

Ground Based Scheduling Engine Application Interface

Tony Krueger, Larry Kramer

Document Version 7.4

Updated 2/4/1999

1.0 Overview

This document describes the application program interface for both the long term and short term scheduling engines used by the ESO-VLT and SUBARU systems. Assume that all functionality is available for both scheduling engines, unless it is explicitly mentioned for one.

2.0 Definitions

- Assignment - The association of a task with a specific start time
- Conflict - A constraint violation for a task
- Constraint - A restriction of some sort between a set of tasks or on one task
- Ignored - A task that is not a candidate for scheduling
- Locked - A task is scheduled at a particular time and can not be moved from that time
- PCF - A piecewise continuous function representing values over time. For example, (time1 value1 time2 value2 ... timeN valueN). There is <value1> available from <time1> (inclusive) to <time2> (exclusive) ...
- Suitability - A measure of goodness. Suitabilities are between 0-1. Higher values are better than lower values.
- Task - An object which is scheduled

3.0 Generic Specifications for Interface Functions

The following specifications will be referred to throughout the document.

Time Specification <TIME-SPEC>

Times are specified as follows. The time is specified in Universal Time.

(yr mon day hr min sec ms)

yr - 4 digit number representing the year (i.e., 1996)

mon - number representing the month (i.e., 1-12)

day - number representing day of month (ie., 1-31)

hr - number representing the hour (ie., 0-23)

min - number representing the minute (ie., 0-59)

sec - number representing the second (ie., 0-59)

ms - number of milliseconds (ie., 0-1000) - optional argument - defaults to 0

PCF Specification <PCF-SPEC>

Specifies how a function (ie.,values) change over time. Value could be preference, conflicts, resource, etc.

Format -> (t1 v1 t2 v12 ... tN vN)

t1-tN are times. See <Time-Spec> definition

v1-vN are values. Units depend on value

This says that v1 is available between t1 (inclusive) and t2 (exclusive), v2 is available between t2 (inclusive) and t2(exclusive), etc.

Task (ie., observation runs & observation blocks) Names

All task names are strings and will be automatically upcased by the software. Spike internally represents everything as Lisp Symbols which are not case sensitive so all task strings are upcased automatically.

4.0 Scheduling Engine Input

This section describes the major data inputs for the scheduling engine.

System Setup

This section describes the functions that can be called to setup the scheduling system.

(defun INIT-LT-SCHEDULER (&rest arguments &key parameter-files task-files constraint-files scheduler-name &allow-other-keys)

ARGS:

parameter-files - a list of file paths for the location of the global scheduling parameter files

parameter-data - a list of forms containing the global parameter data this is the same data that would be stored in the parameter-files

task-files - a list of file paths for the location of the task files

task-data - a list of forms containing the task data this is the same data that would be stored in the task-files

constraint-files - a list of file paths for the location of the constraint files

constraint-data - a list of forms containing the constraint data this is the same data that would be stored in the constraint-files

scheduler-name - the name of the scheduler, string, default "LONG-TERM-SCHEDULE"

DOCUMENTATION

Loads up a long term schedule's observation set & global scheduling parameters which are defined in subsequent sections in this document. Sets up the necessary data structures needed for scheduling. An observation set is a set of tasks (observation runs | blks) and constraints between the tasks. You can store your data in files only, or pass it to the function only, or store it in both the files and pass it in as data. For example with parameter data, here is how things work

```

If parameter-files Then
  loop through parameter-files and load them
Endif
If parameter-data Then
  loop through parameter-data and eval it
Endif

```

(defun INIT-ST-SCHEDULER (&rest arguments &key parameter-files task-files constraint-files cache-file scheduler-name &allow-other-keys)

ARGS:

parameter-files - a list of file paths for the location of the global scheduling parameter files

parameter-data - a list of forms containing the global parameter data this is the same data that would be stored in the parameter-files

task-files - a list of file paths for the location of the task files

task-data - a list of forms containing the task data this is the same data that would be stored in the task-files

constraint-files - a list of file paths for the location of the constraint files

constraint-data - a list of forms containing the constraint data this is the same data that would be stored in the constraint-files

cache-file - a file path for the location of the czl cache for this scheduler optional. If supply, then cache file is used. If not supplied, then Spike calculates things on load each time. For large data sets it is useful to use the cache. For small data sets, the cache is slower. Default is no cache.

scheduler-name - the name of the scheduler, string, default "SHORT-TERM-SCHEDULE"

DOCUMENTATION

Loads up a short term schedule's observation set & global scheduling parameters which are defined in subsequent sections in this document. Sets up the necessary data structures needed for scheduling. An observation set is a set of tasks (observation runs | blks) and constraints between the tasks. Updates the short term cache. You can store your data in files only, or pass it to the function only, or store it in both the files and pass it in as data. For example with parameter data, here is how things work

```

If parameter-files Then
  loop through parameter-files and load them
Endif
If parameter-data Then
  loop through parameter-data and eval it
Endif

```

(defun PREP-OBS-SET (&rest arguments &key parameter-files task-files cache-file &allow-other-keys)

ARGS:

parameter-files - a list of file paths for the location of the global scheduling parameter files

task-files - a list of file paths for the location of the task files

cache-file - a file path for the location of the czl cache for this scheduler

DOCUMENTATION:

“ARGS:

parameter-files - a list of file paths for the location of the global scheduling parameter files
task-files - a list of file paths for the location of the task files
cache-file - a file path for the location of the czl cache for this scheduler
optional. If supply, then cache file is used. If not supplied, then Spike calculates things on load each time. For large data sets it is useful to use the cache. For small data sets, the cache is slower. Default is no cache.

