

GUSTAF Implementation Details

Joshua Shapiro

QSO-020/Version 1.2/2002-01-21

Contents

1 Paradigm	2
1.1 Application Responsibilities	2
1.2 Database Responsibilities	2
2 Database	2
3 Application	4
3.1 Object Overview	6
3.2 Object Details	10
3.2.1 Outstanding Issues	22
3.3 Interface Details	22
4 Project Timeline	23

List of Tables

1 Overview of <code>cfht.gustaf</code> package	8
2 Additions to <code>cfht.common.astronomy</code> package	9
3 Additions to <code>cfht.common.data</code> package	10
4 Behavior of interface	23

List of Figures

1 Flowchart for GUSTAF in two distinct modes	5
2 A mock screen shot of the GUSTAF application	11

1 Paradigm

The application and database will jointly share responsibility for identifying and saving guide stars. I propose the following responsibilities for the two softwares.

1.1 Application Responsibilities

- User Interface
- Identification and ranking of Valid Guide Stars
- Determination of STATUS of pointings, where STATUS identifies if there are valid guide stars.
- Scaling of dither patterns by IC scale factor.

1.2 Database Responsibilities

- Save guide star coordinates and rank
- Provide unique lists of apparent telescope positions
- Provide lookups of dither pattern offsets.
- Provide information on instrument geometry (Optional)

2 Database

The ph2 or op database will require new tables in which to save guide stars, and new stored procedures for inserting, selecting, modifying and deleting those guide stars.

The following stored procedures will be required.

- update guide star given
 - xtarget
 - Dither Pattern
 - Offset ID
 - DP scale factor ?
 - Guider Value
 - guide star position

- guide star rank
- guide star visual mag
- guide star catalog
- guide star name
- guide star proper motion (MAS/yr)
- guide star id (=0 for insert =id for update)
- Delete guide star id
- Select guide stars given xtarget, DP ID, and DP offset
- Select xtargets and DPs given runid pattern or agency
- Select dither patterns given instrument

The stored procedure to retrieve guide stars given an xtarget and dither pattern information should return the following columns.

- Guide star right ascension α
- Guide star declination δ
- Guide star magnitude
- Guide star rank
- Guide star ID (numeric or varchar)
- GUIDER ID (to identify North or South guider)

The stored procedure to retrieve apparent telescope position and dither pattern IDs should include the following columns in the result set.

- xtarget right ascension α
- xtarget declination δ
- xtarget ID (possibly combination of targetID and offsetID)
- Dither Pattern ID

- Dither Pattern Name
- Dither Pattern scale factor

The stored procedure to select dither patterns given instrument should return the following columns in the result set.

