Message-Driven Data Access Layer for Client Applications

Andrey Petrov • Controls' Meeting • 2011-05-04

Motivation

- Feeding Acnet data to non-AD clients (Synoptic Viewer is the principal app).
  - Strong user authentication, data security.
  - Firewall transversal.
  - Graceful handling of dropped connections.
  - Effective bandwidth utilization, asynchronous data callbacks.
- [Yet another] simplified API with minimal dependencies.

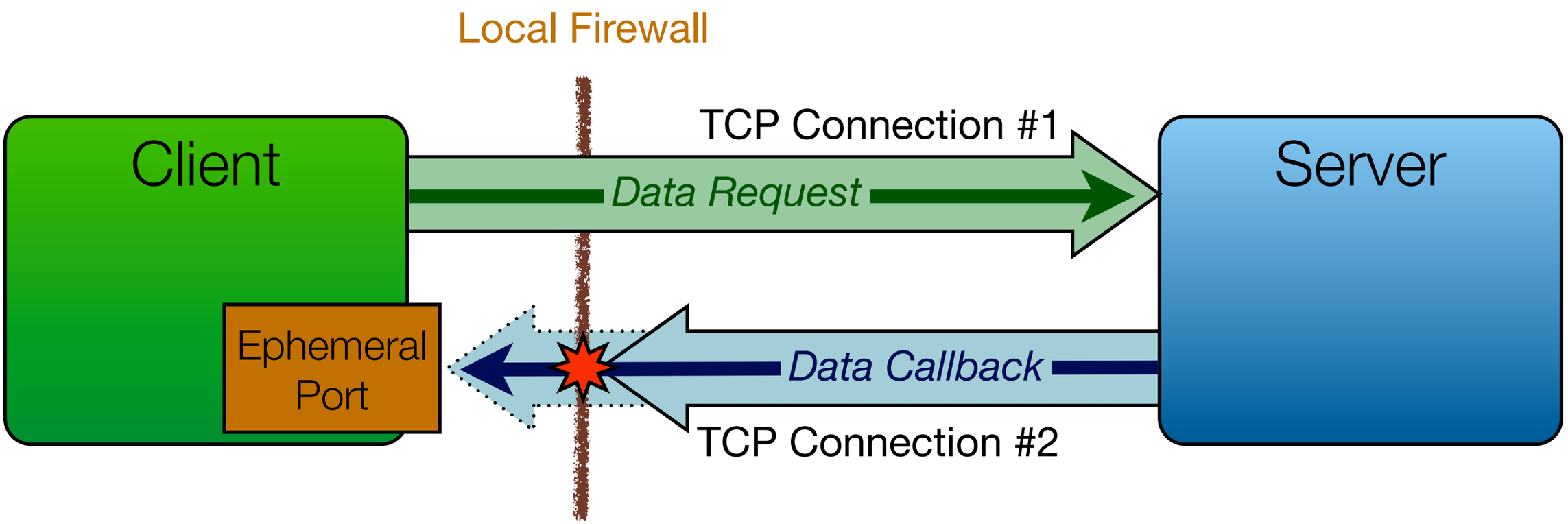
Security Issues

- Using Kerberos credentials uniformly for all kinds of clients.
  - Valid tickets are now available on public consoles and CLX boxes.
- Using message authentication codes to digitally sign critical communications.
  - Signed: requests to start data acquisition jobs, settings.
  - Not signed: readings, heartbeats.
- TLS (SSL) is supported in principle, but not yet enabled.
- Clients on the Fermilab network are allowed to connect, others use VPN.

