Tornado:
A Run-Fail-Grow approach
for
Dynamic Application Tayloring

Author: 
Guillermo POLITO
Version: 1.0

Supervisors: 
Stéphane DUCASSE
Noury BOURAQADI
Luc FABRESE

May 22, 2014
Abstract

Producing a small deployment version of an application is a challenge because static abstractions such as packages cannot anticipate the use of their parts. As such, an application often occupies more memory than actually needed. To solve this problem we propose Tornado, a technique to dynamically tailor applications to only embed code (classes and methods) they use. Tornado uses a run-fail-grow approach to prepare an application for deployment. It launches minimal version of an application and installs a minimal set of statements that will start the user’s application. This application is run and these statements are executed. When the application fails because there are classes or methods missing, the necessary code is installed. The application is executed until it reaches a stable point, allowing possibly human interaction for applications with UIs. Thus, Tornado creates minimal memory footprint versions of applications by tailoring the whole application’s code, including run-time and third party libraries.

We used Tornado to tailor two different applications. We succeeded to tailor a hello world application to occupy 1% of its original size. We also experimented with a Seaside web application tailoring in one case only the application’s and framework’s code and the whole application’s code in the other case. In this latter example, we reached memory savings of about 97%. In this report we present an overview on Tornado, and we give details of the results we obtained.
## Contents

1 **Introduction**  
   1.1 Unused Code Units by Example  
   1.2 Challenges of Application Tailoring  
   1.3 Tornado in a Nutshell  

2 **The Run-Fail-Grow Approach**  
   2.1 Run-Fail-Grow with an example  
   2.2 Tornado's architecture  
   2.3 Detecting Missing Code Units with a Run-Fail approach  

3 **Chosen Technology: Pharo**  

4 **The object runtime manipulation interface: Oz object spaces**  

5 **The Execution Traps: Advanced Intercession with Ghost-like Proxies**  

6 **Object Installation and Object Mappings**  

7 **Handling Tailoring Levels through Seeds**  
   7.1 Loading an already existing memory snapshot  
   7.2 Creating all seed code units from scratch  

8 **Preparing the Application for Deployment**  
   8.1 Snapshot  
   8.2 Building a Static Description  

9 **Results**  
   9.1 Methodology  
   9.2 Hello World Application  
   9.3 Seaside Web Application  

10 **Conclusion**  

A Appendix: Method List of a Nurtured Hello World Application  
B Appendix: Entry Points to Tailor the Seaside Web Application  
C Appendix: Method List of Seaside Counter Application with Full Pharo Seed  
D Appendix: Method List of Seaside Counter Application with Empty Seed
1 Introduction

Deployed object-oriented applications often contain code units (e.g. packages, classes, methods) that the running application never uses. This problem shows itself more evident and harder to control under the usage of third party software. Third party libraries and frameworks are designed in a generic fashion that allows multiple usages and functionalities, while applications use only few of them. Examples are logging libraries, web application frameworks or object-relational mappers.

Unused deployed code units have an undesired impact when targeting a constrained infrastructure. Constrained devices may present restrictive hardware such as low primary or secondary memory, or even software impositions such as the Android’s Dalvik VM restriction to deploy only 65536 methods\(^1\). Big JavaScript mashup applications have an impact on loading time due to network speed and parsing time. These limitations may forbid the deployment of applications that contain lots of code units, or limit the amount of applications and content an user can have in its device.

Existant solutions to this problem propose the extraction of used code units of an application to reduce their size and memory footprint. Java Micro Edition [8] proposes a general purpose specialized runtime environment with no possibility of customization. Other solutions in the field propose to automatically detect and extract used code units, so called tailoring, with static call graph construction as the most dominant technique [7]. Static approaches present limitations in the presence of dynamic features such as reflection or in the absence of static type annotations. Additionally, they are in general designed to extract all used code units with no possibility for the user to customize the process of selection.

1.1 Unused Code Units by Example

To clearly show the problem, consider the application using a logging library in Figure 1. An interface is present in the diagram to show polymorphism between two classes that do not share a class inheritance hierarchy. However, some languages, such as the dynamically typed ones, may not need to represent it in the source code.

Figure 2 shows the code of this application, written in the Pharo Smalltalk language. This application contains a MainApp class with a start method, which is the entry point of our application. The start method creates an instance of StdoutLogger and logs the application’s start and end. In turn, the StdoutLogger uses the stdout global instance to log in the standard output the current time and the message. To print the time, the StdoutLogger makes use of the Time class from the base libraries of the language. Note that for the sake of clarity, we didn’t include in the example all base libraries, though, in modern programming languages they represent a large codebase with several features going from networking to multithreading. For example, Java 8 SE contains 4240 classes\(^2\), and the development edition of Pharo 2.0 contains 3342 classes and traits.

\(^1\)According to dalvik’s bytecode documentation (http://source.android.com/devices/tech/dalvik/dalvik-bytecode.html), the source register accepts values between 0 and 65536.
\(^2\)according to the javadoc API
In this example we can detect the following unused code units, shown in grey in Figure 1 and Figure 2:

1. The logger library includes two logging classes (StdoutLogger and RemoteLogger). Only the StdoutLogger is used and thus, the RemoteLogger class can be discarded.

2. Since the MainApp class does not use the Socket class nor the RemoteLogger class (the only user of the Socket class), the Socket class can be discarded.

3. No class in the application makes use of the Date class. Then, this class can be safely removed.

4. The method newLine (lines 7-8 of Figure 2) of the StdoutLogger class is not used and can be also removed.

5. The StdoutLogger class uses the Time class to print the current time. Then, all code units that are not related to the Time now resolution or printing (i.e., time arithmetic) could be considered as unused.

We would like to generate a new version of this application not containing these unused code units while keeping the application’s behavior. We call this technique Deployment Unit Tailoring or Application Tailoring.

### 1.2 Challenges of Application Tailoring

A lot of work exists on the tailoring of statically-typed applications [4, 20, 12, 18, 24, 17, 23], where the type annotations aid in the resolution of which piece of code will be used during runtime. However, static analysis is not an option in the context of dynamically-typed languages or in the presence of meta-programming and reflection [11]. In this context of dynamically typed and object-oriented programs that may
MainApp>>start
logger := StdoutLogger new.
logger log: 'Application has started'.
"do something"
logger log: 'Application has finished'.

StdoutLogger>>newLine
stdout newLine.

StdoutLogger>>log: aMessage
stdout nextPutAll: Time now printString.
stdout nextPutAll: aMessage.
stdout newLine.

RemoteLogger>>log: aMessage
| socket |
socket := self newSocket.
socket nextPutAll: Time now printString.
socket nextPutAll: aMessage.
socket newLine.

RemoteLogger>>newSocket
"...."
"creates an instance of socket given some configuration"

Figure 2:  Code of the unused code units example. In gray, methods not used by the application.

use reflection, we identify the following main challenges for detecting unused code units:

**Dynamic typing.** Dynamic languages cannot benefit from static analysis due to the absence of type annotations. Those techniques used to detect used code units, such as call-graph analysis, need the support of more dynamic techniques such as tracking runtime information, following the application’s execution flow, or performing symbolic execution.

**Polymorphism and inheritance.** Polymorphism in object-oriented languages allows a code unit to treat objects of different concrete types in the same way as soon as they share a common interface. Inheritance plays a similar role: any class can extend another class and provide different behavior while sharing the same API. As a consequence, both polymorphism and inheritance make the behavior of a program more difficult to predict by just analyzing its code units [22].

**Base libraries are often VM managed.** In most of the modern object-oriented languages, base language libraries such as Java’s bootstrap class loaders or native methods are loaded and initialized by the Virtual Machine (VM) or some low-level component. Since most applications do not use all standard libraries even
if they are initialized, these often big code bases are potentially candidates for removal. However, this raises a challenge since it often requires VM modifications.

**Application runtime configuration.** Modern applications often contain libraries and frameworks besides their proper code. To make these different code units fit together, applications rely on heavy configurations. These configurations are usually present in configuration files looked up dynamically by the application. Based on these configurations, the dependency injection pattern is usually used to dynamically set up the application. This recurrent and standard process for configuring applications implies that static analysis will be inefficient to detect used code units without library-specific knowledge.