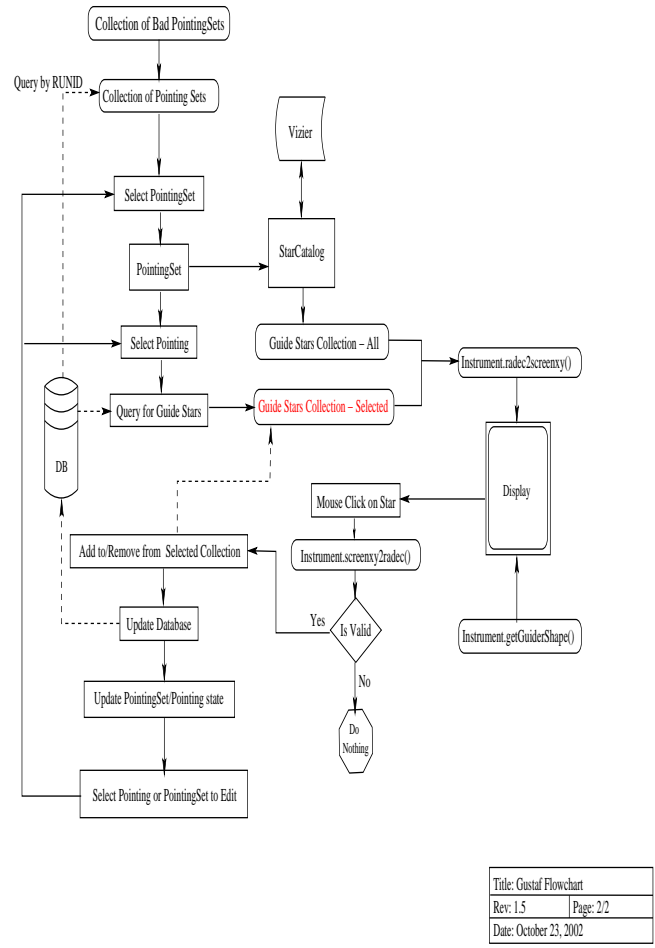
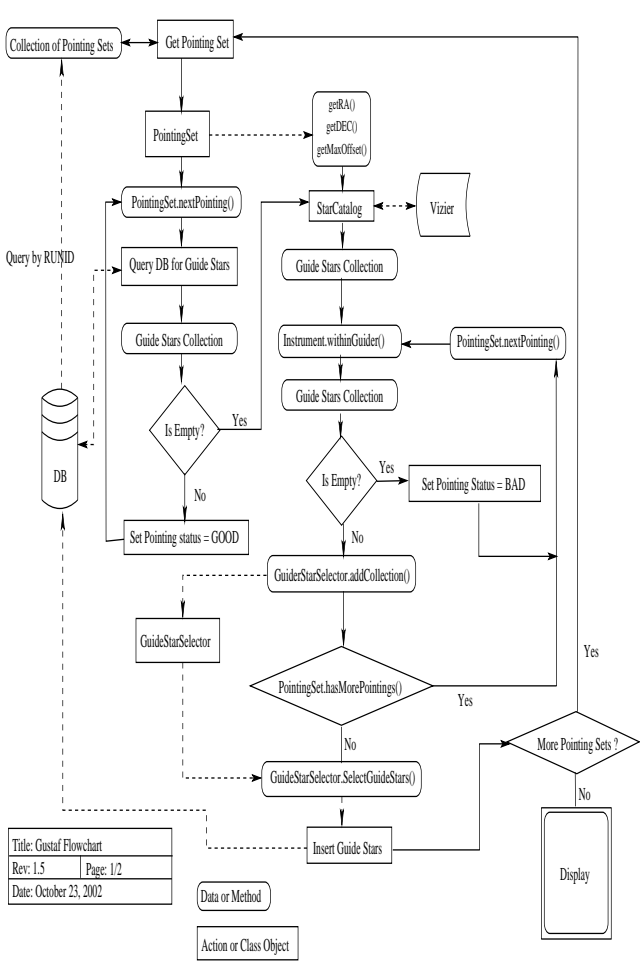
- Dither Pattern ID
- Dither Pattern Name
- Number of offsets
- offset East-West (arcseconds)
- offset North-South (arcseconds)
- offset sequence number
- offset id

3 Application

A general flowchart for the gustaf application is included in *CFHT Memo QSO_019*, and is included in figure 1. A general discussion of the flowchart is as follows:

- The application will create a list of `Pointing Sets` by querying the database for `xtarget` information and correlating this with dither pattern offsets stored in lookup tables and instrument geometry either hardcoded in Java or looked up from the database.
- The `Pointing Sets` will act like iterators and allow the application to sequentially iterate through `Pointings` looking up guide stars. Each guide beam will be treated as an individual `Pointing`, so there will actually be $2 \times \# \text{ of } DP \text{ offsets}$ pointings per `PointingSet`.
- The application will first look in the database for guide stars, and then use star catalogs to choose guide stars if none are found in the database.
- A specialized module called the `GuideStarSelector` will take as input the list of possible guide stars for many pointings, and be responsible for sorting and ranking them for a `Pointing Set`.