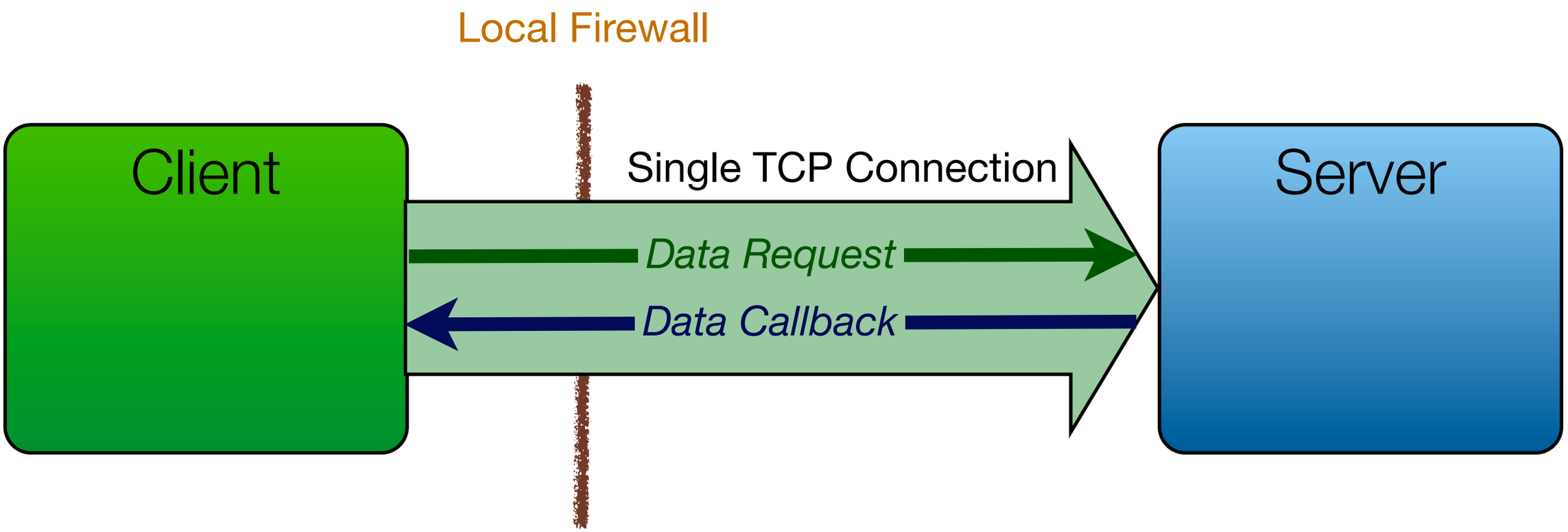
Effective Bandwidth Utilization

- AMQP over TCP/IP.
- The wire format of application data is provided by Protocol Compiler.
  - Binary.
  - Portable across platforms.

Firewall Transversal: How It Was (RMI)