**Reflection.** Reflection makes static analysis inoperative by allowing an application to execute unanticipated pieces of code. Any String resulting from a program execution or program configuration can denote a message send\(^3\), the name of a class to be instantiated, or even a script to be executed. Reflection is indeed important to cover, since it is a broadly used tool in industrial applications with object relational mappers such as Hibernate or Glorp and web frameworks such as Ruby On Rails, Struts or Seaside.

1.3 **Tornado in a Nutshell.**

The rest of this report describes Tornado: a solution to these issues with a novel flexible application tailoring technique. Tornado uses a run-fail-grow approach to identify during runtime those code units that are actually used in an application. It consists into "growing" a seed into a deployable specialized version of an application. Missing code units are used to "feed" a new minimal version of the application (the seed). The resulting deployable application only embeds the seed and used code. By carefully choosing the seed, different levels of tailoring are possible. For example, a seed that includes all the base libraries makes the tailoring process to only select used code in the application-specific part; whereas an empty seed makes the tailoring process to select used code in all parts: base libraries, application libraries and application-specific part. The dynamic nature of our solution allows its usage in languages without type annotations. Our solution does not need to modify the original application thanks to its run-fail approach. It also successfully deals with applications that make use of reflection.

2 **The Run-Fail-Grow Approach**

We propose Tornado, a run-fail-grow approach for tailoring. Tornado works by launching a nurtured application that has only part of its required code units installed and a reference application encompassing all the code units that resulted from the development process. When a failure is detected in the nurtured application, Tornado takes

\(^3\)We refer method invocations as message sends because they represent better from our understanding the dynamic property of the invocation.
the missing code units from the reference application and install them into the nurtured application. Thus, the nurtured application grows progressively as failures are found. Once finished, the nurtured application is ready to be deployed on target devices. Figure 3 depicts the basics of our run-fail-grow approach.

![Diagram of Reference Application and Nurtured Application]

**Figure 3: Application tailoring with a run-fail-grow approach.** We run the nurtured application (1) and detects the missing units on failure (2). At each failure, missing code units are installed from the reference application (3) and the execution restarted (4) until the process finishes.

Tornado starts by launching the reference application to create initial objects and perform startup computations. The reference application is then paused so its state does not change during the tailoring process. Pausing consists in suspending all processes and threads from the application.

Initially, the nurtured application is initialized with only a seed embedding code units that developers want to ensure into the deployed application. When the nurtured application starts from a seed that contains the language base libraries, the tailoring will only affect the application specific code units and third-party libraries. When it starts from an empty seed, it will also tailor base libraries.

Following, it installs one or more application’s entry points. An application’s entry point consists in one or more statements that perform some initial computations of the application (e.g., a main() method in Java, or the initial method of a thread). The execution of an entry point will result into sending messages to some objects. Required code units will then be cloned on demand from the reference application into the nurtured one. Duplication is performed lazily. For example, when duplicating a class, the content of its fields is not duplicated with it, but deferred until it is actually needed. Also, methods are not duplicated until they are invoked. The process repeats until the user ends it explicitly. Ideally, the nurtured application reaches a stable point where it needs no more code units. The nurtured application is then ready for deployment.
2.1 Run-Fail-Grow with an example

We illustrate in this section the ideas behind our tailoring approach with the example introduced in Section 1.1. For the sake of clarity, in this example we will tailor the application’s code units and not the base libraries i.e. the seed includes the base libraries.

**Setup of Tornado.** First, Tornado launches reference application with all its code units (cf. Figure 4) and the nurtured application (cf. Figure 5 Step 0). It initializes the nurtured application with a seed containing the language base libraries. Thus, each application has its own copy of the base libraries of the language, as shown in this case with the Date and Time classes and the stdout object.

![Figure 4: Reference application with all code units.](image)

**Install the application’s entry point.** We install the entry point of the application into the nurtured application. In our example, this is an instance of the MainApp class performing the start message (cf. Figure 5 Step 1). Note that although we are referencing an instance of the class MainApp, the MainApp class is not installed yet.

When the mainApp instance receives the start message Tornado realizes that the MainApp class and the start method do not exist in the nurtured application. Tornado then installs these two missing code units (cf. Figure 5 Step 2) and finally the MainApp.start method is activated and starts running.

**Activating the start method.** The execution starts by activating the start method, defined in Figure 2. As we can see in Figure 5 Step 2, the StdoutLogger class does not exist yet in the nurtured application. When it’s the turn of the execution of the first statement of the start method (line 2 Figure 2), Tornado captures the message `new`. 
Thus, before the statement’s execution, Tornado installs a StdoutLogger class the same shape as its original counterpart (cf. Figure 5 Step 3). However, it will not contain all the methods nor the meta-data (e.g., superclass, package, subclasses) existing in the reference class since they may not be necessary.

Once Tornado installs the StdoutLogger class, it sends the message `new` to it. This message-send results into a new StdoutLogger instance. Tornado is not involved in the resolution of the `new` message because this method is part of the language base library, already available in the seed.

The second statement of the `start` method (line 3 Figure 2) is now executed. The logger instance receives the message `log:` with its corresponding argument (cf. Figure 5 Step 4). Tornado captures the `log:` message because the corresponding method is not installed in the StdoutLogger class. Thus, it installs the method inside the corresponding class, and re-sends the message to the logger instance. This time the method is found, and the `log:` method is activated.

Once the `log:` method finishes, the execution returns to the `start` method. There, the third statement (line 5 Figure 2) is executed with no intervention of Tornado, since the `log:` method is already available. Figure 5 Step 5 shows the finally nurtured application: it contains only the methods and classes that are actually used from our application. Leaf objects used during the process were garbage collected.
2.2 Tornado’s architecture

Tornado is based on an architecture that allows the complex manipulation of both the nurtured and the reference applications. Examples of such complex manipulations are e.g., Tornado must be notified when a failure occurs in the nurtured application because some code unit is missing, it should be able to start/pause/stop its execution at safe points, install missing code units such as classes and methods, and query runtime information from both the reference and nurtured applications. Our approach performs those tasks through the manipulation of the object runtime systems of the applications.

An object runtime system is the runtime system of an object-oriented application e.g., a running Java virtual machine executing a Java program, and thus, containing the objects and classes of the application. We identify the following components as a part of the architecture of our solution (cf. Fig. 6):

![Figure 6: Tornado’s architecture overview.](image)

**Object runtime manipulation interface.** An object runtime manipulation interface allows one to control the runtime execution (starting, pausing and restarting it, and installing new threads/processes), install and load code units such as classes and methods, and query runtime information like the loaded classes, from an object runtime system. A well known example of such an interface is the JVM TI (JVM tool interface) [9]. A typical use of this module is to suspend the execution of the nurtured application while installing missing code units.

**Advanced intercession module.** An advanced intercession module allows advanced reflective capabilities such as modifying an object’s behavior during runtime. Tornado uses this module to capture message sends and so to be notified when it founds missing code units. JRebel [26], Reflectivity [5] or Bifrost [19] are examples of such intercession libraries.
2.3 Detecting Missing Code Units with a Run-Fail approach

Tornado gets notified whenever a certain code unit is missing by installing traps in the nurtured application. Traps are installed dynamically following the information flow of the application e.g., when a method $A$ is installed some traps are installed on it before resuming the execution. Tornado works under a run-fail-grow process based on its traps, as shown in Algorithm 1. Whenever a trap is found during execution, Tornado installs the missing code unit, installs some new traps if needed, and finally prepares the application to continue its execution by restarting the message-send that activated the trap.

```
Initialize reference application;
Initialize nurturing application with the seed;
Install entry point(s);
while not finished do
    run the nurtured application;
    if trap was activated then
        install missing code units;
        restart message send;
    end
end
```

\textbf{Algorithm 1}: An abstract view of the run-fail-grow process

We identified the following as the basic traps that are necessary to tailor an application:

\textbf{Missing object trap}. A \textit{missing object} trap captures message-sends to objects that do not yet exist inside the nurtured application such as classes. When Tornado finds one of these traps, its responsibility is to install the corresponding object. The object installed should be a clone of the original object, containing traps to capture the access to its class, instance variables and fields.