Figure 1: Flowchart for GUSTAF in two distinct modes



- The application will determine a PointingSet status based on the available guide stars for individual pointings, and update a table cell in the interface.
- The user will be able to view plots of individual pointings by selecting a row and a pointing in the table. Once the instrument geometry has been rendered along with guide stars, the user will be able to add, delete, or change the rank of stars by selecting them in the plot and clicking on the appropriate button in the interface. The application will handle this by, deleting, then re-inserting the guide stars into the database.
- Data Objects will be compatible with Data used in the QSO Tools so that a user can launch GUSTAF from the Observing Tool and have it be populated with data from the active queue. This way guide stars can be modified at time of observation in case they need to be changed due to catalog errors or other unforeseen problems.

3.1 Object Overview

The application will be written in Java using swing components. GUSTAF will require the creation of or addition to the following packages in the CFHT java source tree.

- cfht.gustaf
- cfht.common.astronomy
- cfht.common.data

Below are a summary of the classes and interfaces added to each package.

cfht.gustaf	
Class	Description
BasicGuideStarSelector	A basic implementation of GuideStarSelector. It will be responsible for sorting and ranking collections of viable guide stars for a Pointing Set using simple algorithms to select the brightest stars closest to the center of the guiders.

Class	Description
DBProxy	This class is a proxy class for communicating with the database. It will field requests for Data Objects and build them from database queries.
Gustaf	The main executable class. This class will build the GUI, and manage the overall flow of data between different components.
GustafOptions	A class to manage a user interface which modifies parameters in the application such as which star catalog, and plot colors.
GuideStar	child of <code>Star</code> which also includes rank, database ID, and any other data specific to guide stars.
Interface GuideStarSelector	Implementing classes will be able to take several unsorted collection of stars which could be used as guide stars, and determine the best ones to use for each pointing.
Pointing	A Pointing is an individual α, δ for a single Instrument beam.
PointingSet	A Pointing Set is a collection of pointings defined by the telescope main beam position α, δ , a sequence of offsets from the dithering pattern, and fixed offsets for each guider. The set will have <code>Iterator</code> like methods allowing the application level classes to loop through the pointings easily.

Class	Description
SearchThread	The process which queries and searches for guide stars will be spawned in a separate thread because of its network dependencies and potentially long search times.
Abstract SkyPlot	This abstract class will be used by the application for plotting the Instrument and guide stars on a map of the sky. This class will extend a JComponent so that it can be easily added to a container. Subclasses should only need to implement the render method to define the way they are to be plotted.
SkyPlotPanel	This class is intended to abstract the details of rendering a plot away from the Gustaf application. The application will only have to set the current pointing details for the proxy and the proxy will manage the details such as units, plot size, colors and layout This proxy will then utilize classes which implement the SkyPlot interface to actually draw to the graphics objects.
XTarget	A class representing the telescope position, i.e. Target position plus Target Offset.

Table 1: Overview of `cfht.gustaf` package

cfht.common.astronomy	
Class	Description
Star	A class representing a guide star position and magnitude.
Interface StarCatalog	The application will use this interface to query for guide stars. This allows for future implementations of different catalogs if necessary. It will likely require only a single method <code>getStarsAt()</code> which takes telescope position and parameters specifying magnitude and search box size.
StarCatalogFactory	A Factory class for generating classes which implement the StarCatalog interface.
VOCatalog	A class which implements the StarCatalog interface. This catalog will use Vizier at CDS or one of its mirrors and build collections of stars given celestial coordinates and query parameters such as maximum magnitude and search box size.
Angle	An angle. Can be defined to be signed or unsigned, and provide methods for doing angle arithmetic
Position	A spherical position defined by two angles longitude and latitude. Includes methods for doing position arithmetic.
Units	A library of unit definitions and conversions

Table 2: Additions to `cfht.common.astronomy` package

cfht.common.data	
Class	Description
DitherPattern	A class representing a Dither Pattern.
Instrument	A class representing a physical instrument on the telescope. An instrument will have knowledge of its own geometry and optics so that it can determine if a celestial object falls within its borders. Instruments can have child instruments, for instance, MegaCam will have two children representing a North and South guider, and 36 children representing each chip.

