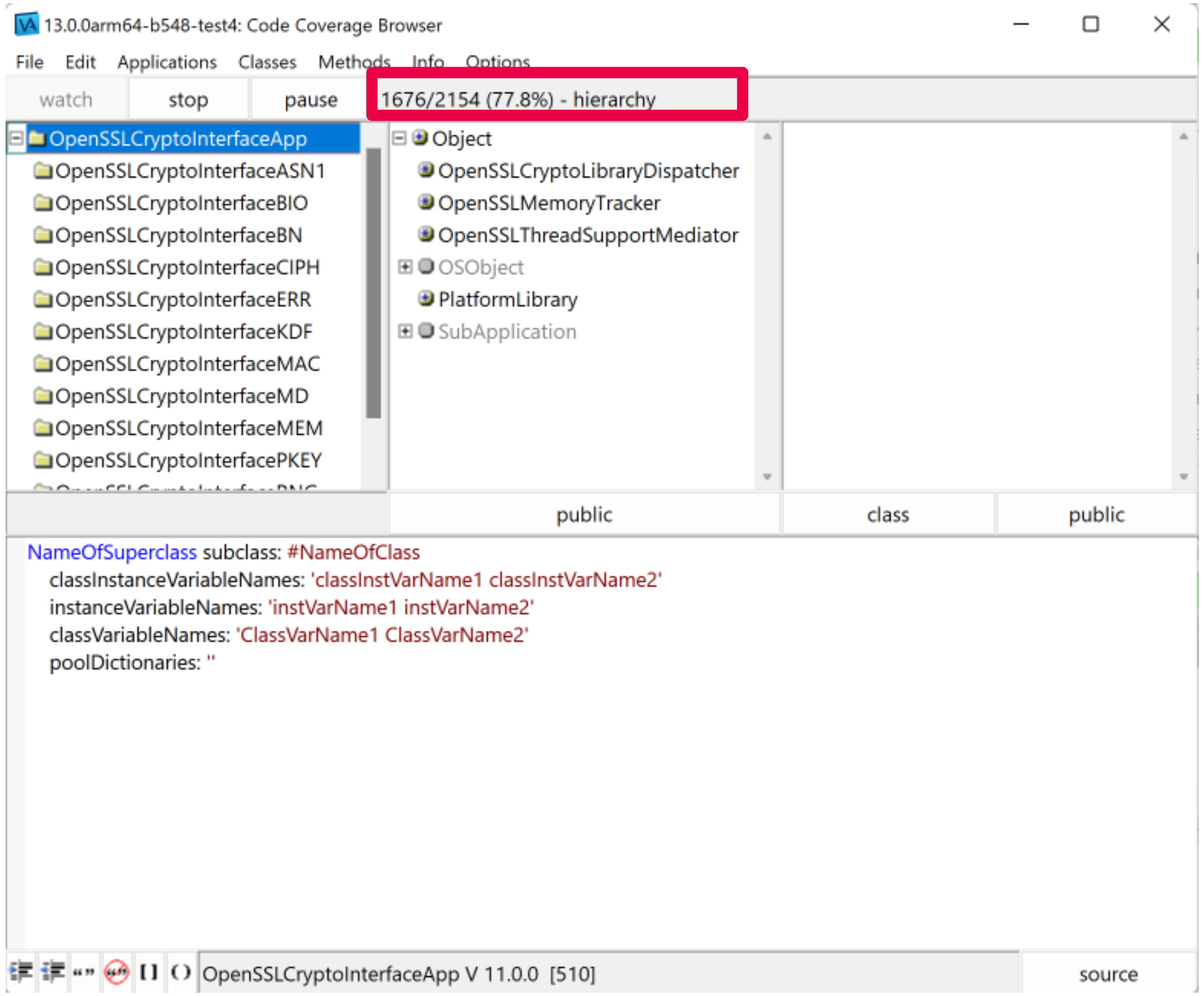


Bugs [1]
  Messages sent but not implemented [1]
Possible bugs [12]
  Messages sent but not implemented in application [1]
  Missing super sends [1]
  Possible missing "; yourself" [1]
  References an abstract class [4]
  Temporaries read before written [5]
Unnecessary code [18]
  Instance variables not read AND written [1]
  Methods implemented but not sent [15]
  Variable referenced in only one method and always assigned first [1]
  Variables not referenced [1]
Intention revealing [2]

