Janos Java NodeOS Programming Manual

Flux Research Group
Department of Computer Science, University of Utah

http://www.cs.utah.edu/flux/janos/

March 17, 2002

This document applies to the Janos Java Libraries v1.2.0 and later snapshots containing minor updates.

The Janos Java NodeOS (or 'jnodeos') is a Java implementation of the NodeOS API Specification. The Janos Java NodeOS supports all of the major NodeOS abstractions including Threads, Channels, Domains, ThreadGroups, and DemultiplexKeys.

1 Organizational Overview

The Janos Java NodeOS distribution is organized as follows. The top-level includes the following directories

- `classes` contains the JanosVM emulation library. This library provides support for JanosVM facilities on standard JVMs. Most of the facilities are simple no-op stubs. The source for this emulation library is available at the Janos web page.
- `config` contains configuration support code. This code and the customized configure script are the basis for the autoconf-like configure tool used in the Java NodeOS. Feel free to reuse the configure engine elsewhere.
- `doc` the documentaiton directory. Both this manual and the javadoc-generated HTML documentation should be in this directory. The manual is available in both HTML and PostScript. All the documentation is also available on the Janos web site.
- `src` the Java NodeOS sources. More details on the sources are provided below.
- `testsuite` the regression test suite. The testsuite uses example programs from the src/test/ directory. A number of very basic regression tests are included. The infrastructure for running the regression tests is in the `testsuite/tools` directory, and should be usable for other projects.

The `src/` directory contains the most interesting bits. Specifically, the subdirectories breakdown as follows:

- `nodeos/` contains the core of the Janos Java NodeOS. The is where the implementation of the NodeOS abstractions is located. The two available "back-ends" for the Janos Java NodeOS are also in this directory. The compile-time choice of back-end is made by a `#define` in the `src/NativeNodeOS.cpp` file. Each back-end has a different prefix for its specific files. Specifically, all the files prefixed with `pj_` are part of the "pure java" back-end, while the `moab_` prefixed ones are part of the Moab-based back-end.

- `route/` contains a simple implementation of the NodeOS-level routing table. It is a simple table that maps addresses to route objects. Namespace controls can be used to restrict access to a read-only interface to the table.

- `io/` contains the logging interfaces provided for EEs built on our NodeOS. The logging API is based on a syslog-like facility, but is extended to allow multiple listeners to log messages.

- `util/` contains utility code for EEs running on the Janos NodeOS interface. All of these files offer simple, convenient interfaces to some of the more compilcated APIs in the NodeOS.

- `tests/` contains the test cases used by the regresssion testing suite. Most of the tests in the `src/tests/nodeos` subdirectory only really test the NodeOS when compiled and run on an "exposed" build.

- The `apps/`, `benchmarks/`, `protocols/`, and `services/` are still under heavy development but are intended to contain useful demonstrations, simple EEs, and utility code for building on the Janos Java NodeOS.

The source files actually contain a lot of useful and up-to-date comments so, as always, the source is the best place to go for answers.

2 Comparsion with the Specifications Goals

The current NodeOS Specification[1] specifies the following standard interfaces and abstractions. The "+" and "-" marks indicate the current degree of support in our Java binding.

+ Domain: the unit of accounting and control

+ ThreadPool: for controlling thread concurrency, interface to threads

+ Channels: sources and sinks for network traffic.

+/- Packets: for manipulating network traffic elements.

+/- FileSystem: interfaces to persistent storage

- MemoryPool: groups memory for inter-domain sharing.

- Events: for non-packet related timers and scheduling.

Of these interfaces, the Janos Java NodeOS currently provides Java bindings for the Domain, ThreadPool, and Channels. A Buffer abstraction that works at a higher level than the Packet abstraction is also provided. The Java bindings in this interface are meant to simultaneously take advantage of Java's direct support for objects (e.g., utilizing constructors) and hide NodeOS details that need not be exposed at this level (e.g., the memory page mapping interfaces). To these ends, on some details the mapping deviates from the C bindings defined in the NodeOS specification. The goal of this interface is that it be trivial to implement *on top of* the specified NodeOS C API, while only exposing the details that a Java-based EE needs.