Table 3: Additions to `cfht.common.data` package

3.2 Object Details

This section contains a basic outline of the methods for some of the major classes, and descriptions of how they will operate.

class DBProxy

This class will be the applications interface to the database. It will have methods which execute sql and return java objects.

```

/* Get a collection of PointingSets given a runid pattern */
Collection getPointingSets(String runid_pattern)

/* Get guide stars, if any, for a Pointing from DB */
Collection getGuideStars(Pointing p)

/* Insert guide stars for a pointing from DB.
 * This method will have to extract the xtarget, dither pattern
 * and guider information from the pointing.
 */
boolean insertGuideStars(Pointing p, Collection guideStars)

/* Delete guide stars for a pointing from DB*/
boolean deleteGuideStar(Pointing p, GuideStar guideStar)

```


3. column to use for ra,dec
 4. column to use for proper motion
 5. column to use for class
 6. constraints on above columns
- Plot
 1. plot colors, (foreground, background, selection)
 - GuideStarSelector
 1. Select algorithm to use and tuning? options.
 2. Describe # of guide stars required for computed status.

GuideStar

This class will extend a `Star` and provide storage and access methods for data specific to guide stars. While some of this information remains to be determined by the final design of the stored procedures, it will include:

- Guide Star ID
- Guide Star Rank

interface GuideStarSelector

Implementing classes are the brains of the GUSTAF . There is the opportunity here to make the selection process as smart or stupid as we like.

The guide star selector, according the mandate of document *QSO_019* needs to intelligently choose guide stars with foreknowledge of the dither patterns. To accomplish this, the `GuideStarSelector` will need to be populated with `Star Collections` for each `Pointing` in a `PointingSet`. When this is complete, the algorithm to select the guide stars will be invoked, after which the ranked collections of guide stars can be extracted and used to populate the database.

```

/* Add a collection of Stars (Not Guide Stars) which fall within the
 * Field of View of the instrument associated with Pointing p.
 */
addCollection(Pointing p, Collection stars)

/* Determine the best guide stars to use for each collection that
 * has been added with the \texttt{add()} method. Pointings are
 * assumed to all be members of the same pointing set. This means
 * that they are related by a dither pattern. NOTE: pointings are

```

```
* and in fact should have duplicate sequence numbers. This is
* because each guider is an independent pointing, but they
* correspond to the same step in the dither pattern.
*/
selectGuideStars()

/* Return the collection of GuideStars following a call to
* \texttt{selectGuideStars()}
*/
getCollection(Pointing P)

/* Reinitialize the selector for a new pointing set. */
clearAll()
```

class Pointing

The Pointing class represents a position centered on the instrument associated with the pointing, given the target and dither pattern offset of the parent PointingSet.

```
/* Get the xtarget RA + instrument RA offset + DP RA offset */
Angle getRA()

/* Get the xtarget DEC + instrument DEC offset + DP DEC offset */
Angle getDEC()

/* Get the sequence number of this pointing in the dither pattern.
* Multiple pointings with the same sequence
* number are allowed for a single pointing set. This allows
* a Pointing to be defined for each guider of the instrument
*/
int getSequenceNumber()

/* get the instrument associated with this Pointing */
Instrument getInstrument()

/* get the XTarget associated with this pointing */
XTarget getXTarget()

/* get the Dither Pattern associated with this pointing */
DitherPattern getDitherPattern()
```

class PointingSet

This class will be constructed by GUSTAF inside DBProxy from database stored procedures. Alternately, it can be constructed by the interface between GUSTAF and the Observing Tool.