instantiations

Coverage Analysis

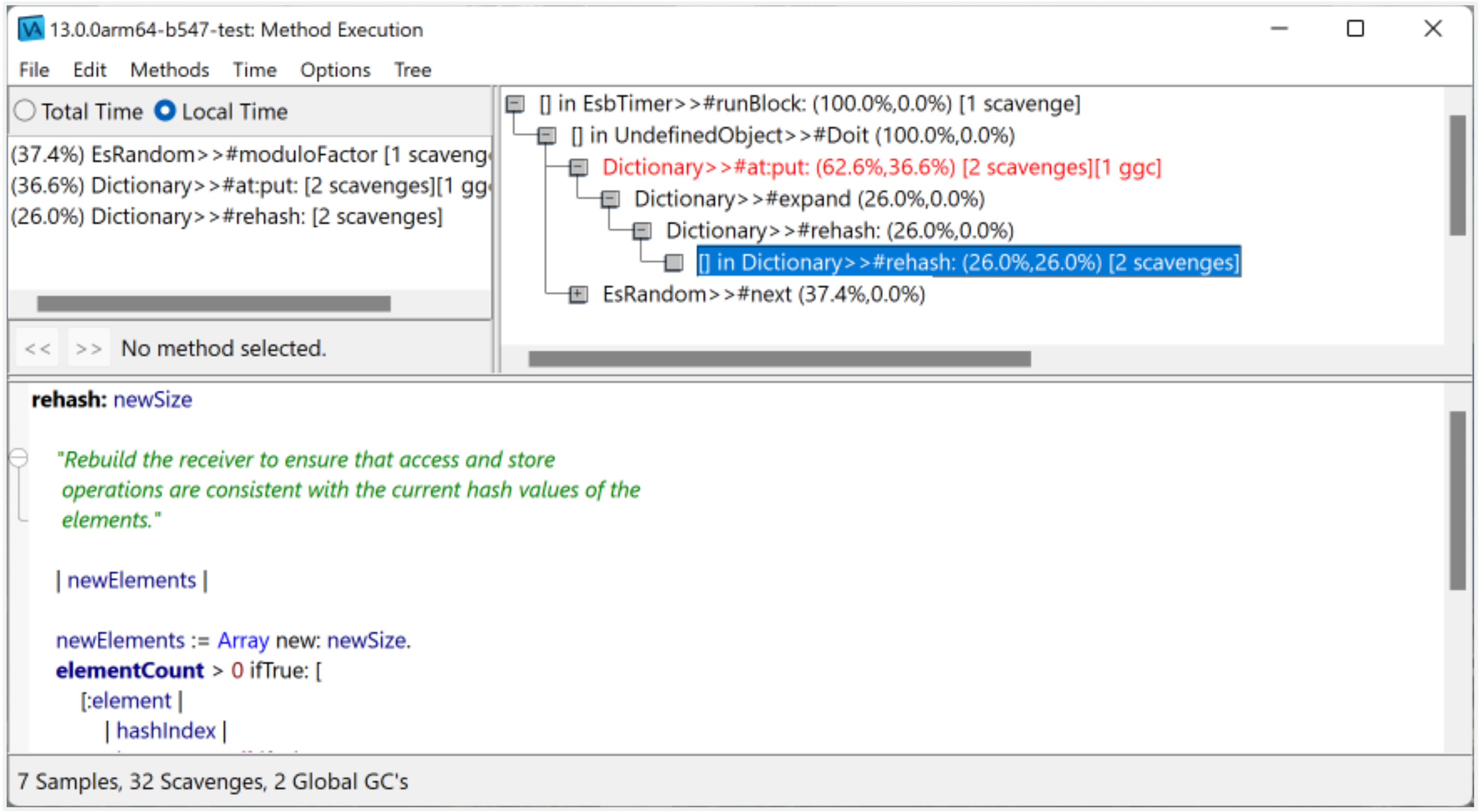
- We use the coverage analysis tool.
- Builds confidence that you have developed working code.
- Gives a possible measure of how effective your test suite is.
- If code didn't get exercised…
  it should be assumed that it's broken!


13.0.0arm64-b548-test4: Code Coverage Browser
File
Edit
Applications
Classes
Methods
Info
Options
watch
stop
pause
1676/2154 (77.8%) - hierarchy
OpenSSLCryptoInterfaceApp
OpenSSLCryptoInterfaceASN1
OpenSSLCryptoInterfaceBIO
OpenSSLCryptoInterfaceBN
OpenSSLCryptoInterfaceCIPH
OpenSSLCryptoInterfaceERR
OpenSSLCryptoInterfaceKDF
OpenSSLCryptoInterfaceMAC
OpenSSLCryptoInterfaceMD
OpenSSLCryptoInterfaceMEM
OpenSSLCryptoInterfacePKEY
Object
OpenSSLCryptoLibraryDispatcher
OpenSSLMemoryTracker
OpenSSLThreadSupportMediator
OSObject
PlatformLibrary
SubApplication
public
class
public
OpenSSLCryptoInterfaceApp V 11.0.0 [510]
source

```
NameOfSuperclass subclass: #NameOfClass
    classInstanceVariableNames: 'classInstVarName1 classInstVarName2'
    instanceVariableNames: 'instVarName1 instVarName2'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
```

Performance Profiling

- We use the Performance profiler
- VAST should be as efficient as possible


13.0.0arm64-b547-test: Method Execution
File Edit Methods Time Options Tree
Total Time [✓ Local Time]
<< >> No method selected.

```
(37.4%) EsRandom>>#moduloFactor [1 scaveng
(36.6%) Dictionary>>#at:put: [2 scavenges][1 gg
(26.0%) Dictionary>>#rehash: [2 scavenges]
```

```
[] in EsbTimer>>#runBlock: (100.0%,0.0%) [1 scavenge]
  [] in UndefinedObject>>#Doit (100.0%,0.0%)
    Dictionary>>#at:put: (62.6%,36.6%) [2 scavenges][1 ggc]
      Dictionary>>#expand (26.0%,0.0%)
        Dictionary>>#rehash: (26.0%,0.0%)
          [] in Dictionary>>#rehash: (26.0%,26.0%) [2 scavenges]
    EsRandom>>#next (37.4%,0.0%)
```

```
rehash: newSize

  "Rebuild the receiver to ensure that access and store
   operations are consistent with the current hash values of the
   elements."

  | newElements |

  newElements := Array new: newSize.
  elementCount > 0 ifTrue: [
    [:element |
      | hashIndex |
```