\textbf{Missing method trap}. A \textit{missing method} trap captures method invocations whose methods are not defined in the nurtured application yet. When Tornado detects one of these traps, it installs the corresponding method in the class hierarchy of the object. In case some classes are missing, Tornado installs them too. Missing method traps should also capture overridden methods. If an overridden method is not trapped, the method lookup may find a superclass implementation and execute it, resulting into an unexpected behavior. Figure 7 illustrates this problem: the class $B$ from the reference application contains an override, while it is not present in the nurtured application. If no trap is placed to capture the override, the method doSomething from class $A$ would be executed, thus changing the semantics of our application.
Figure 7: The need of override traps. Method traps should capture the overridden doSomething message-send to avoid the superclass method to be executed wrongly.

3 Chosen Technology: Pharo

We implemented Tornado in the Pharo programming language. Pharo is a reflective and dynamic programming language inspired on Smalltalk. In Pharo, code units such as classes and methods are reified, allowing their manipulation as any other object in the language. These reifications eased our implementation: classes and methods can be treated in the same way as other terminal objects such as strings with no extra effort.

Pharo is a modern programming language including modern language features and a growing infrastructure including:

**JIT Virtual Machine.** Pharo presents a modern Virtual Machine based on Cog’s architecture [14], containing a Just in Time compiler, polymorphic inline caches and stack optimizations.

**Open classes and class extensions.** Open classes and class extensions [2] allow a package to define methods in classes from other packages.

**Advanced reflection.** Pharo possesses the same reflective facilities present originally in Smalltalk-80 [6]. Introspection and interception in both behavioral and structural axis.

**First Class Instance Variables.** Pharo includes from its 3.0 release first class instance variables [25], namely Slots.

**Traits.** Pharo programming model includes Trais [21] in addition to classes to define the behavior of the system.
4 The object runtime manipulation interface: Oz object spaces

Tornado monitors the execution of the nurtured application and manipulates it during runtime as explained in Section 2.2. We built Tornado using our Oz\(^4\) object spaces [15] solution as the object runtime manipulation interface. Oz is an extension of the Pharo programming language implementing protection domains so called object spaces. Oz presents a first class representation of an object space providing a high level API to manipulate those protection domains.

In our Tornado implementation, the reference and nurtured applications are contained each in a different object space. Tornado places traps inside the nurtured object space and starts its execution. This execution is performed directly on a Pharo Virtual Machine, and thus, there is no speed overhead as soon as traps are involved. Whenever the nurtured object space’s execution founds a trap, it pauses and returns the control to Tornado. Tornado inspects the classes and methods in the reference object space through mirrors [3] and installs the needed code units from the reference object space on demand, either by creating new objects or compiling new methods. Then, it restarts the nurtured object space’s execution from the message send that activated the trap.

5 The Execution Traps: Advanced Intercession with Ghost-like Proxies

Implementing execution traps such as the ones described in Section 2.3 requires a powerful intercession module or library. Traps must capture all message sends to objects provided by the language runtime as well as the application objects. They must capture self and super message sends, as well as detect overrides.

To achieve this behavior, we implemented a set of proxies following the Ghost model [13]. Ghost is a low-memory footprint, general purpose proxy model supporting the creation of proxies for normal objects as well as classes and methods. Proxies allow detecting all situations corresponding to our traps. Tornado handles a table relating each proxy to the code unit or object it represents in the reference application. Additionally, each proxy is attached to a handler that may perform some action when detecting a message send. We rely on this concept to perform the right action for each trap. We discuss below the different kinds of proxies and handlers we use and how they support our run-fail-grow approach.

Missing object trap. This trap is implemented as a proxy taking the place of the original object. This trap is triggered when the proxy receives a message. Its handler replace the proxy by a copy of the original object from the reference application. The replacement of the proxy is achieved through the become: facility of the Pharo language that swaps pointers. Each field and the class of this new installed object are installed as new missing object traps.

\(\text{\textsuperscript{4}}\)Not related to the Oz programming language. The name Oz for Pharo is inspired on the metaphor of multiple world manipulation.
**Missing method trap.** We implemented the missing method trap in Tornado as a class proxy located at the top of the class hierarchy. Whenever a message is sent to an object, the VM looks up the method in the object’s class hierarchy. This trap is triggered whether a message arrives to the top of the hierarchy, meaning that there was no method for it in the hierarchy. When triggered, the handler installs the classes part of the hierarchy of this method and the missing method in its corresponding class. If no method is found to install in the hierarchy of the reference object space, Tornado sends the `doesNotUnderstand:` message (an equivalent to e.g., Ruby’s `method_missing` and Python’s `__getattr__`) to honor the dynamic semantics of Pharo.

**Missing override trap.** We implemented missing override traps in Tornado using method proxies. Method proxies are placed in the method dictionaries of classes containing overridden methods, taking the place of the original method. When Tornado installs a class into the nurtured application that contains overridden methods in the reference object space, it installs into this class a method proxy for each of its overridden methods. This proxy is activated whenever it is about to be executed. The handler of this trap takes the original method from the reference object space and compiles a new one with its same source code inside the nurtured object space.

**Primitive methods trap.** Primitive method traps are implementation specific related to the Pharo language. Pharo’s primitive operations such as number arithmetic are implemented through primitive methods. Primitive methods are implemented in the Virtual Machine and do often access directly the fields of its receiver and arguments by forging references and manipulating directly the memory. Thus, when a missing object trap proxy is the argument of such a method the VM can modify this proxy without activating the trap. Primitive method traps decorate Pharo’s primitive methods to capture their execution and trigger each of the missing object traps received as arguments. In this way, Tornado forces the installation of these traps and the primitive is executed with actual objects instead of proxies, as expected.

## 6 Object Installation and Object Mappings

Tornado installs all objects inside the nurtured application on demand. When Tornado installs an object inside the nurtured application, this new object has the same format and size as its original counterpart. Propagation rules determine how each of the object’s fields are treated on installation. Tornado provides the following propagation rules to customize installation:

**Missing object trap.** This is the default propagation rule. This propagation rules installs a missing object trap in each field of the object that is being installed.

**Materialization.** This propagation rule forces the installation of the object referenced by the field. This is used for those cases where some structure should be guaranteed to the Virtual Machine.
Swapping. This propagation rule forces the reference of the object installed be swapped to another object’s reference. The usual use case of this rule is to replace the reference to a reference to the nil object, and so, force lazy initializations.

Tornado takes care of the identity of objects with an identity table. This is important because Tornado works at the object granularity. Due to the inherent graph nature of object-oriented programs, an object being installed may reference another object that is already installed inside the nurtured application. In such a case, Tornado ensuring the correctness of the graph.

End user applications can be tailored usually with the default propagation rules. However, other propagation rules serve to the purpose of tailoring the base libraries. In Pharo, some special objects are used by the Virtual Machine (VM) and their installation should be forced to ensure its correct behavior. For example, the first three fields of class objects (superclass, format and method dictionary) cannot be proxified because they are used by the VM for the method lookup. The same happens with other objects reifying low level concepts such as methods, activation records or semaphores.

7 Handling Tailoring Levels through Seeds

Tornado’s seeds specify the level of tailoring. The seeds are in charge of initializing the nurtured application’s object space with the elements we want to ensure on it. Note that a seed can indeed contain any arbitrary code units and objects. They are not restricted to have only base or third party libraries. The selection or extraction of what is included as part of a seed is application dependent and orthogonal to the run-fail-grow process. Our current prototype supports two ways of describing and building seeds:

7.1 Loading an already existing memory snapshot

The nurtured application’s object space is initialized by loading an already existing snapshot or image (i.e., this is an image in the same sense as Smalltalk or Lisp). This technique consists in using a memory dump from an object heap containing all the classes and objects desired in the seed. This memory snapshot should follow Pharo’s object format.

Oz provides a primitive method to do such image loading. An image file is read, the internal representation of an object space is initialized, and an object space is retrieved.

7.2 Creating all seed code units from scratch

The nurtured application’s object space is initialized with objects built from scratch. This technique uses a bootstrapping process we defined in [16]. With bootstrapping, we describe declaratively the contents we want in the seed and we build it automatically.
8 Preparing the Application for Deployment

Once Tornado finishes and the application contains all the code units needed for running, it prepares it for deployment. That is: removing all trap leftovers and extract the nurtured application.