The idea is that the application will be able to iterate through each pointing and query the database to see if guide stars exist. The first encounter with an empty collection of guide stars will prompt GUSTAF to use a StarCatalog to get a collection of stars centered on the getRA() and getDEC() and within bounds defined by getMaxOffsetRA() and getMaxOffsetDEC().

```
/* Constructor - A pointing set will be initialized with an
 * XTarget, a DitherPattern, and an Instrument.
 * At initialization, the child instruments which are guiders
 * will be extracted and cached.
 */
PointingSet(XTarget xt, DitherPattern dp, Instrument inst)

/* Return the number of pointings in this set */
int getNumPointings()

/* Get the Nth pointing */
Pointing getPointing(int)

/* Get the xtarget RA + guider RA offset */
Angle getRA(Instrument inst)

/* Get the xtarget DEC + guider DEC offset */
Angle getDEC(Instrument inst)

/* Get the number of guiders */
int getGuiderCount()

/* Get a specific guider, which is really an instrument */
Instrument getGuider(int i)

/* Get the maximum East West offset in the Dither Pattern */
Angle getMaxOffsetRA()

/* Get the maximum North South offset in the Dither Pattern */
Angle getMaxOffsetDEC()

/* Set the pointing status (Good or Bad)
 * for a particular pointing
 */
```

```
void setStatus(int pointingIndex, String|int status)

/* Get the XTarget for this Pointing Set */
XTarget getXTarget()

/* Get the Dither Pattern */
DitherPattern getDitherPattern()

/* Get the status for all pointings */
String|int getPointingStatus()
```

class SearchThread

A new instance of this class will spawn a thread which accepts a (Collection of?) PointingSet, Catalog, Instrument, DBProxy, and GuideStarSelector as input and goes through the selection process. This thread must execute the following process:

- Search DB for guide stars for this pointing
- Determine pointing set status.
- Decide whether or not to continue searching for guide stars and if so.
- Define search regions from pointing set and instrument.
- Search StarCatalog for stars.
- Select subset of stars which are viable (i.e. within guider bounds).
- Add the star set and associated pointing to the guide star selector.
- (Execute Guide Star Selector filtering process?)
- (Commit to database)

In addition, the thread must safely handle interrupts during any of the above steps and return the application and database to a stable state. Logical stopping points are:

- After “Determine Pointing Set Status”
- Before “Search StarCatalog for stars.”
- After “Commit to database.”

The thread must also assume the responsibility for informing the application what state a search is in, (SEARCHING, COMPLETE, HALTED ?), so that the GUI can be updated appropriately.

SkyPlotProxy

This class will abstract the details of managing the SkyPlots out of the Gustaf application. This class must have access to GustafOptions so that it can extract user specified parameters such as plot type and colors.

This class will also make use of a StarCatalog to select a group of stars to be plotted in addition to guide stars. It would be smart to load these stars in a separate thread.

```
// The following two methods will be used by \thetool to
// specify which pointings to render.
```

```
/**
 * Set the PointingSet to display with default step of 0
 */
public void setPointingSet(PointingSet p);
```

```
/**
 * Set the dither pattern step within the pointing set.
 * @param i Is a number between 1 and the maximum number of dither pattern steps
 */
public void setDitherPatternStep(int i);
```

```
// There will also need to be internal methods for handling
// the skyplot objects.
```

sequence of events

- o determine plot center
 - The plot center should = pointing pos + instrument offset + avg instrument bo
- o set plot instrument & color
- o get plot width and height
- o query for stars in separate thread
 - set plot stars and color on callback
 - render on callback
- o set plot guide stars and color
- o render

AbstractSkyPlot

Implementing classes will have the ability to draw an instrument geometry and celestial objects. It is assumed that users will interact with the plot to modify the guide star selections.

The abstract class will extend a javax.swing.JComponent so that subclasses can be added to Containers and placed within GUSTAF .

```
/* set the plot title */
  setPlotTitle(String s, Color c);

/* set the instrument to render
 * pos will be the origin which inst.getOffset() refers to
 */
  setInstrument(Position pos, Instrument inst, Color c)

/* The plot reference will be used as the reference
 * position when rendering the stars. There are no restrictions
 * on the value of pt or pos.  u are the units of point pt.
 *
 * Maybe we don't need this . We want the plot to determine its
 * own reference pixel for position.  If the position specified in
 * setInstrument indicates the ``origin'' of the instrument, the plot
 * can decide where to place the instrument on the graphics device.
 */
#  setPlotReference(Point pt, Units u, Position pos)

/* plot a collection of stars */
  setStars(List stars, Color c)

/* plot a collection of guide stars */
  setGuideStars(List guideStars, Color c)

/* get a collection of guide stars
 * This is useful if the user has modified
 * the guide star collection using the plot
 */
  getGuideStarList()

/* Get whichever star is selected in the
 * plot. return null if none are selected.
 */
  getSelectedStar()

/* Get the computed width of this plot
 * in arcminutes. Should be = n * instrument width
 */
  getPlotWidthArcminutes()

/* Get the computed height of this plot
```

```

* in arcminutes. Should be = n * instrument height
*/
getPlotHeightArcminutes()

/* The paintComponent() method will be implemented
* to call this method .
render(Graphics2D)

