

DIY Zoning: Changes to OWAPI

by Vadim Tkachenko

1. Introduction

`com.dalsemi.onewire` classes [originally provided](#) by [Dallas Semiconductor](#) were somewhat modified. A complete history of changes may be seen in the [CVS repository](#), but since the changes were fixing problems that are quite common, I thought that it would be a good idea to summarize them here, in case someone's interested, and just as a note to self, should I need to port another OWAPI release.

2. `com.dalsemi.onewire.OneWireException`

Original code was lacking a constructor with `Throwable` as an argument, which is a minor annoyance. A major annoyance, though, was that it didn't include a placeholder for the address of the 1-Wire® device that caused the problem. You'd probably agree that it is extremely convenient to know *what device* is the root cause of the problem, and a message like "OneWireException: device not present" is not particularly useful.

Note:

It is generally a very good idea to include some sort of a representation of a root cause into the exception. Be careful and don't include the object itself, lest exception gets stored somewhere along with the reference to the object, therefore disallowing its garbage collection.

3. `com.dalsemi.onewire.utils.OWPath`

Original implementation of `OWPath` couldn't be placed into ordered container - it didn't implement `Comparable` interface. So, now `OWPath` does implement `Comparable`, and in addition to that, `hashCode()` method had been added to make sure that it will behave properly in regard to `equal()` in a `Hashtable` or a `HashMap` - previously, it could have happened that hash based sets could contain more than one instance of `OWPath` that were considered `equal()`, but produced different `hashCode()`, with catastrophic consequences.

Note:

This omission, no matter how innocent looking, caused the most evasive [bug #594880](#) (Notification: departure before arrival),

which couldn't be fixed for almost exactly two years.

Note:

It is generally a good idea to implement `Comparable` interface where applicable, and *always* a good idea to implement `equals()` and `hashCode()` if an object is intended to be placed into a container.

4. `beginExclusive()/endExclusive()` considered harmful

Original OWAPI implementation uses `beginExclusive()/endExclusive()` to control access to the adapter in case when certain operation sequences have to be atomic. An implementation has several drawbacks that make it very awkward - in particular, the code loops using `Thread.sleep(50)`, it is not quite thread safe and what's most important, doesn't support nested calls - `endExclusive()` just releases the lock if called by the same thread. It doesn't check whether there were multiple `beginExclusive()` calls, and doesn't notify the caller if the caller doesn't belong to the thread that got the lock in the first place.

Therefore, these two method calls were replaced by [RWLock](#), which provides accountability (it is always possible to determine who's the current lock owner), nesting support (no matter how many times a lock is requested from the same thread, it will be released correctly) and proper support for Java semantics (the `RWLock` implementation is not looping, consuming resources, but waits on a semaphore).

© 2004 Vadim Tkachenko