To remove the trap leftovers, Tornado identifies the traps by the presence of proxies and replaces the references to those proxies by references to another object, defaulting to the nil object. Proxy objects do not represent a drawback in space consumption because they are garbage collected. Once the traps are removed, the nurtured application keeps no dependencies to Tornado nor its infrastructure. Thus, the application can run outside the Oz infrastructure with no performance penalties.

8.1 Snapshot

To extract the nurtured application and allow it to run outside of Tornado’s infrastructure, our implementation makes a snapshot of it: it saves all objects, classes and methods in a binary format file to be later started with the actual state. Making a snapshot pays off because it speeds up the loading and starting time of the application. The snapshot technique is the same used by image-based languages such as Smalltalk, Lisp.

8.2 Building a Static Description

Alternatively to the snapshot approach, Oz allows one to inspect the state of an object space to know which classes and methods are installed inside it. Therefore we can build an static representation of the nurtured application code units. Appendix A, Appendix C and Appendix D shows the list of methods tailored by tornado in our case studies, extracted using Oz.

9 Results

9.1 Methodology

We tested our Tornado implementation by tailoring two different Pharo applications: a hello world application and a simple but yet interactive web application based on the Seaside framework [1]. Our methodology consisted in: setting up a seed for the application, preparing the application entry points and executing the application. In the case of the interactive web application, we interacted with it through a web browser. Once we finished the process, we extracted the resulting application by making a snapshot of it in a Pharo image file. We tested the generated snapshots to verify they work properly (under the assumption that only the previously used features of the application should work).

Finally, to present our results we measured the size of the generated snapshots files and compared them with the snapshots of the full applications under Pharo’s production
The results prove the soundness of our solution.

9.2 Hello World Application

We used Tornado to tailor a hello world application writing 10 times the ‘hello world’ string to the standard output (stdout). In this case study we used an empty seed to grow both base libraries and the application’s code. Figure 8 shows the installed entry point to tailor this application. Table 1 shows our results for this case. We succeed to reduce the application’s size to 1% of its original counterpart.

```
1 FileStream startUp: true.
2 1 to: 10 do: [ :i | FileStream stdout nextPutAll: 'hello'; crlf ].
```

Figure 8: Entry point of the Hello World application with an empty seed.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Size(KB)</th>
<th>Occupied(%)</th>
<th>Saved(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tailored</td>
<td>131</td>
<td>1%</td>
<td>99%</td>
</tr>
</tbody>
</table>

Table 1: Results of the tailored Hello World application.

9.3 Seaside Web Application

We also used Tornado to tailor a simple web application consisting in a webpage with a counter containing two buttons. These two buttons perform requests to the web server to increase and decrease the counter. The Seaside application framework was configured with its default values, without making any customizations.

In this case, we used two different seeds for tailoring: a seed containing all Pharo base libraries and an empty seed. Appendix B presents the entry points for these both seeds. The tailoring was done by starting the application and exercising it by generating requests through a web browser, clicking on its decrease and increase buttons.

Table 2 shows the results obtained when tailoring this application with each of these two seeds. Figure 9 presents a tailoring map illustrating how Tornado selects the code units from a reference application given a seeds. This figure also presents the notation we use in Table 2: $P$ is the Pharo base libraries, $S$ is the Seaside Framework and $C$ is the Counter application code units present in the reference application. $P'$, $S'$ and $C'$ are their counterparts selected by Tornado when using an empty seed. $P''$, $S''$ and $C''$ are their counterparts, as selected by Tornado when using a seed with all base libraries. In the latter, we can note that $P=P''$.

---

1 Pharo allows to prepare a snapshot for production. This option cleans some caches and removes some well known objects from the system, thus, freeing space.
Table 2: Results of second case study. Results of tailoring a web application with two different seeds. On the left, the total sizes of the original application deployment components (base libraries, application framework and counter application). On the right, our results when applying after tailoring. The first two results rows are compared against the total of the reference application. The third row presents the comparison without including base libraries, already inside the seed.

<table>
<thead>
<tr>
<th>Component</th>
<th>Size (KB)</th>
<th>Occupied (%)</th>
<th>Saved (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P' + S' + C' / P + S + C</td>
<td>573</td>
<td>3%</td>
<td>97%</td>
</tr>
<tr>
<td>P'' + S'' + C'' / P + S + C</td>
<td>13090</td>
<td>76%</td>
<td>24%</td>
</tr>
<tr>
<td>S'' + C'' / S + C</td>
<td>218</td>
<td>5%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Figure 9: Tailoring Map. Tailoring map describing the Seaside application generated with the empty seed (left) and the full Pharo seed (right).

10 Conclusion

In this paper our run-fail-grow approach for tailoring applications, namely Tornado. We presented both a reference model and a prototype implementation. Tornado tailors an application by starting it with a minimal set of entry points and installing missing code units on demand, following the execution flow of the application. Tornado defines a series of traps to detect missing code units.

By following the runtime execution, it supports dynamic features such as reflection and meta-programming. Tornado is able to tailor third party libraries and base libraries of the language, and so, to produce minimal footprint deployment units for applications. We validate our approach on two different applications: an hello world and a Web application. The results show that memory has been successfully saved while the resulting applications do still work.

We identify as future work the dynamic adaptation of a distributed Tornado-like system. We believe that our model in conjunction with reflective language architectures are one more step closer of the runtime adaptation and distribution of applications with
almost zero downtime.

Acknowledgements. This work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council, FEDER via the ’Contrat de Projets Etat Region (CPER) 2007-2013’, the Cutter ANR project, ANR-10-BLAN-0219.

References


A Appendix: Method List of a Nurtured Hello World Application

List of methods extracted from the nurtured Hello World application. This list includes all methods installed from the Pharo base libraries and the simple Hello World application.

Array class»new:
ArrayedCollection»size
Association class»key:value:
Association»value:
Association»value
BlockClosure»on:do:
BlockClosure»repeat
BlockClosure»valueNoContextSwitch
ByteString class»compare:with:collated:
ByteString class»findFirstInString:inSet:startingAt:
ByteString class»stringHash:initialHash:
ByteString class»at:put:
ByteString class»at:
ByteString class»isByteString
ByteString class»replaceFrom:to:with:startingAt:
ByteTextConverter class»unicodeToByteTable
ByteTextConverter»nextPut:toStream:
ByteTextConverter»unicodeToByte:
Character class»cr
Character class»lf
Character class»value:
Character»=
Character»asInteger
Character»asciiValue
Character»charCode
Collection»detect:ifNone:
Dictionary»at:ifAbsent:
Dictionary»at:ifPresent:
Dictionary»put:
Dictionary»noCheckAdd:
Dictionary»scanFor:
FileStream class»newForStdio
FileStream class»new
FileStream class»standardIOStreamNamed:forWrite:
FileStream class»openOnHandle:name:forWrite:
FileStream class»primWrite:from:startingAt:count:
FileStream class»primWrite:
FileStream class»startUp:
FileStream class»stdioHandles
FileStream class»voidStdioFiles
FileStream class»collectionSpeciesStandard
FileStream class»enableReadBufferingSmalltalkImage
FileStream class»isBinaryStandard
FileStream class»next:putAll:startingAt:
FileStream class»nextPut:
FileStream class»isString:
FileStream class»isCharacter:
FileStream class»voidStdioFiles
FileStream class»localeID:
LookupKey»key: Semaphore«critical:
LookupKey»key: SequenceableCollection«copyFrom:to:
Magnitude»max: SequenceableCollection«copyUpTo:
MultiByteFileStream»basicNext:putAll:startingAt: SequenceableCollection«do:
MultiByteFileStream»basicNextPut: SequenceableCollection«first:
MultiByteFileStream»converter: SequenceableCollection«first:
MultiByteFileStream»installLineEndConventionInConverter: SequenceableCollection«identityIndexOf:ifAbsent:
MultiByteFileStream»nextPut: SequenceableCollection«indexOf:ifAbsent:
Number»negative: SequenceableCollection«writeStream
OSPlatform class»isWin32: SimplifiedChineseEnvironment class«supportedLanguages
OSPlatform class»platformName: SmallInteger»bitXor:
Object»=: Stream«basicNextPut:
Object»at:put: Object»isCharacter String class«new:
Object»at: String class«with:
Object»hash: String«=:
Object»isInteger: String«compare:with:collated:
Object»species: String«findDelimiters:startingAt:
Object»~~ String«findTokens:
OrderedCollection class»arrayType: String«hash
OrderedCollection class»new: String«indexOf:startingAt:ifAbsent:
OrderedCollection class»new: String«isString
OrderedCollection»add: String«skipDelimiters:startingAt:
OrderedCollection»addLast: TextConverter class«defaultSystemConverter
OrderedCollection»at: TextConverter class«initializeLatin1MapAndEncodings
OrderedCollection»ensureBoundsFrom:to: TextConverter class«latin1Encodings
OrderedCollection»resetTo: TextConverter class«latin1Map
OrderedCollection»reset TextConverter«initialize
OrderedCollection»setCollection: TextConverter«installLineEndConvention:
OrderedCollection»size: TextConverter«nextPutAll:toStream:
PositionableStream class»on: TextConverter«nextPutByteString:toStream:
PositionableStream»isBinary VirtualMachine class«getStringAttribute:
PositionableStream»on: WriteStream«contentsStandard
ProtoObject»basicIdentityHash WriteStream«crlf
ProtoObject»flag: WriteStream«nextPut:
ProtoObject»identityHash: WriteStream«on:
ProtoObject»initialize: WriteStream«reset
RussianEnvironment class«supportedLanguages
B Appendix: Entry Points to Tailor the Seaside Web Application