7 Samples, 32 Scavenges, 2 Global GC's

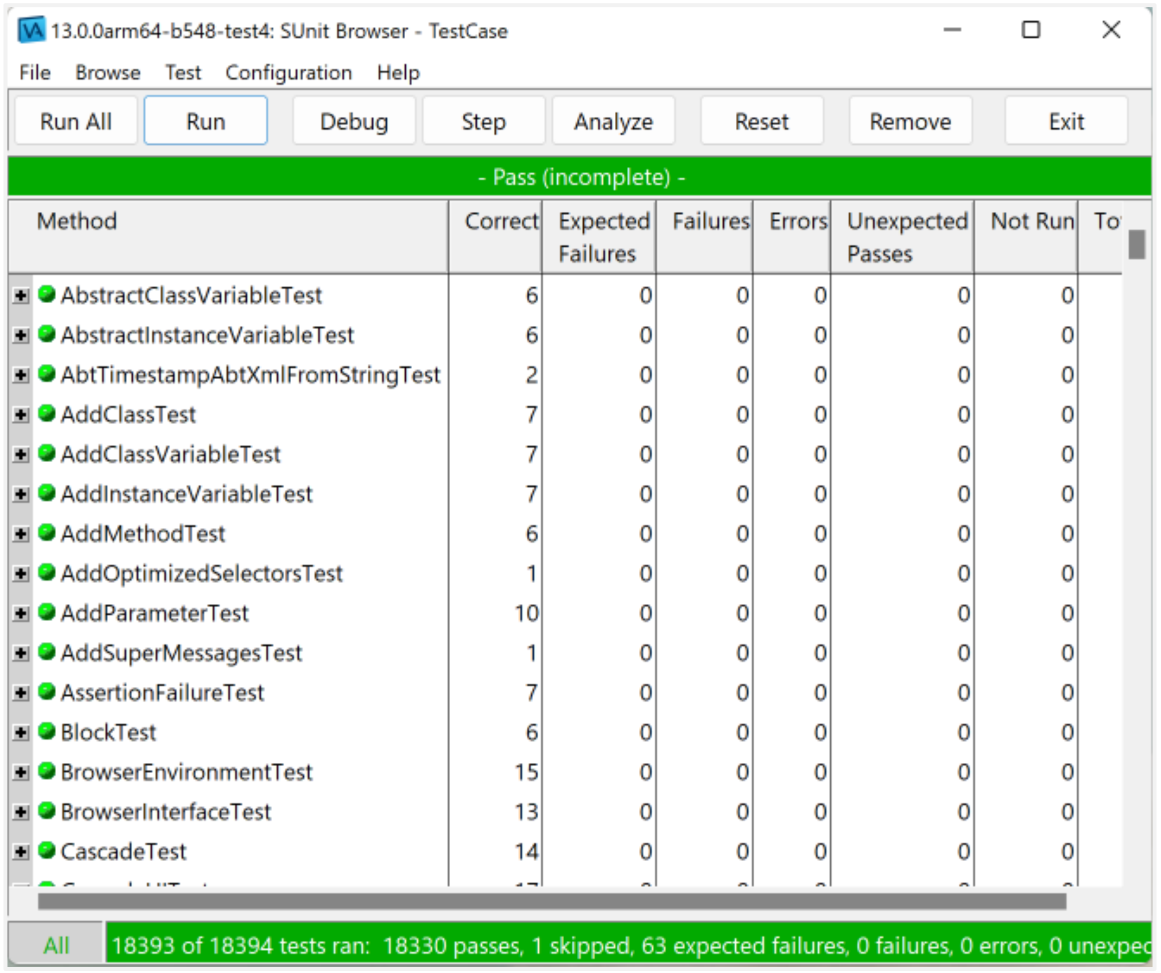
instantiations

Testing

instantiations

Multiple Types of Tests

- Extensive Smalltalk SUnit test suite
- VM tests
- IVT (Installation Verification Tests)
- Manual/regression tests


13.0.0arm64-b548-test4: SUnit Browser - TestCase
File Browse Test Configuration Help
Run All
Run
Debug
Step
Analyze
Reset
Remove
Exit
- Pass (incomplete) -

Method	Correct	Expected Failures	Failures	Errors	Unexpected Passes	Not Run	To
AbstractClassVariableTest	6	0	0	0	0	0	
AbstractInstanceVariableTest	6	0	0	0	0	0	
AbtTimestampAbtXmlFromStringTest	2	0	0	0	0	0	
AddClassTest	7	0	0	0	0	0	
AddClassVariableTest	7	0	0	0	0	0	
AddInstanceVariableTest	7	0	0	0	0	0	
AddMethodTest	6	0	0	0	0	0	
AddOptimizedSelectorsTest	1	0	0	0	0	0	
AddParameterTest	10	0	0	0	0	0	
AddSuperMessagesTest	1	0	0	0	0	0	
AssertionFailureTest	7	0	0	0	0	0	
BlockTest	6	0	0	0	0	0	
BrowserEnvironmentTest	15	0	0	0	0	0	
BrowserInterfaceTest	13	0	0	0	0	0	
CascadeTest	14	0	0	0	0	0	