Firewall Transversal: How It Should Be



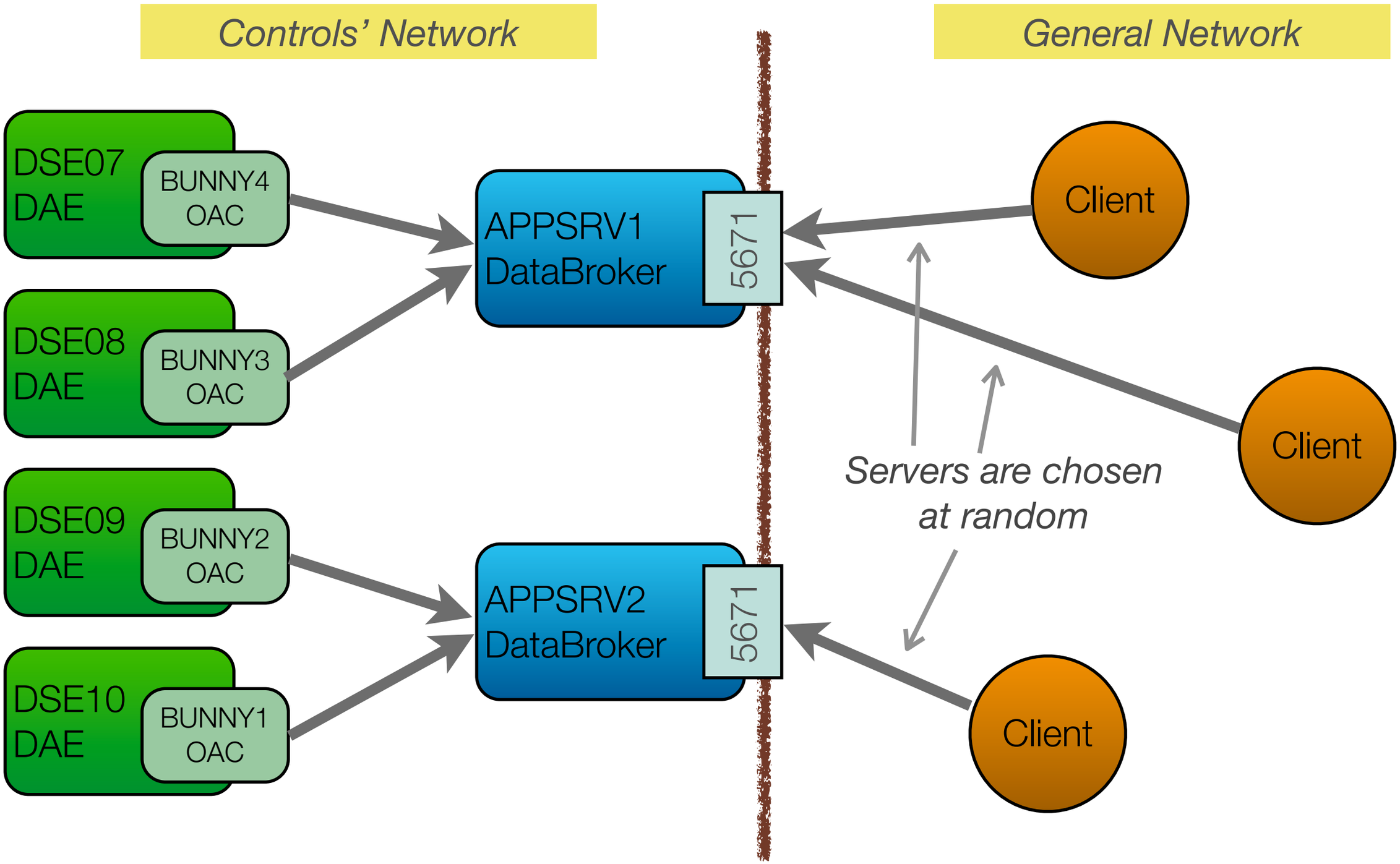
Issues with Remote Connections' Lifecycle

- Random disconnects are normal and must be expected.
  - The app doesn't need to know about the existence of a remote connection: Ideally, no *connect* and *disconnect* methods, no option to set the server name.
- The client's state associated with the connection needs to be restored.
- Detection of dead connections is a painful and imprecise task.
- Difficult to debug.

Messaging Passing (vs. Remote Calling)

- Another model of communication: no *stub* object representing the opposite party.
  - Messaging is not necessarily better than RPC, but it's completely different.
- Sender and receiver are loosely coupled: there is no state.
- Messages are asynchronous.
- Often includes an intermediary party — message broker.
- The broker queues messages and monitors the presence of clients.

Architecture of DAQ Messaging



The DMQ Programming Framework

- DMQ = Data Acquisition Message Queue

- Java packages:

  gov.fnal.controls.service.dmq ← public API

  gov.fnal.controls.service.dmq.examples ← examples

  gov.fnal.controls.tools.timed ← application data types

- Required libraries:

  jars/split/dmq-client.jar

  jars/rabbitmq-client.jar

  jars/commons-io.jar

  jars/split/kerberos-client.jars

Features of DMQ API

- A model of *data acquisition jobs*, similar to that of the old DAQ framework.
- The principal class is **gov.fnal.controls.service.dmq.DaqClient**; it's used as a factory to create reading and setting jobs.
- Readings (and settings' acknowledgements) are returned asynchronously via data callback listeners.
- There is no need to *connect* or *start* anything: all object are immediately usable after creation.
- Unneeded objects should be destroyed by calling **dispose()**. Destroyed objects are not reusable.

