General Utilities

CONFIGURATION

- In addition to usual properties files, many newer systems particularly where configuration needs to be quite structured use XML config files.
- Classes which need this type of configuration can implement interface *ngat.util.XmlConfigurable*
  - Public void configure(Element node)
  - Use JDOM to extract the root node or some lower level node and push this to the class via the configure method to allow it to extract info from it and any sub-nodes and attributes.
- Convenience class *ngat.util.XmlConfigurator*
  - Static chained call, passing java.io.File to extract root node
    - *XmlConfigurator.use(File).configure(XmlConfigurable)*
- Used in most newer systems
  - TCM – *telescope.xml*
  - ICM – *ireg.xml*
  - ERS – *rules.xml*

LOGGING

- Most systems use *ngat.util.logging* package.
- Loggers are obtained using static
  - LogManager.getLogger(name)
    - Attach LogHandler to logger, determines where logging stream is directed.
      - FileLogHandler (File system)
      - SocketLogHandler (TCP socket destination)
      - ConsoleLogHandler (System.err)
    - Attach LogFormatter to handler to determine output formatting.
      - CsvLogFormatter
      - StandardLogFormatter
      - BasicLogFormatter
      - XmlLogFormatter

Log Record

- The basic entity created is a LogRecord
  - String message – the actual text.
  - String loggerName – name of this logger.
  - String clazz – name of class where call is made.
  - String source – ID of object where call is made.
  - String method – current method where call is made.
  - String thread – current thread.
  - long time – timestamp.
  - int seqno – log sequence no.
  - int level – importance/verbosity level.
  - Object[] params – some extra data.

Log Publish

- A series of overloaded and complicated calls to generate varying degrees of LogRecord complexity, i.e. which of the fields are actuall filled in…
  - log(int level, String message)
  - log(int level, Exception e)
  - log(int level, Object[] params)
  - log(int level, String message, Exception e)
  - log(String cat, int level, String message)
  - log(String cat, int level, String message, Exception e)
  - log(int level, String clazz, String message)
  - log(String cat, int level, String clazz, String message)
  - log(int level, String clazz, String source, String message)
  - log(String cat, int level, String clazz, String source, String message)
  - log(int level, String clazz, String source, String method, String message)
  - log(String cat, int level, String clazz, String source, String method, String message)
  - log(String cat, int level, String clazz, String source, String method, String message, Object[] params, Exception exception)

Extended Logging

- A new system was introduced to overcome this complexity and add additional information into the records.
  - *ngat.util.logging.ExtendedLogRecord*
- Parameters in extended record.
  - long time – Timestamp.
  - String system – Name of the originating system.
  - String subSystem -Name of the originating sub-system.
  - String srcCompClass – Name of originating class.
  - String srcCompId – Name/ID of originating object.
  - String block – ID of block or method name.
  - int severity – Severity level
    - (FATAL, CRITICAL, ERROR, WARNING, INFO)
  - int level - Verbosity level (1 High, 5 Low).
  - String theme – User-defined high level category for message.
  - String category – User-defined lower level category for message.
  - String message – Message text.
  - String thread – Calling thread.
  - Map context – Map of key/value context information.

Log generator

- Extended log records are published differently.
- To avoid confusion between the various overloaded calls, all calls to extended records can be chained, so that as much or as little info can be included as required.
- Key classes are *LogCollator* and *LogGenerator*.
  - LogGenerator stores the identification information about the source object and typically setup in constructor…
  - myLogGenerator

```
    .system("RCS")
    .subSystem("TCM")
    .srcCompClass(this.getClass().getName())
    .srcCompId(this.name);
```

LOG COLLATOR

- Log collator is an object used to build up a log record ready to publish. It is created by a generator using a call to:-
  - LogGenerator.create();

- Can now chain any of the various methods to build up an extended log record…
  - *block(String block)* – Set the method or block name.
  - *severity(int severity)* – Set the severity level.
  - *level(int level)* – Set the verbosity level.
  - *theme(String theme)* – Set the theme.
  - *category(String category)* – Set the category.
  - *msg(String message)* – Set the message text.
  - context(String key, String value) – Add a context entry.
  - *fatal()* – Shortcut, set severity FATAL.
  - *critical()* – Shortcut, set severity CRITICAL.
  - *error()* - Shortcut, set severity ERROR.
  - *warn()* - Shortcut, set severity WARNING.
  - *info()* - Shortcut, set severity INFO.
  - *extractCallInfo()* – Fill in call information from stack.

- Finally call *send()* to actually publish the record to its attached Logger:-
  - loggen.create().info().level(2).block("connect").msg("Connecting to: "+host).send();
  - loggen,.create().warn().level(1).msg("Slow connection").send();
  - loggen.create().level(2).msg("Testing").send();

FULL EXAMPLE

```
// Create and setup Logger
Logger logger = LogManager.getLogger("TCM");
logger.setLogLevel(3);
logger.addHandler(new ConsoleLogHandler(new
    BasicLogFormatter(100));

// Create LogGenerator and setup.
LogGenerator loggen =
    logger.generate()
            .system("TCM")
            .subSystem("Telescope")
            .srcCompClass(getClass().getSimpleName())
            .srcCompId("Scope");
// Make a logging call
loggen.create().info().level(1).extractCallInfo().msg("Starting server").send();
```

Extract Call Info

- Method *LogCollator.extractCallInfo()* pulls data from the stack for the current logging call and populates the *LogCollator* with this information.
- It is potentially expensive but so far does not appear to have had any spurious effects.
- It should probably be used sparingly i.e. not used in logging calls which are made hundreds/ thousands of times per second.
- Information extracted includes:
  - Class name.
  - Calling thread.
  - Method name.
  - Source line number if available.