All 18393 of 18394 tests ran: 18330 passes, 1 skipped, 63 expected failures, 0 failures, 0 errors, 0 unexpec

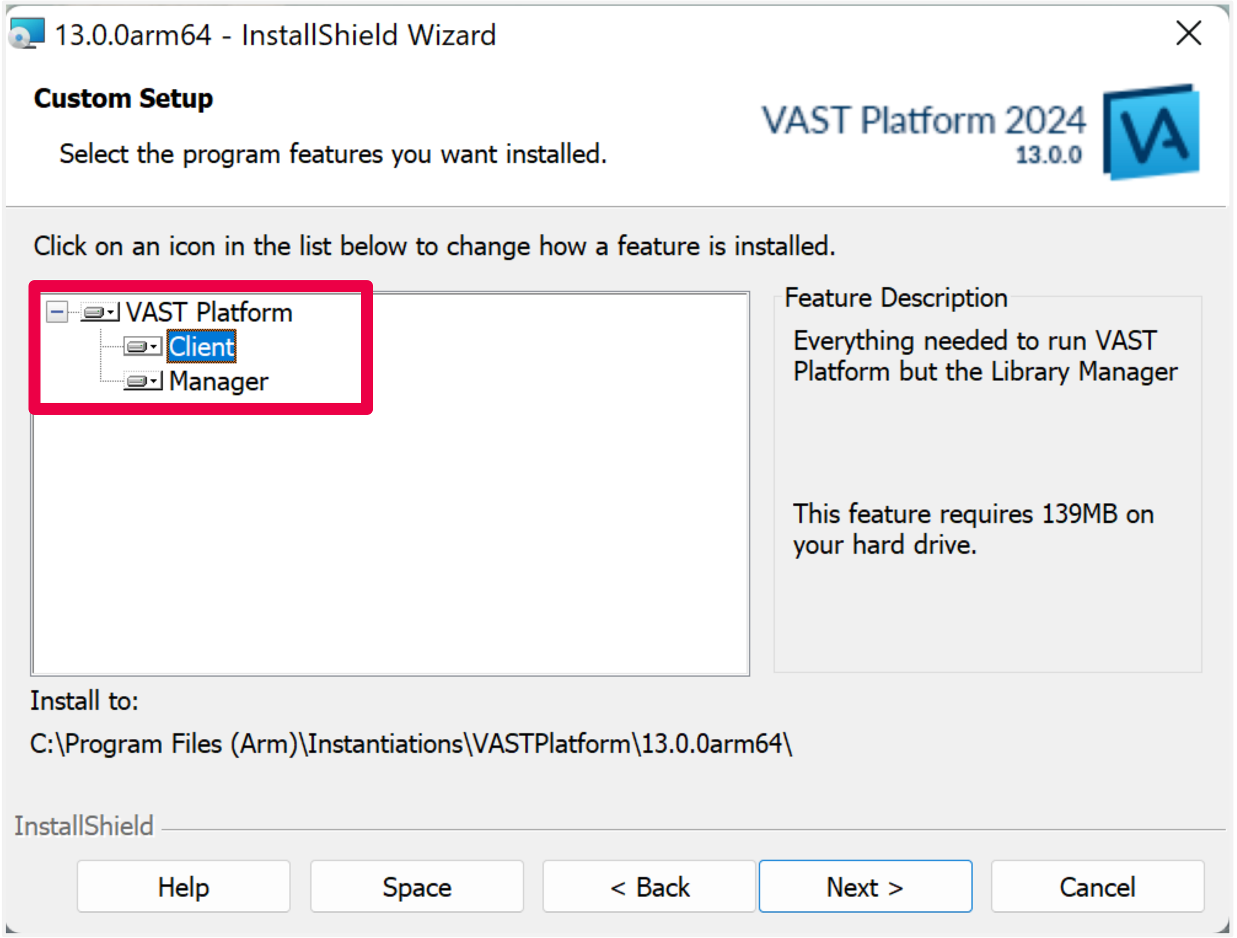
Multiple Platforms

- Operating Systems
  - Windows & Linux
- CPU Architectures
  - Intel x86, Intel x64, ARMv7 (32bit) and ARMv8 (64bit)
- Screen Depth
  - HiDPI vs Non HiDPI
  - Multiple scaling factors
- Linux Variations
  - Installers: deb, rpm
  - Types: desktop, server
  - GUI: KDE, Gnome, etc.
- Windows Variations
  - desktop, server

instantiations

Multiple VAST Installers Options

- Client
- Manager
- Standalone


13.0.0arm64 - InstallShield Wizard

Custom Setup

Select the program features you want installed.

VAST Platform 2024
13.0.0

Click on an icon in the list below to change how a feature is installed.

- VAST Platform
  - Client
  - Manager

Feature Description

Everything needed to run VAST Platform but the Library Manager

This feature requires 139MB on your hard drive.