Example of Reading Data

```
DaqClient client = new DaqClient();
client.setCredentialHandler(new DefaultCredentialHandler());

Set<String> req = new HashSet<String>(
   Arrays.asList("M:OUTTMP", "B:TRZ[5:7]@P,1000,TRUE"));
ReadingJob job = client.createReadingJob(req);
job.addDataCallback(/*---data callback listener---*/);

// Some code...

job.destroy();
```

Example of Setting Data

```
DaqClient client = new DaqClient();
client.setCredentialHandler(new DefaultCredentialHandler());
```

```
client.enableSetting(1, TimeUnit.MINUTES);
```

```
Set<String> req = new HashSet<String>(
   Arrays.asList("Z_CACHE", "Z_SERBAL");
SettingJob job = client.createSettingJob(req);
job.addDataCallback(/*---data callback listener---*/); // Optional ack.
job.setData("Z_CACHE", /*---new data---*/);

// Some code

job.destroy();
```

Data Request Format

Described in http://www-bd.fnal.gov/controls/public/drf2

```
M:OUTTMP
    M:OUTTMP.READING@U
Z_SERBAL
    Z:SERBAL.SETTING@U
I:Q701K[5]@P,2500
I:K6AUCD.SETTING.RAW
I$H226S
Z|CACHE@E,1F
M~BLOW
```