There are currently two implementations of the Janos Java NodeOS. One, the "pure java" version, is written purely in Java and runs on standard JDK 1.1-compliant runtimes. The other implementation runs on the Janos project's C implementation of the NodeOS, Moab. The two implementations share a lot of code, the choice of which to use is a compile-time configuration option. For details read the installation instructions.

[1] *NodeOS Interface Specification*, Active Network Node OS Working Group, Draft of January 10, 2001. Available at `http://www.cs.princeton.edu/nsg/papers/nodeos.ps`

Several of the names chosen for Janos NodeOS Interface functions are different than their C API NodeOS counterparts. Some of these will be changed to reflect the NodeOS name; others are more in line with Java conventions and will remain. Details are provided later.

The javadoc-style documentation of the actual Java interfaces is available at `http://www.cs.utah.edu/flux/janos/apidoc/`.

3 Comparison to NodeOS C bindings

This section provides a function-to-function comparison of the C language binding of the NodeOS API with the equivalent Janos Java NodeOS interface function. This section will make sense only if you are *very* familiar with the NodeOS specification, or if you have a copy of it next to you.

Each of the Janos classes listed below is in the `edu.utah.janos.nodeos` package, e.g., the fully qualified name of the method `Domain.newDomain(...)` is `edu.utah.janos.nodeos.Domain.newDomain(...)`.

Because Java supports real namespaces (via package names), the Janos Java NodeOS dispenses with the `an_`, `AN_`, and `ani_` prefixes that plague the C implementation.

3.1 Domains

The Janos Domain abstraction is designed to directly map a NodeOS domain's capabilities into Java. There are no semantic changes introduced other than the `DomainHandle` which is a "cross-process" handle on a Domain object, and is used for termination.

- an_Domain an_domainCreate(an_Credentials *c, an_ThreadPoolSpec *tp, an_MemSpec *vm, an_NotifyFunc termHandler)
  See the Janos method `Domain.newDomain(Credentials cred, ThreadPoolSpec tPoolSpec, MemSpec memSpec, DomainStartSpec domainStartSpec, Runnable termHandler)`. Note that most of the parameters line up with the NodeOS C API. One difference in this case is the `DomainStartSpec`. This spec provides the set of Java class files that the new domain can see and gives the initial class and the set of arguments to pass to its `main` function. This deviation from the spec stems from a change to the ThreadPool that we are making (more below).

- void an_domainDestroy(an_Domain fid)
  See the Janos `Domain.destroy()` and `DomainHandle.destroy()` methods.

- an_Domain an_domainId(void)
  See the Janos `Domain.currentDomain()` method.

3.2 Thread Pool

Thread Pools are basically unchanged in the Janos implementation. We do expose a `Thread` object, but it contains only the features defined by the NodeOS Thread Pool interface. The Janos Java NodeOS bindings do not expose the ability to disable/enable interrupts or to change a thread's priority. Additionally, thread-interrupt handlers are not available at the Java level. Currently, the `interrupt()` method only provides the function provided by native Java; it will only wake sleeping threads and post an interrupt flag on them.

While the NodeOS interfaces are the same, one change is made to the semantics. The NodeOS C API maintains a strictly one-to-one binding between `ThreadPool` and `Domain`. We relax that restriction in Janos. Thus, in Janos, a domain's start function is part of the `Domain` object, and not part of the `ThreadPool` constructor (as in the NodeOS C API). This lets us create multiple `ThreadPools` within a single Domain, for example, to have different thread pools to service the control and data packets in a single logical domain.