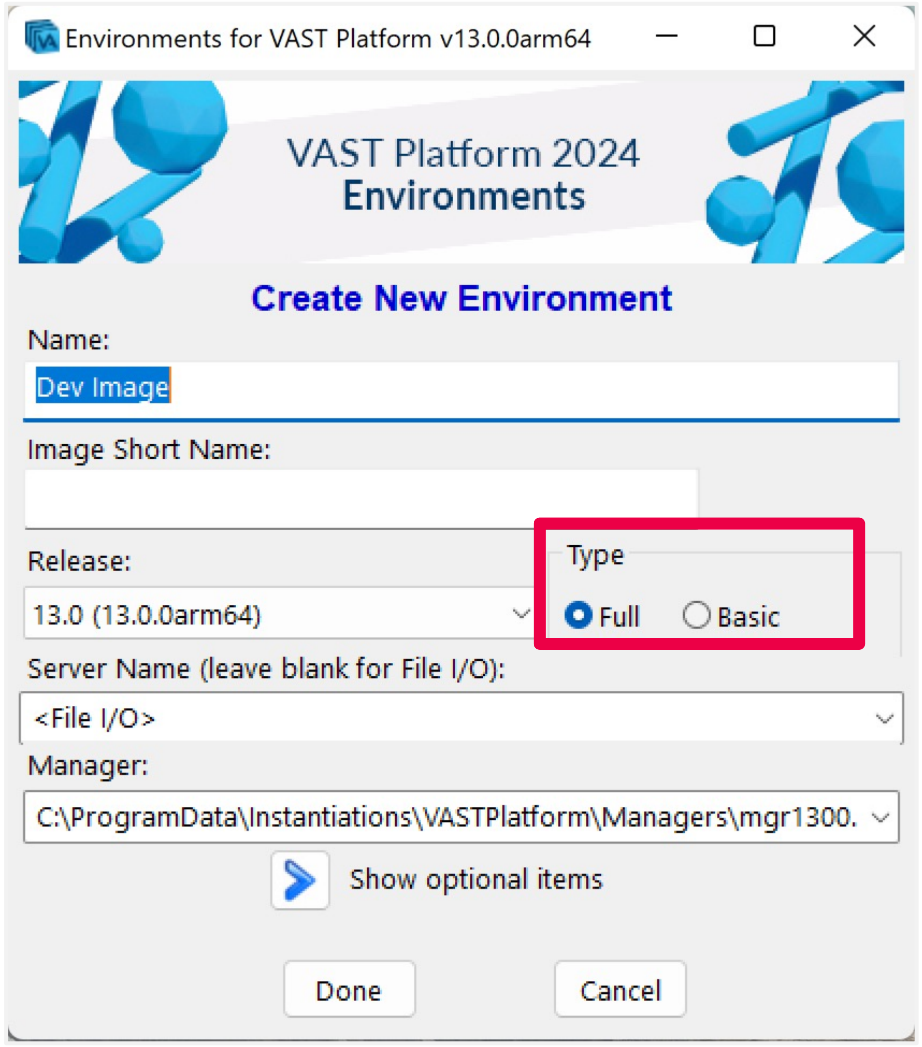
Install to:
C:\Program Files (Arm)\Instantiations\VASTPlatform\13.0.0arm64\

InstallShield

instantiations

Multiple VAST Images Types

- Full image
- Base image



Name: Dev Image

Image Short Name:

Release: 13.0 (13.0.0arm64)

Type: [✓ Full] Basic

Server Name (leave blank for File I/O): <File I/O>

Manager: C:\ProgramData\Instantiations\VASTPlatform\Managers\mgr1300.

instantiations

So Many Testing Combinations!

- Multiple Types of Tests
- Multiple Platforms
- Multiple VAST Installer Options
- Multiple VAST Image Types

Building

instantiations

Making a VAST Build

- Building
  - All the Smalltalk images
  - VAST Installers
  - Runtime support files