← device description

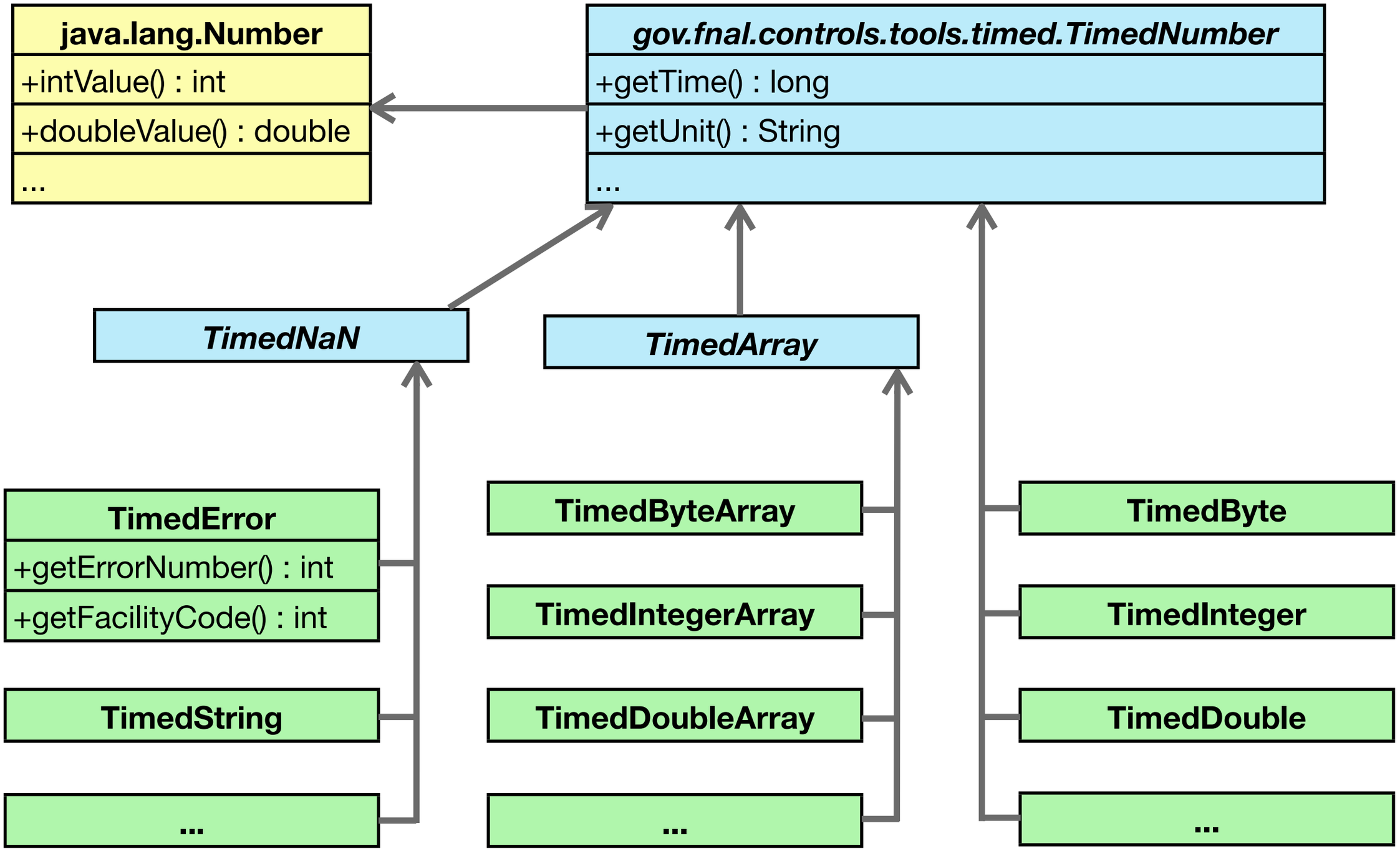
Data Types

Low-level model: **gov.fnal.controls.proto.DAQData.Reply.***	High-level model (recommended): **gov.fnal.controls.tools.timed.***
`DoubleSample` ⟷	`TimedDouble`
`IntegerSample` ⟷	`TimedInteger`
`DoubleArraySample` ⟷	`TimedDoubleArray`
`AnalogAlarmSample` ⟷	`TimedAnalogAlarm`
`...`	`...`

`interface DataCallback {`
 	
`void dataChanged(`
`String dataRequest,`
`Reply data);`
 	
`}`
	`abstract class TimedNumberCallback`
	`implements DataCallback {`
	
	`abstract void dataChanged(`
	`String dataRequest,`
	`TimedNumber data);`
	
	`}`

To convert between the two models, use **gov.fnal.controls.tools.timed.TimedNumberFactory**	

`TimedNumber`'s



Example of a Data Callback Listener

```
DataCallback lis = new TimedNumberCallback {

    @Override
    public void dataChanged(String req, TimedNumber data) {
        double val = data.doubleValue();
        double xyz = (val - 1.0) / 3.1415;
        System.out.println("XYZ=" + xyz
            + " Timestamp=" + data.getTimeAsString());
        if (data instanceof TimedError)
            System.out.println("This was an error.");
    }

}
```

Special Error Codes

```
package gov.fnal.controls.service.dmq.impl;

public class DAQStatus {

    public static final int DMQ_FACILITY_CODE = 72;

    public static final int PENDING = 1;
    public static final int OK = 2;
    public static final int SECURITY_VIOLATION = -99;
    // ...

}
```

Positive codes are "good" ones.

Reading Callback Sequence

```
$ java -cp rabbitmq-client.jar:commons-io.jar:split/kerberos-
client.jar:split/dmq-client.jar
gov.fnal.controls.service.dmq.example.SimpleReading m:outtmp@p,1000

An example of simple reading via DMQ framework
by Andrey Petrov <apetrov@fnal.gov>

Exit with ^C.

    >>> null «72 1» Request Pending 2011-04-28T11:22:34.078-0500

    >>> m:outtmp@p,1000  42.30954535841943 DegF 2011-04-28T11:22:34.797-0500

    >>> m:outtmp@p,1000  42.30954535841943 DegF 2011-04-28T11:22:35.797-0500

    >>> m:outtmp@p,1000  42.30954535841943 DegF 2011-04-28T11:22:36.797-0500
```