Entry points as used to tailor the Seaside web application with a full Pharo seed and an empty seed. The first one (Figure 10) only consists in starting the web server as the base libraries are initialized and available in the seed. The latter one (Figure 11) includes the initialization of the minimal runtime needed to do networking.

1 ZnZincServerAdaptor startOn: 8888.

Figure 10: Entry point of the Seaside application with a full Pharo seed.

"We initialize some classes of the system"
2 SmaltalkImage initializeForTornado.
3 Symbol initializeForTornado.
4 Object initialize.
5 ExternalSemaphoreTable initialize.
6 Socket initialize.
7 Delay initialize.
8 Delay startUp: true.
9 Delay shutDown: true.
10 OSPlatform initialize.
11 DiskStore initialize.
12 FileStream initialize.
13 NetNameResolver initialize.
14 DateAndTime initialize.
15 ProcessorScheduler initialize.
16 WeakFinalizationList initialize.
17 UUIDGenerator initialize.
18 WeakArray initialize.
19 GRPharoRandomProvider initialize.
20 WASlime initialize.
21 UIManager basicDefault: DummyUIManager new.
22 ZnServer initialize.
23 WAServerManager initialize.
24 Smaltalk instVarNamed: 'session' put: Smaltalk newSessionObject.
25 Smaltalk startupImage: true snapshotWorked: true.
26 "Finally we start the web server"
27 ZnZincServerAdaptor startOn: 8888.

Figure 11: Entry point of the Seaside application with an empty seed.
C Appendix: Method List of Seaside Counter Application with Full Pharo Seed

List of methods extracted from the nurtured Web application when using a seed containing all base libraries from Pharo. This list includes all methods installed from Seaside framework and the counter application. The list of methods part of the base library are excluded as it is the same list of the methods found in Pharo base library.