ASTROMETRY

- The astrometry package is used extensively in many places including:-
  - RCS
  - Scheduler
  - OpsUI
  - Phase2UI
  - Simulator
  - TEA
  - NSO Interface
- There are 2 versions of this package:-
  - Old system is quite cumbersome to use.
  - Newer system is simpler and more powerful.
  - Any recent software uses the new system exclusively and many older systems have been updated.
  - There are still chunks of old astro lib code dotted about.
- Both systems make use of Slalib via JNI calls.
  - This part of both systems is the most troublesome.
  - Neither set of JNI code have been successfully compiled for 64 bit linux.
- There are a small number of functions available in the new system which do not use Slalib so as to be available in Phase2UI.

NEW LIBRARY

- New library: *ngat_new_astrometry.jar*
- *AstrometryCalculator* and *AstrometrySiteCalculator*
  - Main interfaces of interest.
  - Provide methods to find alt, az, rise and set, transit elevation etc of targets.
- Implemented as *BasicAstrometryCalculator a*nd as *BasicAstrometrySiteCalculator*.
- *Coordinates* are a representation of a target's position on the celestial sphere at a given instant.
  - getRa(): double
  - getDec(): double
- *TrackCalculator* is a representation of a target's track over a period of time.
  - getCoordinates(long time): Coordinates
- Various implementations:-
  - *SolarCalculator* – sun track.
  - *LunarCalculator* – moon track.
  - *BasicTargetCalculator* – track for general target.

ASTROMETRY METHODS

BasicAstrometrySiteCalculator

- boolean canRise(Coordinates coord, double horizon, long time)
- boolean canSet(Coordinates coord, double horizon, long time)
- double getAltitude(Coordinates coord, long time)
- double getAzimuth(Coordinates coord, long time)
- double getHourAngle(Coordinates c, long time)
- double getMaximumAltitude(TargetTrackCalculator track, long t1, long t2)
- double getMinimumAltitude(TargetTrackCalculator track, long t1, long t2)
- long getTimeSinceLastRise(Coordinates coord, double horizon, long time)
- long getTimeSinceLastSet(Coordinates coord, double horizon, long time)
- long getTimeSinceLastTransit(Coordinates coord, long time)
- long getTimeUntilNextRise(Coordinates coord, double horizon, long time)
- long getTimeUntilNextSet(Coordinates coord, double horizon, long time)
- long getTimeUntilNextTransit(Coordinates coord, long time)
- double getTransitAltitude(Coordinates coord, long time)
- boolean isRisen(Coordinates coord, double horizon, long time)
- boolean isSet(Coordinates coord, double horizon, long time)
- double getParalacticAngle(Coordinates c1, long time)
- double getAngularSeperation(Coordinates c1, Coordinates c2)
- double getClosestPointOfApproach(TargetTrackCalculator trk1, TargetTrackCalculator trk2, long t1, long t2)

Examples

```
// Setup calculator for LT. approx site
final double LT_LAT = Math.toRadians(28.0);
final double LT_LONG = Math.toRadians(-17.0);
ISite lt = new BasicSite(“Liverpool Telescope”, LT_LAT, LT_LONG)
AstrometrySiteCalculator astro = new BasicAstrometrySiteCalculator(lt);

// Obtain sun Ra, dec,  elevation in 30 minutes
SolarCalculator sun = new SolarCalculator();
long nowPlus30 = System.currentTimeMillis() + 30*60*1000L;
Coordinates sun30 = sun.getCoordinates(nowPlus30);
double sunRa30 = sun30.getRa();
double sunDec30 = sun30.getDec();
double sun30Height = astro.getAltitude(sun30,  nowPlus30);

// How close is an NEO ephemeris target to the moon ?
XEphemerisTarget neoEphem;
LunarCalculator moonTrack = new LunarCalculator(lt);
TargetCalculator neoTrack = new BasicTargetCalculator(neoEphem, lt);
Coordinates neo = neoTrack.getCoordinates(now);
Coordinates moon = moonTrack.getCoordinates(now)
Double distRads = astro.getAngularSeperation( neo, moon);
```

Additional interfaces

- NonSiderealTrackingCalculator
  - Provides tracking rates for NS targets
    - getNonSiderealTrackingRates(TargetTrackCalculator track, long time) : TrackingRates
- CardinalPointingCalculator
  - Provides information on cardinal pointing and transformations between sky and mount angles.
    - isFeasibleSkyAngle(double skyAngle, ITarget target, double instrumentOffset, long t1, long t2) : boolean
    - getBestCardinalAngle(ITarget target, double instrumentOffset, long t1, long t2) : double
    - getMountAngle(double skyAngle, ITarget target, double instrumentOffset, long time) : double
    - getSkyAngle(double mountAngle, ITarget target, double instrumentOffset, long time) : double