- an_thread_t an_thread_self(void)
  See Janos `Thread.currentThread()`.
- an_error_t an_thread_start(an_thread_t *out_thread, an_Domain fid, void (*f)(void *arg), void *arg)
  See the Janos `ThreadPool.threadStart(Runnable r)` method.
- void an_thread_exit(void)
  Oops. We seem to have not implemented this one yet. Returning from the runnable that started the thread is equivalent, though.
- void an_thread_yield(void) See the Janos `Thread.yield()` method.
- void an_thread_sleep(const an_timespec_t *delay)
  See Janos `Thread.sleep(long millis, int nanos)`
- void an_thread_setdata(void *data)
  See Janos `Thread.setObject(Object o)`
- void * an_thread_getdata(void)
  See Janos `Thread.getObject()`
- void an_thread_interrupt(an_thread_t thread)
  See Janos `Thread.interrupt(Thread target)`. Beware, as none of the NodeOS-specified interrupt delivery or handling is yet really implemented in this Java binding. However, standard Java-specified "interrupt" is still available. I.e., this method will interrupt `wait()` calls. NodeOS methods protect themselves from interrupts and will delay the interrupt until after they have completed. Some long running methods will "correctly" handle an interrupt and return early.
- void an_thread_block_interrupts (void)
  Unavailable. See above.
- void an_thread_unblock_interrupts (void)
  Unavailable. See above.
- void an_thread_setprio(an_thread_t t, int priority)
  Not available at the Java level. The Java interface to thread scheduling parameters is not complete yet.
- int an_thread_getprio(an_thread_t t)
  Not available at the Java level.

In terms of synchronization, the Janos Java NodeOS interface provides the standard Java synchronization mechanisms in place of the NodeOS specified functions. Additionally, there is a `Semaphore` construct available. Mutexes may be added later, but we have not yet had a need for them.

3.3 Channels

The Janos Java NodeOS interface wraps the low-level NodeOS Channel abstractions into three distinct classes: `OutChannel`, `InChannel` and `CutThroughChannel`. Instead of sending or receiving `Packets`, at the Java level `BufferHandle` objects are used.

Just as in the NodeOS spec, Java Channel objects are fairly static. The constructor determines the demultiplex key, address, protocol processing, and thread pool that the channel should use.

3.4 Memory Pool

All memory management in Janos is implemented within the JVM (err, it will be implemented in there). There are currently no memory management interfaces at the Java level except for specifying a limit when a new Domain is created (see `MemSpec`).

3.5 FileSystem

Janos does not define any wrappers for the POSIX-like file system interfaces specified in the NodeOS document. On the other hand, the existing JDK file interfaces are quite close to POSIX already. Janos does define some utility classes for logging, but they are "above" the NodeOS specifications.

3.6 Events

There is some support in the implementation of `ThreadPools` and their `WorkItems` to support events, but no public interface is yet implemented (ANTSR didn't need it...).

3.7 Heap

Janos uses the Java-standard `new` to allocate objects. Since Java is garbage collected, the reclamation of objects is hidden from Java.

3.8 Packets

The NodeOS defines a `Packet` object for sending and receiving chunks of data from the network. `Packets` are designed to simplify adding and re-writing headers. The Janos NodeOS interface provides a higher-level abstraction, the `BufferHandle` which, in addition to the services provided by a `Packet`, provide a level of indirection that makes Buffers suitable for sharing between domains. See the Javadoc description of `BufferHandle` for more detail.

3.9 Inter-domain Sharing

The Janos NodeOS interface is designed for a JVM (the JanosVM, actually) that will provide strong separation of domain heaps. As such, sharing between domains is strictly limited to raw buffers (shared via `BufferHandles`).

Additionally, a rudimentary inter-domain communication mechanism, the `CommSpace`, has been implemented. These interfaces and their semantics are not derived from any part of the NodeOS specification. They are unique to the Janos approach to multiple domains in a single JVM. Note also that at the Java level we do not plan to expose any of the inter-domain or inter-EE facilities specified in the NodeOS spec.

Credits

The development effort was led by Patrick Tullmann. The Java NodeOS layer was developed by Patrick and Chris Hawblitzel, while the latter was on a summer internship at Utah. Tim Stack wrote the JanosVM bindings and re-worked a lot of the internals for that work. Patrick wrote most of this document.