Setting Callback Sequence

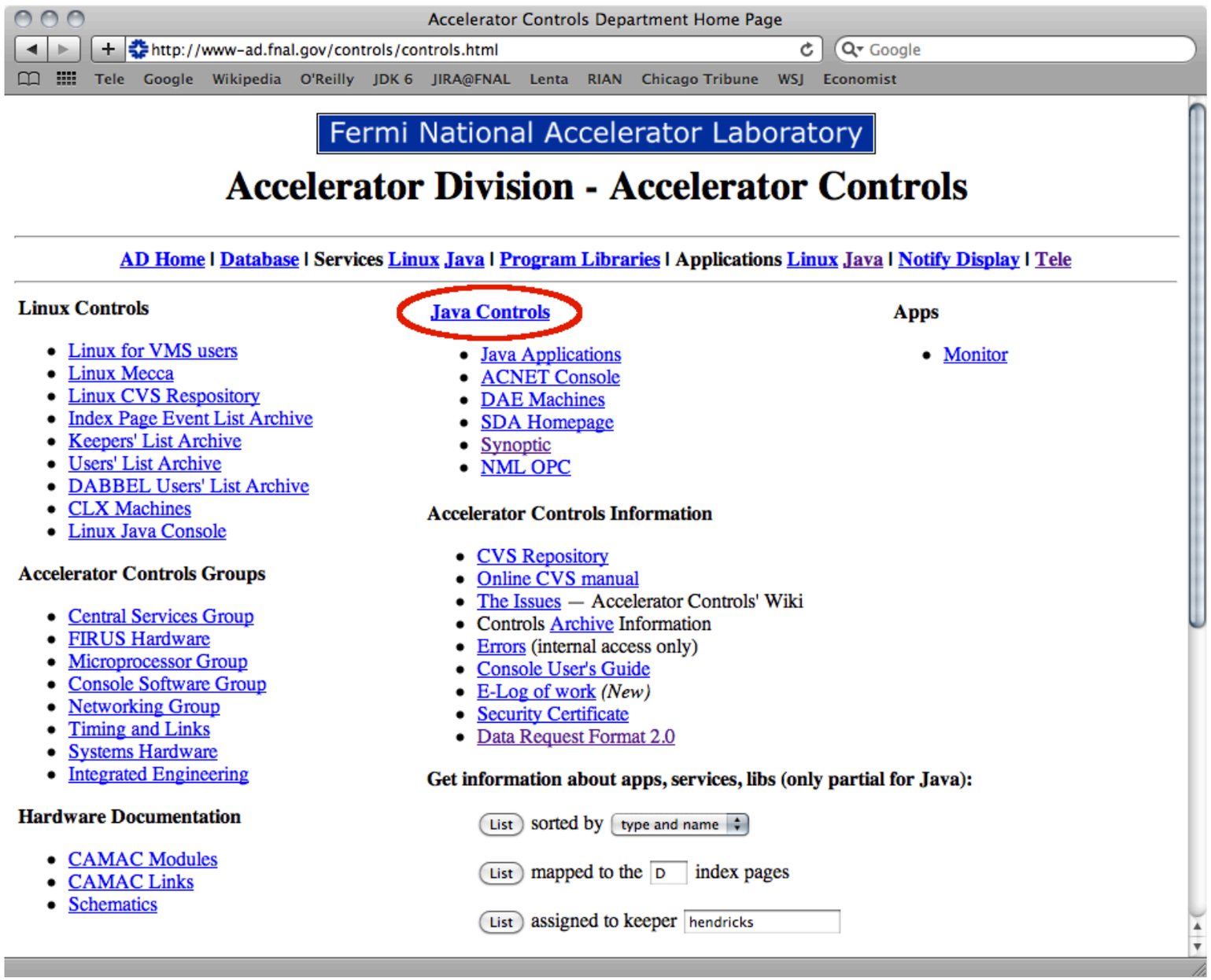
```
$ java -cp rabbitmq-client.jar:commons-io.jar:split/kerberos-
client.jar:split/dmq-client.jar
gov.fnal.controls.service.dmq.example.SimpleSetting z_cache

An example of simple setting via DMQ framework
by Andrey Petrov <apetrov@fnal.gov>

Exit with ^C.

   >>> null «72 1» Request Pending 2011-04-28T13:27:30.043-0500

Value: 56
Set 56.0 to [z_cache]:
Value:

   >>> z_cache 56.0 2011-04-28T13:27:39.384-0500
```