WAIntervalReapingStrategy » defaultConfiguration
WAIntervalReapingStrategy » initialize
WAIntervalReapingStrategy » interval
WAIntervalReapingStrategy » reap
WAIntervalReapingStrategy » stored:key:
WAApplication » content-type
WAApplication » doesHandlerSupportCookies:
WAApplication » handleDefault:
WAApplication » handleFiltered:
WAApplication » isApplication
WAApplication » keyField
WAApplication » libraries
WAApplication » mainClass
WAApplication » mimeType
WAApplication » newSession
WAApplication » resourceBaseUrl
WAApplication » sessionClass
WAApplication » configuration » parents
WAAccessIntervalReapingStrategy » defaultConfiguration
WAAttributeSearchContext » initializeWithKey:
WAAttributeSearchContext » isAttributeInheritedOn:
WAAttributeSearchContext » isAttributeLocalOn:
WAAttributeSearchContext » key
WAAccessIntervalReapingStrategy » interval
WAAccessIntervalReapingStrategy » initialize
WAAccessIntervalReapingStrategy » key
WAAccessIntervalReapingStrategy » reap
WAAccessIntervalReapingStrategy » stored:key:
WAAnchorTag » callback:
WAAnchorTag » tag
WAAnchorTag » url
WAAnchorTag » with:
WAApplication » content-type
WAApplication » doesHandlerSupportCookies:
WAApplication » handleDefault:
WAApplication » handleFiltered:
WAApplication » isApplication
WAApplication » keyField
WAApplication » libraries
WAApplication » mainClass
WAApplication » mimeType
WAApplication » newSession
WAApplication » resourceBaseUrl
WAApplication » sessionClass
WAApplication » configuration » parents
WAAccessIntervalReapingStrategy » defaultConfiguration
WAAttributeSearchContext » initializeWithKey:
WAAttributeSearchContext » isAttributeInheritedOn:
WAAttributeSearchContext » isAttributeLocalOn:
WAAttributeSearchContext » key
WABrush » initialize
WABrush » parent
WABrush » setParent:canvas:
WABrush » with:
WACache » at:ifAbsent:
WACache » at:ifPresent:
WACache » at:put:
WACache » attribute
WACache » cachedValues
WACache » findAttributeAndSelectAncestorsOf:
WACacheCapacityConfiguration » describeOn:
WACacheMissStrategy » missed:
WACachePlugin » configuration
WACachePlugin » defaultConfiguration
WACachePlugin » initialize
WACachePlugin » removed:key:
WACachePlugin » retrieved:key:
<table>
<thead>
<tr>
<th>Class/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WACachePlugin.setCache</td>
<td>WADocument.close</td>
</tr>
<tr>
<td>WACachePlugin.stored.key</td>
<td>WADocument.destroy</td>
</tr>
<tr>
<td>WACacheReapingStrategy.reap</td>
<td>WADocument.initializeWithStream.codec</td>
</tr>
<tr>
<td>WACallback.class.on</td>
<td>WADocument.nextPut</td>
</tr>
<tr>
<td>WACallback.convertKey</td>
<td>WADocument.nextPutAll</td>
</tr>
<tr>
<td>WACallback.evaluateWithFieldValues</td>
<td>WADocument.open</td>
</tr>
<tr>
<td>WACallback.key</td>
<td>WADynamicVariable.class.use.during</td>
</tr>
<tr>
<td>WACallback.setKey callbacks</td>
<td>WADynamicVariable.class.value</td>
</tr>
<tr>
<td>WACallback.valueForField</td>
<td>WAEncoder.class.on.table</td>
</tr>
<tr>
<td>WACallbackRegistry.advanceKey</td>
<td>WAEncoder.class.on</td>
</tr>
<tr>
<td>WACallbackRegistry.handle</td>
<td>WAEncoder+initializeOn.table</td>
</tr>
<tr>
<td>WACallbackRegistry+increaseKey</td>
<td>WAEncoder.nextPut</td>
</tr>
<tr>
<td>WACallbackRegistry+initialize</td>
<td>WERRORHandler.class+exceptionSelector</td>
</tr>
<tr>
<td>WACallbackRegistry+nextKey</td>
<td>WAExampleComponent+rendererClass</td>
</tr>
<tr>
<td>WACallbackRegistry+store</td>
<td>WAExceptionFilter+exceptionHandler</td>
</tr>
<tr>
<td>WACanvas+brush</td>
<td>WAExceptionFilter+handleFiltered</td>
</tr>
<tr>
<td>WACanvas+flush</td>
<td>WAExceptionHandler.class+context</td>
</tr>
<tr>
<td>WACanvas+nest</td>
<td>WAExceptionHandler.class+exceptionSelector</td>
</tr>
<tr>
<td>WACanvas+render</td>
<td>WAExceptionHandler.class+handleExceptionsDuring:</td>
</tr>
<tr>
<td>WACanvas+text</td>
<td>WAExceptionHandler.class+handles</td>
</tr>
<tr>
<td>WACallbackRegistry+initialize</td>
<td>WAExceptionHandler.class+handles</td>
</tr>
<tr>
<td>WACallbackRegistry+nextKey</td>
<td>WAExceptionHandler+initializeWithContext</td>
</tr>
<tr>
<td>WACanvas+render</td>
<td>WAHeaderFields+checkValue</td>
</tr>
<tr>
<td>WACanvas+text</td>
<td>WAHeaderFields+privateAt:put</td>
</tr>
<tr>
<td>WACounter.count</td>
<td>WAHeadingTag+initialize</td>
</tr>
<tr>
<td>WACounter+decrease</td>
<td>WAHeadingTag+level1</td>
</tr>
<tr>
<td>WACounter+increase</td>
<td>WAHeadingTag+level</td>
</tr>
<tr>
<td>WACounter+initialize</td>
<td>WAHeadingTag+tag</td>
</tr>
<tr>
<td>WACounter+initializeContentOn</td>
<td>WAHtmlAttributes+encodeOn</td>
</tr>
<tr>
<td>WACounter+renderContentOn</td>
<td>WAHtmlAttributes+privateAt:put</td>
</tr>
<tr>
<td>WACounter+states</td>
<td>WAHtmlCanvas+anchor</td>
</tr>
<tr>
<td>WACounter-beXhtml10Strict</td>
<td>WAHtmlCanvas+heading</td>
</tr>
<tr>
<td>WACounter+spaceEntity</td>
<td>WAHtmlCanvas+heading</td>
</tr>
<tr>
<td>WACounter+initializeContentOn</td>
<td>WAHtmlCanvas+spaceEntity</td>
</tr>
<tr>
<td>WACounter+scriptGenerator</td>
<td>WAHtmlDocument+scriptGenerator</td>
</tr>
<tr>
<td>WACounter+scriptGenerator</td>
<td>WAHtmlDocument+scriptGenerator</td>
</tr>
<tr>
<td>WACounter+scriptGenerator+root</td>
<td>WAHtmlElement+class+root</td>
</tr>
<tr>
<td>WADefaultScriptGenerator+close.on</td>
<td>WAHtmlElement+attributeAt:put</td>
</tr>
<tr>
<td>WADefaultScriptGenerator+open.on</td>
<td>WAHtmlElement+attributes</td>
</tr>
<tr>
<td>WADevelopmentConfiguration+parents</td>
<td>WAHtmlElement+encodeBeforeAt:put</td>
</tr>
<tr>
<td>WADispatcher.class+default</td>
<td>WAHtmlElement+encodeOn</td>
</tr>
<tr>
<td>WADispatcher+handleFiltered.named</td>
<td>WAHtmlElement+initializeWithRoot:</td>
</tr>
<tr>
<td>WADispatcher+handleFiltered</td>
<td>WAHtmlElement+isClosed</td>
</tr>
<tr>
<td>WADispatcher+handlerAt:ifAbsent</td>
<td>WAHtmlRoot+add</td>
</tr>
<tr>
<td>WADispatcher+handlerAt:with</td>
<td>WAHtmlRoot+beXhtml10Strict</td>
</tr>
<tr>
<td>WADispatcher+handlers</td>
<td>WAHtmlRoot+bodyAttributes</td>
</tr>
<tr>
<td>WADispatcher+nameOfType:Handler</td>
<td>WAHtmlRoot+closeOn</td>
</tr>
<tr>
<td>WADispatcher+uriFor:</td>
<td>WAHtmlRoot+docType</td>
</tr>
<tr>
<td>WADocument.class+on.codec</td>
<td>WAHtmlRoot+htmlAttributes</td>
</tr>
</tbody>
</table>
WARenderContext » visitor
WARenderContext » visitor
WARenderLoopConfiguration » parents
WARenderLoopContinuation » createActionContinuation
WARenderLoopContinuation » createRenderContinuation
WARenderLoopContinuation » presenter
WARenderLoopContinuation » toPresenterSendRoot
WARenderLoopContinuation » updateRoot
WARenderLoopContinuation » updateStates
WARenderLoopContinuation » updateUrl
WARenderLoopContinuation » withNotificationHandler
WARenderLoopMain » createRoot
WARenderLoopMain » prepareRoot
WARenderLoopMain » rootClass
WARenderLoopMain » rootDecorationClasses
WARenderLoopMain » start
WARenderPhaseContinuation » createHtmlRootWithContext
WARenderPhaseContinuation » createRenderContext
WARenderPhaseContinuation » handleRequest
WARenderPhaseContinuation » processRendering
WARenderVisitor class » context
WARenderVisitor » initializeWithContext
WARenderVisitor » renderContext
WARenderer class » context
WARenderer » actionUrl
WARenderer » callbacks
WARenderer » context
WARenderer » document
WARenderer » flush
WARenderer » initializeWithContext
WARenderer » render
WARenderer » text
WARequest class » method:uri:version
WARequest » at:ifAbsent
WARequest » cookiesAt
WARequest » cookies
WARequest » destroy
WARequest » fields
WARequest » headerAt:ifAbsent
WARequest » headerAt
WARequest » initializeWithMethod:url:version
WARequest » isGet
WARequest » isPrefetch
WARequest » isXmlHttpRequest
WARequest » method
WARequest » postFields
WARequest » queryFields
WARequest » setBody
WARequest » setCookies
WAResponse class » messageForStatus: 
WAResponse class » statusFound 
WAResponse class » statusFound 
WAResponse » cookies 
WAResponse » destroy 
WAResponse » found 
WAResponse » headerAt: ifAbsent: 
WAResponse » headerAt: put: 
WAResponse » initializeOn: 
WAResponse » initialize 
WAResponse » location: 
WAResponse » redirectTo: 
WAResponse » status: message: 
WAResponse » status: 
WAResponse » status 
WAResponseGenerator class » expiredRegistryKey
WAResponseGenerator » initializeOn: 
WAResponseGenerator » requestContext 
WAResponseGenerator » request 
WAResponseGenerator » respond 
WAResponseGenerator » response 
WARoot class » context: 
WARoot » setContext: 
WAScriptGenerator » initialize 
WAScriptGenerator » loadScripts 
WAScriptGenerator » writeLoadScriptsOn: 
WAScriptGenerator » writeScriptTag: on: 
WAServerAdaptor class » default 
WAServerAdaptor » defaultPort 
WAServerAdaptor » defaultRequestHandler 
WAServerAdaptor » handle: 
WAServerAdaptor » handlePadding: 
WAServerAdaptor » handleRequest: 
WAServerAdaptor » initializeWithManager: 
WAServerAdaptor » initialize 
WAServerAdaptor » manager 
WAServerAdaptor » port: 
WAServerAdaptor » port 
WAServerAdaptor » process: 
WAServerAdaptor » requestFor: 
WAServerAdaptor » requestHandler 
WAServerAdaptor » responseFor: 
WAServerManager class » default 
WAServerManager » adaptors 
WAServerManager » canStart: 
WAServerManager » register: 
WAServerManager » start: 
WASession » actionField 
WASession » actionUrlForContinuation: 
WASession » actionUrlForKey: 
WASession » application 
WASession » clearJumpTo 
WASession » createCache 
WASession » handleFiltered: 
WASession » initializeFilters 
WASession » initialize 
WASession » isSession 
WASession » presenter 
WASession » properties 
WASession » start 
WASession » start 
WASession » unknownRequest 
WASession » updateRoot: 
WASession » updateStates: 
WASession » updateUrl: 
WASession » url 
WASessionContinuation » basicValue 
WASessionContinuation » captureAndInvoke 
WASessionContinuation » captureState 
WASessionContinuation » redirectToContinuation: 
WASessionContinuation » registerForUrl: 
WASessionContinuation » registerForUrl 
WASessionContinuation » request 
WASessionContinuation » respond: 
WASessionContinuation » setStates: 
WASessionContinuation » states 
WASessionContinuation » updateStates: 
WASessionContinuation » updateUrl: 
WASessionContinuation » value 
WASessionContinuation » withUnregisteredHandlerDo: 
WASnapshot » initialize 
WASnapshot » register: 
WASnapshot » reset 
WATagBrush » after 
WATagBrush » attributes 
WATagBrush » before 
WATagBrush » closeTag 
WATagBrush » document 
WATagBrush » isClosed
D Appendix: Method List of Seaside Counter Application with Empty Seed

List of methods extracted from the nurtured Web application when using an empty seed. This list includes all methods installed from Seaside framework, the Counter application and the base library of Pharo.