- Automated Testing
  - IVT tests
  - SUnit tests

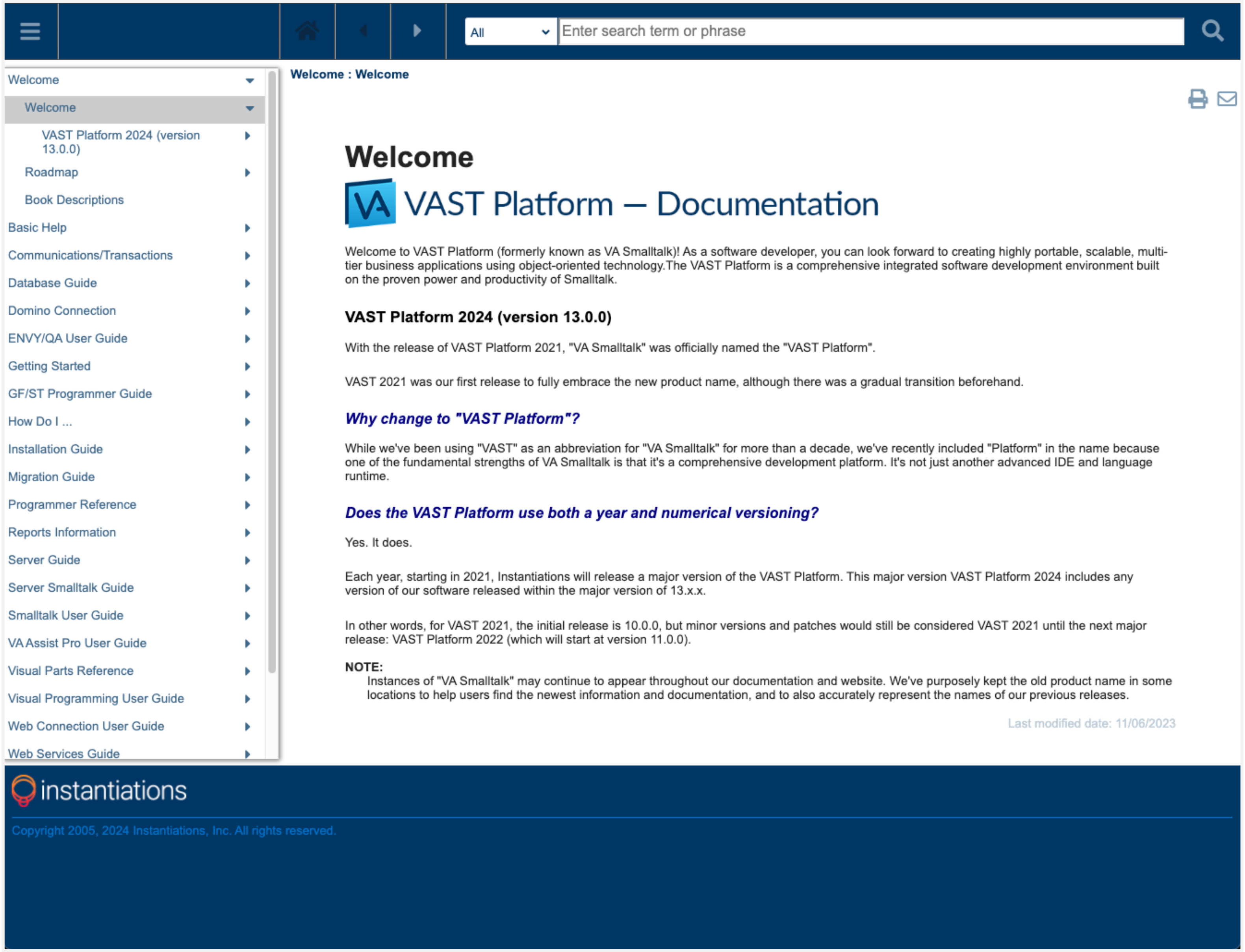
- Benchmarking
  - Performance benchmarks
  - Comparisons with previous builds

instantiations

Documenting

instantiations

All versions documented!



Welcome : Welcome

- Welcome
  - Welcome
    - VAST Platform 2024 (version 13.0.0)
  - Roadmap
  - Book Descriptions
- Basic Help
- Communications/Transactions
- Database Guide
- Domino Connection
- ENVY/QA User Guide
- Getting Started
- GF/ST Programmer Guide
- How Do I ...
- Installation Guide
- Migration Guide
- Programmer Reference
- Reports Information
- Server Guide
- Server Smalltalk Guide
- Smalltalk User Guide
- VA Assist Pro User Guide
- Visual Parts Reference
- Visual Programming User Guide
- Web Connection User Guide
- Web Services Guide

Welcome

VAST Platform — Documentation

Welcome to VAST Platform (formerly known as VA Smalltalk)! As a software developer, you can look forward to creating highly portable, scalable, multi-tier business applications using object-oriented technology.The VAST Platform is a comprehensive integrated software development environment built on the proven power and productivity of Smalltalk.

VAST Platform 2024 (version 13.0.0)

With the release of VAST Platform 2021, "VA Smalltalk" was officially named the "VAST Platform".

VAST 2021 was our first release to fully embrace the new product name, although there was a gradual transition beforehand.

Why change to "VAST Platform"?

While we've been using "VAST" as an abbreviation for "VA Smalltalk" for more than a decade, we've recently included "Platform" in the name because one of the fundamental strengths of VA Smalltalk is that it's a comprehensive development platform. It's not just another advanced IDE and language runtime.

Does the VAST Platform use both a year and numerical versioning?

Yes. It does.

Each year, starting in 2021, Instantiations will release a major version of the VAST Platform. This major version VAST Platform 2024 includes any version of our software released within the major version of 13.x.x.

In other words, for VAST 2021, the initial release is 10.0.0, but minor versions and patches would still be considered VAST 2021 until the next major release: VAST Platform 2022 (which will start at version 11.0.0).