```

class XTarget

The XTarget class represents a telescope position defined by target coordinates (α, δ) plus user offset coordinates $(\Delta\alpha, \Delta\delta)$. A single target can have multiple user offsets, therefore a special ID will have to be defined for indicating the target/offset combination when inserting guide stars in the database.

```

/* Return the apparent telescope right ascension */
Angle getRA()

/* Return the apparent telescope declination */
Angle getDEC()

/* Return the xtarget id */
Angle getID()

/* Return the program associated with this XTarget.
* While it is possible that two programs could have
* the same XTarget, we will ignore that possibility
* in the application.
*/
Program getProgram()

```

interface StarCatalog

An interface with a single method for querying a star catalog. The searchSize parameter will be radius for circular searches and side for box searches. The searchArea parameter will be an integer code specifying a box or circle search.

```

/* Return a collection of Stars */
Collection getStarsAt(Position p,
                    float magUpperLimit,
                    float magLowerLimit,
                    float searchSize,
                    int searchArea)

```

class StarCatalogFactory

A factory class will simplify the creation of StarCatalogs from the perspective of the application. The application can specify a Catalog to use and allow the star catalog to handle the details of creating the specific object for querying that catalog.

```
/* Return the available catalog names
 * as strings in a collection.
 */
Collection getCatalogs()

/* Return a specific catalog given
 * its name.
 */
StarCatalog getCatalog(String name)
```

class VOCatalog

This is currently the only planned implementation of StarCatalog. It will use the vizier online catalogs at CDS or one of its mirrors to return stars. It will use the parsing classes provided by CDS to parse the outputs of a vizier query. Depending on how it is invoked it can access different catalogs.

class DitherPattern

This class represents a dither pattern. Because there are a limited number of possible dither patterns, they will all be looked up at the start of the application. The same dither pattern instances will be used by all pointing sets. For security, this class will have no setXXX() methods.

abstract class Instrument

This class is an abstract class. The only method which will not be abstract is draw(), which will have the default behavior described below.

```
/* Get type of instrument, IMAGER, GUIDER, SPECTROGRAPH */
String|int getInstrumentType()

/* test if instrument has children */
boolean hasChildren()
```

```
/* get child instruments, if they exist, as a Collection */
    Collection getChildren()

/* get the offset from the telescope main beam
 * to a reference position on this instrument
 */
    Position getOffset()

/* get the bounds of the instrument */
    Shape getBounds()

/* get the units of the offset and bounds */
    Units getUnits()

/* Project celestial coordinates to x,y
 * at the focal plane of the instrument.
 * IT IS IMPORTANT that this method accurately convert
 * to virtual coordinates for child instruments. Presumably
 * the equation used for projecting position P to the focal
 * plane, will be defined for the main beam of the telescope.
 * An instrument can represent a position relative to the
 * main beam of the telescope.
 * This MUST be considered when performing the operation.
 */
    Point2D celestialToVirtual(Position p)

/* Determine if the position is within the bounds of this instrument.
 * The same warning applies to this method as celestialToVirtual(),
 * but this method will likely make use of celestialToVirtual()
 * anyway.
 */
    boolean withinBounds(Position p)

/* render this instrument to the graphics object at the
 * specified pixel using a scaling factor of units/pixel.
 * The default behavior will be to call this method recursively
 * on this instruments children or to draw the bounds if no
 * children exist.
 */
    void draw(Graphics g, int x, int y, double scale)
```

class Angle

A class to abstract the details of handling angles. Spherical Coordinates have angles in two different formats.