- Array class » new:
- Array » isSelfEvaluating
- Array » printOn:
- Array » replaceFrom: to: with: startingAt:
- ArrayedCollection class » new: withAll:
- Association class » key: value:
- BlockClosure » argumentCount
- BlockClosure » asContext
- BlockClosure » assert
- BlockClosure » asContextWithSender:
- BlockClosure » forkAt: named:
- BlockClosure » forkAt:
- BlockClosure » forkCallbackTemps
- BlockClosure » ifCurtailed:
- BlockClosure » ifError:
- BlockClosure » isClosure
- BlockClosure » newProcess
- BlockClosure » numArgs
- BlockClosure » numCopiedValues
- BlockClosure » outerContext
- BlockClosure » renderOn:
- BlockClosure » repeat
- BlockClosure » repeatWithGCIf:
- BlockClosure » startpc
- BlockClosure » value: value: value:
- ByteArray » asByteArray
- ByteString » at: put:
- ByteString » at:
- ByteString » beginsWith:
- ByteString » byteAt: put:
- ByteString » byteSize
- ByteString » findSubstring: in: startingAt: matchTable:
- ByteString » findSubstringViaPrimitive: in: startingAt: matchTable:
- ByteString » isByteString
- ByteString » isOctetString
- ByteString » privateAt: put:
- ByteString » stringHash: initialHash:
- ByteString » string:
- ByteSymbol class » stringHash: initialHash:
- ByteSymbol » at:
- ByteSymbol » findSubstring: in: startingAt: matchTable:
- ByteSymbol » isByteString
- ByteSymbol » privateAt: put:
- ByteSymbol » species
- CNGBTextConverter class » encodingNames
- CP1250TextConverter class » encodingNames
- CP1253TextConverter class » encodingNames
- ChangesLog class » default
- ChangesLog » recordStartupStamp
- Character class » codePoint:
- Character class » cr
- Character class » if
DateAndTime class»clock
DateAndTime class»fromSeconds:offset:
DateAndTime class»initializeOffsets
DateAndTime class»localOffset
DateAndTime class»milliSecondsSinceMidnight
DateAndTime class»millisecondClockValue
DateAndTime class»now
DateAndTime class»readFrom:
DateAndTime class»readOptionalSeparatorFrom:
DateAndTime class»readTimezoneOffsetFrom:
DateAndTime class»readTwoDigitIntegerFrom:
DateAndTime class»setJdn:seconds:nano:offset:
Delay class»primSignal:atMilliseconds:
Delay class»restoreResumptionTimes
Delay class»runTimerEventLoop
Delay class»scheduleDelay:
Delay class»startTimerEventLoop
Delay class»stopTimerEventLoop
Delay»beingWaitedOn:
Delay»delayDuration
Delay»resumptionTime:
DelayWaitTimeout»isExpired
DelayWaitTimeout»signalWaitingProcess
Dictionary»addAll:GRSmall
Dictionary»associationsDo:
Dictionary»at:ifAbsent:GRSmall
Dictionary»at:ifAbsent:Small
Dictionary»at:ifAbsentPut:GRSmall
Dictionary»at:ifAbsentPut:
Dictionary»at:ifPresent:GRSmall
Dictionary»at:ifPresent:
Dictionary»at:put:GRSmall
Dictionary»at:put:
Dictionary»fixCollisionsFrom:
Dictionary»findIndexFor:GRSmall
Dictionary»findIndexForKey:Small
Dictionary»schedule
Dictionary»setDelay:forSemaphore:
Dictionary»signalWaitingProcess
Dictionary»unschedule
Dictionary»wait
Integer»floor
Integer»isFraction
Integer»isInteger
Integer»noMask:
Integer»normalize
Integer»numerator
Integer»printOn:
Integer»printStringBase:length:padded:
Integer»printStringLength:padded:
Integer»quo:
Integer»rounded
Integer»timesRepeat:
Integer»truncated
Interval class»from:to:by:
Interval class»new
Interval»collect:
Interval»setFrom:to:by:
Interval»size
Interval»species
KOI8RTextConverter class»encodingNames
LanguageEnvironment class»defaultFileNameConverter
LargeNegativeInteger»negative
LargeNegativeInteger»normalize
LargePositiveInteger»*
LargePositiveInteger»+
LargePositiveInteger»-
LargePositiveInteger»//
LargePositiveInteger»<
LargePositiveInteger»asFloat
LargePositiveInteger»digitAt:
LargePositiveInteger»digitLength
LargePositiveInteger»highBitOfMagnitude
LargePositiveInteger»negated
LargePositiveInteger»negative
LargePositiveInteger»normalize
LargePositiveInteger»quo:
LargePositiveInteger»strictlyPositive
LargePositiveInteger»to:
LimitedWriteStream»nextPut:
LimitedWriteStream»nextPutAll:
LimitedWriteStream»setLimit:limitBlock:
LookupKey class»key:
Number class»one
Number»abs
Number»asDuration
Number»asFloat
Number»floor
Number»fractionPart
Number»integerPart
Number»isNumber
Number»isZero
Number»negated
Number»raisedToInteger:
Number»rem:
Number»strictlyPositive
Number»to:
MacOSXPlatform class»isActivePlatform
MacRomanTextConverter class»encodingNames
Magnitude»->
Magnitude»between:and:
Magnitude»max:
Magnitude»min:
MethodContext class»sender:receiver:method:arguments:
MethodContext»aboutToReturn:through:
PositionableStream->isEmpty
PositionableStream->on:
PositionableStream->originalContents
PositionableStream->peekFor:
PositionableStream->peek
PositionableStream->position:
PositionableStream->reset
PositionableStream->skip:
PositionableStream->skipSeparators
PositionableStream->skipTo:
Process class->forContext:priority:
Process->activateReturn:value:
Process->calleeOf:
Process->complete:
Process->isSetActiveOf:
Process->name:
Process->popTo:
Process->primitiveResume
Process->priority:
Process->priority
Process->psValueAt:put:
Process->psValueAt:
Process->resume
Process->return:value:
Process->suspendedContext:
Process->suspendingList
Process->suspend
Process->terminate
ProcessLocalVariable class->value:
ProcessLocalVariable->value:
ProcessSpecificVariable class->soleInstance
ProcessSpecificVariable->value
ProcessSpecificVariable->default
ProcessorScheduler class->idleProcess
ProcessorScheduler class->initialize
ProtoObject->identityHash
ProtoObject->initialize
ProtoObject->instVarsInclude:
ProtoObject->isNil
ProtoObject->pointsTo:
ProtoObject->~~
Random->initialize
ReadStream class->on:from:to:
ReadStream->on:from:to:
ReadStream->upTo:
ReadStream->upToEnd
Semaphore class->forMutualExclusion
Semaphore->critical:ifError:
Semaphore->critical:
Semaphore->initSignals
Semaphore->signal
Semaphore->waitTimeoutMSecs:
Semaphore->wait
Semaphore class->new
SequenceableCollection class->new:streamContents:
SequenceableCollection->streamContents:
SequenceableCollection->streamContents:limitedTo:
SequenceableCollection->streamSpecies
SequenceableCollection->allButFirst:
SequenceableCollection->at:ifAbsent:
SequenceableCollection->atAllPut:
SequenceableCollection->copyAfter:
SequenceableCollection->copyFrom:to:
SequenceableCollection->copyReplaceFrom:to:with:
SequenceableCollection->copyUpTo:
SequenceableCollection->do:
SequenceableCollection->do:separatedBy:
SequenceableCollection->doWithIndex:
SequenceableCollection->first:
SequenceableCollection->first
SequenceableCollection->from:to:put:
SequenceableCollection->grownBy:
SequenceableCollection->includes:
SequenceableCollection->indexOf:ifAbsent:
SequenceableCollection->indexOf:startingAt:ifAbsent:
SequenceableCollection->indexOfSubCollection:startingAt:
SequenceableCollection->keysAndValuesDo:
SequenceableCollection->last
WABrush
WABufferedResponse
WABufferedResponse+contents
WABufferedResponse+destroy
WABufferedResponse+initializeOn:
WABufferedResponse+stream
WACache+at:ifAbsent:
WACache+expiryPolicy
WACache+initializeCollections