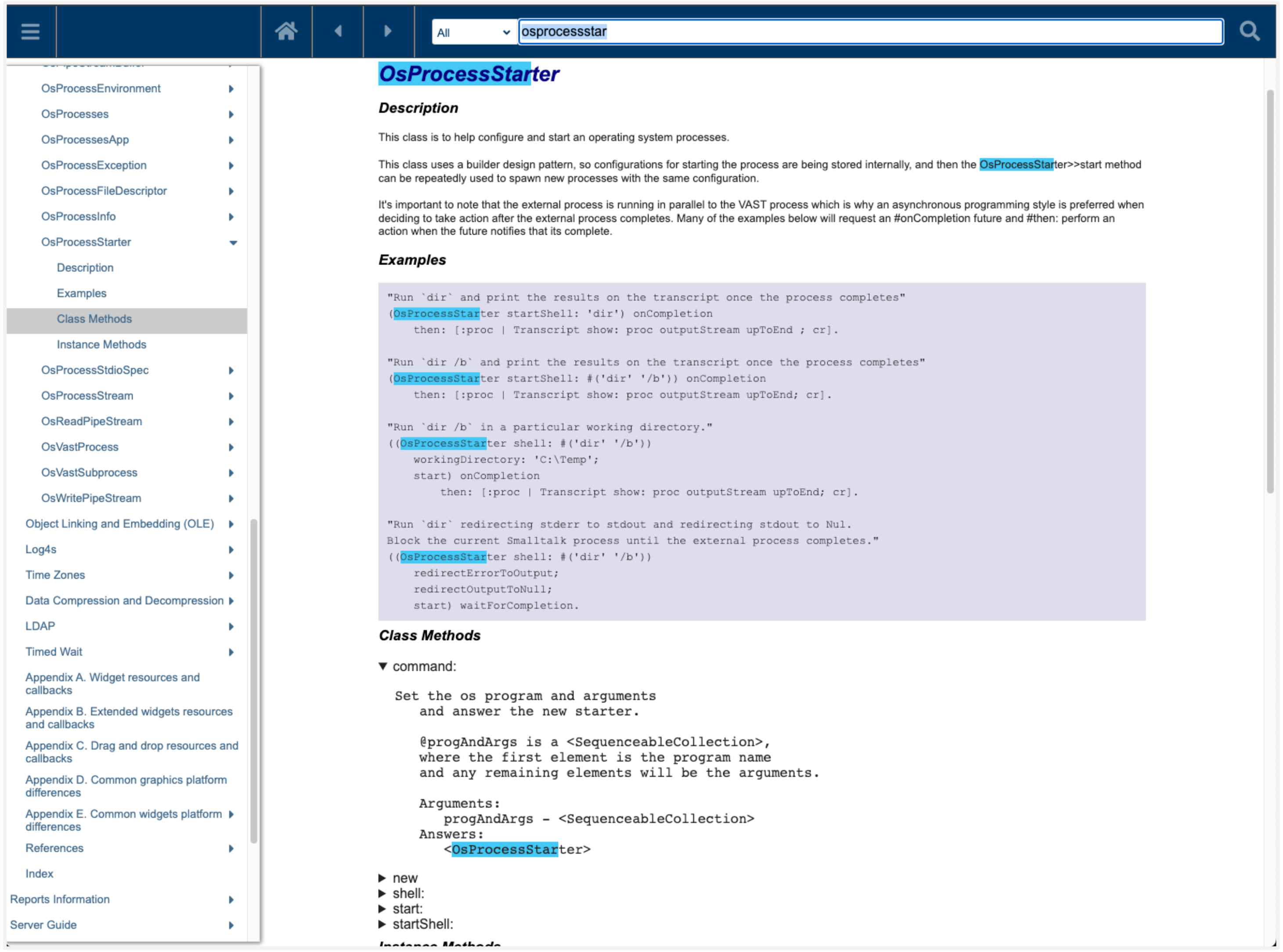
NOTE:
Instances of "VA Smalltalk" may continue to appear throughout our documentation and website. We've purposely kept the old product name in some locations to help users find the newest information and documentation, and to also accurately represent the names of our previous releases.

Last modified date: 11/06/2023

instantiations

instantiations

Documentation auto-generated from comments!



OsProcessStarter

Description

This class is to help configure and start an operating system processes.

This class uses a builder design pattern, so configurations for starting the process are being stored internally, and then the OsProcessStarter>>start method can be repeatedly used to spawn new processes with the same configuration.

It's important to note that the external process is running in parallel to the VAST process which is why an asynchronous programming style is preferred when deciding to take action after the external process completes. Many of the examples below will request an #onCompletion future and #then: perform an action when the future notifies that its complete.

Examples

```
"Run `dir` and print the results on the transcript once the process completes"
(OsProcessStarter startShell: 'dir') onCompletion
    then: [:proc | Transcript show: proc outputStream upToEnd ; cr].

"Run `dir /b` and print the results on the transcript once the process completes"
(OsProcessStarter startShell: #('dir' '/b')) onCompletion
    then: [:proc | Transcript show: proc outputStream upToEnd; cr].

"Run `dir /b` in a particular working directory."
((OsProcessStarter shell: #('dir' '/b'))
    workingDirectory: 'C:\Temp';
    start) onCompletion
        then: [:proc | Transcript show: proc outputStream upToEnd; cr].

"Run `dir` redirecting stderr to stdout and redirecting stdout to Nul.
Block the current Smalltalk process until the external process completes."
((OsProcessStarter shell: #('dir' '/b'))
    redirectErrorToOutput;
    redirectOutputToNull;
    start) waitForCompletion.
```

Class Methods

▼ command:

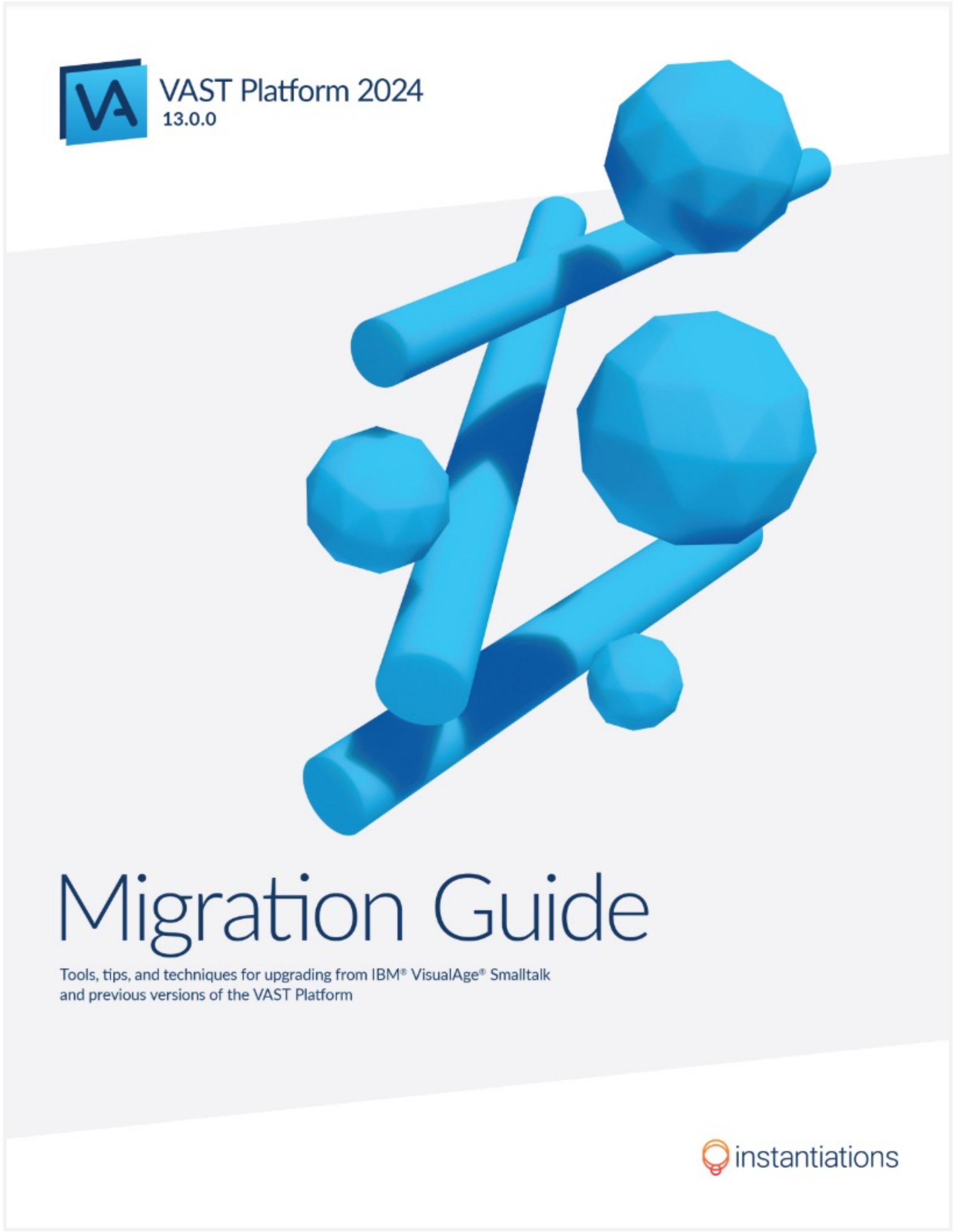
```
Set the os program and arguments
   and answer the new starter.

   @progAndArgs is a <SequenceableCollection>,
   where the first element is the program name
   and any remaining elements will be the arguments.

   Arguments:
      progAndArgs - <SequenceableCollection>
   Answers:
      <OsProcessStarter>
```

▶ new
▶ shell:
▶ start:
▶ startShell:

Migration Guide

Use it to upgrade from:

IBM VisualAge 3.0 (released in 1995)
to latest available VAST Platform!



VAST Platform 2024
13.0.0

Migration Guide

Tools, tips, and techniques for upgrading from IBM® VisualAge® Smalltalk
and previous versions of the VAST Platform

instantiations

Conclusion

instantiations

Why does all this matter?

- Software doesn’t live in isolation. It **MUST** move forward to effectively evolve alongside all the systems and platforms that surround it.
- This presentation outlined the foundation of what we call “responsible evolution”, which will continue to be the core engineering priority for VAST.
- “Responsible evolution” has allowed our customers to keep pace with the inevitable changes of technology, while also being able to count on the stability of a commercial Smalltalk system like the VAST Platform.

instantiations

instantiations

Questions?

Thanks for attending!

Mariano Martinez Peck
Team Lead & Sr. Software Engineer

mpeck@instantiations.com

@MartinezPeck

/in/mariano-martinez-peck

Contact

General Inquiry
info@instantiations.com

Sales
sales@instantiations.com

VAST Support Portal
vast-support.instantiations.com

North America, Toll Free
855 476 2558

International
+1 503 263 0058