- Signed angles run from -180 to +180
- Unsigned angles run from 0 to +360

This class will correctly wrap the angles when boundaries are crossed and return the value in the requested format.

```
/* Return the angle as a degree in double percision */
double getDegrees()
```

```
/* Return the angle as a radian in double percision */
double getRadian()
```

```
/* Add this angle to another angle. The sense of this
 * angle will be preserved.
 */
Angle add(Angle a)
```

```
/* Subtract another angle from this angle. The sense
 * of this angle will be preserved.
 */
Angle subtract(Angle a)
```

```
/* Return the cosine of this angle */
double cos()
```

```
/* Return the sine of this angle */
double sin()
```

```
/* Return the tangent of this angle */
double tan()
```

class Position A class representing a position in Spherical Coordinates. Composed of two Angles, a signed angle for latitude, and an unsigned angle for longitude. This class will support arithmetic of positions and abstract the complexities at boundaries and poles.

```
/* Return the longitude like angle */
Angle getLongitude()
```

```
/* Return the latitude like angle */
Angle getLatitude()
```

```

/* Return the right ascension, or longitude */
Angle getRightAscension()

/* Return the declination, or latitude */
Angle getLatitude()

/* Add the two longitudes together and add the two latitudes together. */
Position add(Position p1, Position p2)

/* Subtract the two longitudes lon1 - lon2 and the
 * two latitudes lat1 - lat2
 */
Position subtract(Position p1, Position p2)

/* Return the distance between two positions as an unsigned angle. */
Angle distance(Position p1, Position p2)

```

class Units This class will define conversions between commonly used units, and work as a utility for objects, such as instruments, to communicate the units in which they are defined. The implementation will be created as needed.

3.2.1 Outstanding Issues

- Should a Collection of guide stars, once identified, be saved with a Pointing or a PointingSet?
- What are the possible statuses for a pointing? How are the statuses for many pointings to be combined to give an overall status for a pointing set?

Item 1. It is logical to save guide stars with a Pointing. Because individual lists of guide stars are only meaningful for a particular pointing.

3.3 Interface Details

The interface will have the following behavior for action events. Refer to figure 2 for a mockup of the interface.

Action	Behavior
"Runid" Combo Box Action	Populate table with pointing sets for specified runid pattern

Action	Behavior
“Catalog” Combo Box Action	Acquire a new star catalog from StarCatalogFactory
“Pointing” cell edit action	Change dither pattern step in the pointing set that will be viewed by “View Selected” actions
“Search All” Button Action	Search all pointing sets for guide stars
“Search” Button Action	Search highlighted pointing set for guide stars
“Stop Search” Button Action	Halt any search in progress
“View Selected” Button Action	Display highlighted pointing set & selected pointing in a SkyPlot
“Add Star” Button Action	Add selected star on SkyPlot to the guide star collection. Set Rank to lowest in collection
“Remove Star” Button Action	Remove the selected star in the SkyPlot from the guide star collection. Shift all lower ranked guide stars up in rank.
“Change Rank” Button Action	Change the rank of the selected guide star in the SkyPlot. Shift rank of inclusive guide stars down or up accordingly.

Table 4: Behavior of interface

4 Project Timeline

The goal of the Gustaf project is to have a working version by the beginning of February. A working version only requires up through milestone 6, which should be complete at least two weeks prior to release, for testing purposes. Milestones 6, 7 and possibly eight can be developed in parallel with milestones 1 through 6, making a release date of early February possible but tight. This schedule includes the time I will spend observing at the end of december as well as holiday time. However, the schedule does not include time that must be devoted to finalizing the CFHT-CADC ICD and metadata transfer.

#	Milestone	Time Required	Date Complete
1	Build Pointing Sets from database	1 week	December 13, 2002
2	Star Catalog queries from PointingSets	1 week	December 20, 2002
3	Basic Guide Star Selection	2 weeks	January 17, 2003
4	Plotting for validation of guide star selection	1 week	January 24, 2003
5	Remaining GUI details	1 week	January 31, 2003
6	Database insertion & deletion of stars	?	?
7	Guide star selection & modification from plot	?	?
8	Integration of GUSTAF into QSO Tools and OT	?	?