WACache+initializeMutex
WACache+initialize
WACache+keyAtValue:ifAbsent:
WACache+keyAtValue:
WACache+keySize
WACache+missStrategy
WACache+notifyRemoved: key:
WACache+notifyRetrieved: key:
WACache+notifyStored: key:
WACache+pluginsDo:
WACache+reapingStrategy
WACache+reap
WACache+removalAction
WACache+setExpiryPolicy:
WACache+setMissStrategy:
WACache+setReapingStrategy:
WACache+setRemovalAction:
WACache+store:
WACacheCapacityConfiguration+describeOn:
WACachePlugin+configuration
WACachePlugin+defaultConfiguration
WACachePlugin+initialize
WACachePlugin+removed: key:
WACachePlugin+retrieved: key:
WACachePlugin+setCache:
WACachePlugin+stored: key:
WACachePlugin+reap
WACacheReapingStrategy+reap
WACallback class+on:
WACallback+convertKey:
WACallback+evaluateWithFieldValues:
WACallback+key
WACallback+setKey: callbacks:
WACallback+valueForField:
WACallbackRegistry+advanceKey
WACallbackRegistry+handle:
WACallbackRegistry+increaseKey
WACallbackRegistry+initialize
WACallbackRegistry+nextKey
WACallbackRegistry+store:
WACanvas+brush:
WACanvas+flush
WACanvas+nest:
WACanvas+render:
WACanvas+text:
WAComponent+accept:
WAComponent+acceptDecorated:
WAComponent+decoration
WAComponent+initialize
WACounter+count:
WACounter+decrease
WACounter+increase
WACounter+initialize
WACounter+states
WACounter+renderContentOn:
WACounter+renderFiltered:
WADispatcher+default
WADispatcher+handleFiltered:named:
WADispatcher+handleFiltered:
WADispatcher+handlerAt:ifAbsent:
WADispatcher+handlerAt:with:
WADispatcher+handlers
WADispatcher+nameOfHandler:
WADispatcher+urlFor:
WADocument class+on:
WADocument class+on: codec:
WADocument class+on: codec:
WADocument+close:
WADocument+destroy
WADocument+initializeWithStream: codec:
WADocument+initializeWithStream: codec:
WADocument+nextPut:
WADocument+nextPutAll:
WADocument+nextPutAll:
WADocument+open:
WADynamicVariable+class+use: during:
WADynamicVariable+class+value
WAEncoder class+on: table:
WAEncoder class+on:
WAEncoder class+on:
WAEncoder+initializeOn: table:
WAEncoder+nextPut:
WAExceptionFilter+exceptionHandler
WAExampleComponent+rendererClass
WAErrorHandler class+exceptionSelector
WAExceptionFilter+exceptionHandler
WAExceptionFilter+handleFiltered:
WARequestContext->responseGenerator
WARequestContext->response
WARequestContext->session
WARequestFilter->handleFiltered:
WARequestFilter->initialize
WARequestFilter->next
WARequestFilter->setNext:
WARequestFilter->updateStates:
WARequestHandler->addFilter:
WARequestHandler->addFilterLast:
WARequestHandler->basicUrl:
WARequestHandler->configuration:
WARequestHandler->configuration
WARequestHandler->defaultConfiguration
WARequestHandler->documentClass:
WARequestHandler->filters
WARequestHandler->filter
WARequestHandler->handle:
WARequestHandler->initialize
WARequestHandler->isApplication
WARequestHandler->isRoot
WARequestHandler->parent
WARequestHandler->preferenceAt:
WARequestHandler->responseGenerator
WARequestHandler->serverHostname
WARequestHandler->serverPath
WARequestHandler->serverPort
WARequestHandler->serverProtocol
WARequestHandler->setFilter:
WARequestHandler->setParent:
WARequestHandler->url
WAResponse class->messageForStatus:
WAResponse class->statusFound
WAResponse class->statusNotFound
WAResponse->contentType:
WAResponse->contentType
WAResponse->cookies
WAResponse->destroy
WAResponse->found
WAResponse->headerAt:ifAbsent:
WAResponse->headerAt:put:
WAResponse->headers:
WAResponse->initializeOn:
WAResponse->initialize
WAResponse->location:
WAResponse->nextPutAll:
WAResponse->notFound
WAResponse->redirectTo:
WAResponse->status:message:
WAResponse->status
WASession»handleFiltered:
WASession»initializeFilters
WASession»initialize
WASession»isSession
WASession»presenter
WASession»properties
WASession»start
WASession»updateRoot:
WASession»updateStates:
WASession»updateUrl:
WASession»url
WASessionContinuation»basicValue
WASessionContinuation»captureAndInvoke
WASessionContinuation»captureState
WASessionContinuation»redirectToContinuation:
WASessionContinuation»registerForUrl:
WASessionContinuation»registerForUrl
WASessionContinuation»request
WASessionContinuation»respond:
WASessionContinuation»setStates:
WASessionContinuation»states
WASessionContinuation»updateStates:
WASessionContinuation»value
WASessionContinuation»withUnregisteredHandlerDo:
WASlime class»initialize
WASlime class»startUp
WASnapshot»initialize
WASnapshot»register:
WASessionContinuation»basicValue
WASessionContinuation»captureAndInvoke
WASessionContinuation»captureState
WASessionContinuation»redirectToContinuation:
WASessionContinuation»registerForUrl:
WASessionContinuation»registerForUrl
WASessionContinuation»request
WASessionContinuation»respond:
WASessionContinuation»setStates:
WASessionContinuation»states
WASessionContinuation»updateStates:
WASessionContinuation»updateUrl:
WASessionContinuation»value
WASessionContinuation»withUnregisteredHandlerDo:
WAUrl class»absolute:
WAUrl class»decodePercent:
WAUrl initialize:
WAUrl addField:value:
WAUrl addField:
WAUrl decodedWith:
WAUrl encodeOn:
WAUrl encodePathOn:
WAUrl encodeQueryOn:
WAUrl encodeSchemeAndAuthorityOn:
WAUrl fragment
WAUrl initializeFromString:
WAUrl initialize
WAUrl parsePath:
WAUrl parseQuery:
WAUrl password
WAUrl path:
WAUrl pathElementsIn:do:
WAUrl pathString
WAUrl path
WAUrl postCopy
WAUrl printOn:
WAUrl queryFields:
WAUserConfiguration addParent:
WAUserConfiguration canAddParent:
WAUserConfiguration expressionAt:ifAbsent:
WAUserConfiguration initializeFrom:
WAValueExpression determineValueWithContext:configuration:
WAValueExpression value
WAValueHolder class»with:
WAValueHolder contents:
WAValueHolder contents
WAUpdateRootVisitor with:root:WeakKey
WAUpdateRootVisitor initializeWithRoot:
WAUpdateRootVisitor root
WAUpdateStatesVisitor visitPainter:
WAUpdateStatesVisitor snapshot:
WAUpdateStatesVisitor initializeWithSnapshot:
WAUpdateStatesVisitor visit:
WAXmIdocument closeTag:
WAXmIdocument destroy
WAXmIdocument initializeStream:codec:
WAXmIdocument openTag:attributes:
WAXmIdocument openTag:
WAXmIdocument openTag:
ZnUrl e enforceKnownScheme
ZnUrl hasFragment
ZnUrl hasHost
ZnUrl hasPath
ZnUrl hasPort
ZnUrl hasQuery
ZnUrl hasScheme
ZnUrl hasUsername
ZnUtils class signalProgress: total:
ZnUtils class streamingBufferSize
ZnZincServerAdaptor basicStart
ZnZincServerAdaptor configureDelegate
ZnZincServerAdaptor configureServerForBinaryReading
ZnZincServerAdaptor defaultCodec
ZnZincServerAdaptor defaultDelegate
ZnZincServerAdaptor defaultZnServer
ZnZincServerAdaptor isRunning
ZnZincServerAdaptor isStopped
ZnZincServerAdaptor requestAddressFor:
ZnZincServerAdaptor requestBodyFor:
ZnZincServerAdaptor requestCookiesFor:
ZnZincServerAdaptor requestFieldsFor:
ZnZincServerAdaptor requestHeadersFor:
ZnZincServerAdaptor requestMethodFor:
ZnZincServerAdaptor requestUrlFor:
ZnZincServerAdaptor requestVersionFor:
ZnZincServerAdaptor responseFrom:
ZnZincServerAdaptor server
ZnZincServerAdaptor shutDown
ZnZincServerAdaptor startUp
ZnUtils class httpDate:
ZnUtils class httpDate