Various Issues

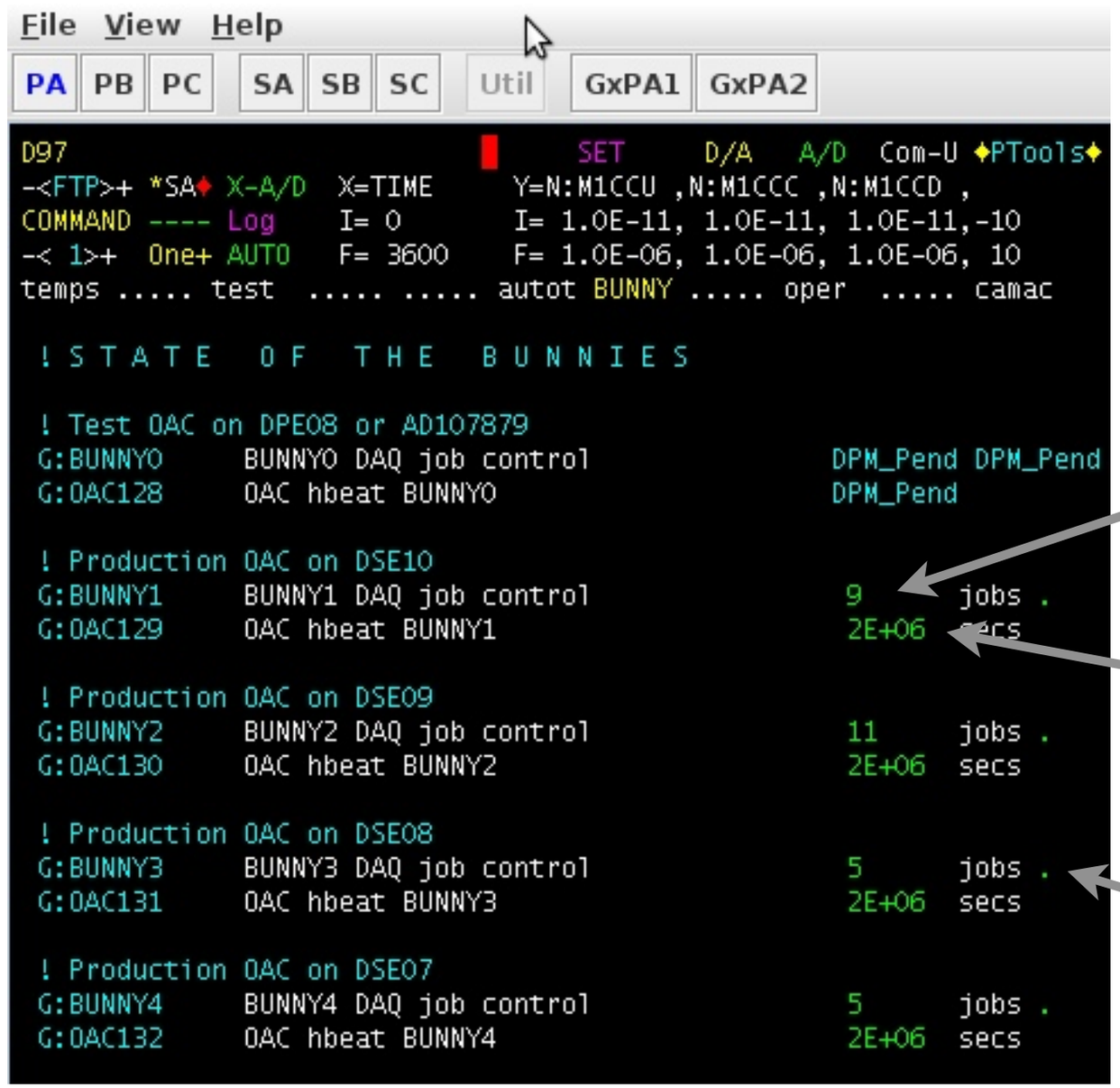
- Not supported: Data Loggers, FTP, Snapshot Data.
- Not supported: Long device names.
- Supported: Device descriptions (“I~BEAM”).
- In a setting job, properties matter: “Z_CACHE” or “Z:CACHE.SETTING”, not “Z:CACHE”.
- All 3 scalings are supported [most of the time].

Diagnostics Over the Web

http://www-bd.fnal.gov/telescope/bunny



Diagnostics via Parameter Page



Number of jobs

Uptime

ON/OFF status
(can be controlled)

The End