DOCUMENTATION:

Loads up the short terms schedule’s observation set & global scheduling parameters. Sets up the necessary data structures for updating and saving the short term cache. Updates the short term cache. This function is called from init-st-scheduler and should not be called directly when init-st-scheduler since it will overwrite the global data structures. This function could be called directly under the following scenerio.

- Prep all targets known say 1000 targs daily -> call prep-obs-set
- Next day we build a short term schedule for say 100 of the 1000 targets. There will be no overhead loading and calculating things since the targets were prepped ahead of time.”

)

(defun RESTORE-SCHEDULE (&rest arguments &key file (which-scheduler :tl)&allow-other-keys)

ARGS:

file - pathname to schedule. required arg
which-scheduler - either :lt or :st for long term or short term respectively. default :lt

DOCUMENTATION:

Loads a schedule from file and makes it the current (ie., set of assignments, locked, ignored tasks) schedule. NOTE: If any observations are locked, ignored, or assigned in memory, these are all undone and the information in the file is applied.

Task & Constraint Definition

This section describes the functions which setup the task and task constraints

(defun DEF-OBS-BLOCK (&rest arguments &key name program-id ra dec task-duration priority telescope instrument mode setup-time req-seeing sky-transparency frac-lunar-illum moon-target-exclusion-angle meridian-suitability-map meridian-efficiency-map task-info override-scheduling-intervals &allow-other-keys))

ARGS:

name - a string which names the task. required arg

program-id	- a string which identifies the program id, required arg
ra	- right ascension of target for task (J2000), degrees, required arg
dec	- declination of target for task (J2000), degrees, required arg
task-duration	- duration of the task in seconds, greater than 0, required arg
priority	- priority of task, lower numbers mean higher priority, positive nums, required arg
telescope	- string naming the telescope that is needed, required arg
instrument	- string describing the instrument used, required arg
mode	- string describing the mode used or a list of strings describing the modes used by the observation. If a single string, then the observation is assumed to use only one mode and overheads will be calculated using the mode supplied. If a list is supplied, Spike assumes that the first mode in the list is used at the beginning of the observation and the last mode in the list is used at the end of the observation. All modes supplied in between the first and last in the list are ignored. The first mode is used to calculate overheads before the observation is executed. The last mode is used to calculate overheads after the observation is executed.
sun-twilight-angle	- If the angle separation between the meridian and the sun is less than this value, then the sun is up and the sun suitability is 0. Angle in degrees. Default is nil. If non-nil, overrides the global setting from DEF-ST-SCHEDULER-PARAMETERS. If not supplied, the global default is used.
setup-time	- overhead associated with task in seconds. Default is 0
req-seeing	- number representing the observation block's required seeing. When creating a seeing pcf for the observation the following algorithm is performed. If the observation's seeing \geq telescope seeing at a specific time, then suitability is 1 else 0. Default is nil, which means that the seeing pcf will not affect scheduling. This attribute may also be expressed as a pcf as follows. (\langle current-seeing-value \rangle \langle suit \rangle \langle current-seeing-value \rangle \langle suit \rangle) See \langle PCF-SPEC \rangle above.
sky-transparency	- number representing the observation block's required transparency. When creating a seeing pcf for the observation the following algorithm is performed. If the observation's transparency \geq telescope transparency at a time, then suitability is 1 else 0. Default is nil, which means that the transparency pcf will not affect scheduling. This attribute may also be expressed as a pcf as follows. (\langle sky-trans-value \rangle \langle suit \rangle \langle sky-trans-value \rangle \langle suit \rangle). See \langle PCF-SPEC \rangle above.
frac-lunar-illum	- maximum allowed Fractional Lunar Illumination at local civil midnight. Values between 0 (new moon) and 1 (full moon). required arg
moon-target-exclusion-angle	- minimum allowed angular distance between target and the moon (in degrees, 0 to 180). required argument.

- meridian-suitability-map- table that maps angular separation between the meridian and the line of sight of the target to suitability values. A PCF. Values are suitabilities between 0.0 and 1.0. What usually is time is the angular separation in degrees. If this value is supplied, then it is applied to this obs block only and overrides the defaults specified in the DEF-ST-SCHEDULER-PARAMETERS form. If not supplied, the global default is used.
- meridian-efficiency-map- table that maps angular separation between the meridian and the line of sight of the target to efficiency values. A PCF. Values are efficiencies between 0.0 and 1.0. What usually is time is the angular separation in degrees. If this value is supplied, then it is applied to this obs block only and overrides the defaults specified in the DEF-ST-SCHEDULER-PARAMETERS form. If not supplied, the global default is used.
- task-info - property list of specific info pertaining to the observation block. this could be things like ‘(:council-rank 1 :merit-rank 9)
- override-scheduling-intervals
 - Either t or nil. Default is nil. If T then any absolute constraint can be violated. This flag allows the user to manually assign the observation to any time even if it violates an absolute constraint. If nil then no absolute constraints can be violated.

DOCUMENTATION

Defines one observation block. SHORT TERM SCHEDULER ONLY.

(defun DEF-CALIBRATION-OBS-BLOCK (&rest arguments &key name program-id task-duration priority telescope instrument mode setup-time viewing-time task-info override-scheduling-intervals &allow-other-keys))

ARGS:

- name - a string which names the task. required arg
- program-id - a string which identifies the program id, required arg
- task-duration - duration of the task in seconds, greater than 0, required arg
- priority - priority of task, lower numbers mean higher priority, positive nums, required arg
- telescope - string naming the telescope that is needed, required arg
- instrument - string describing the instrument used, required arg
- mode - string describing the mode used or a list of strings describing the modes used by the observation. If a single string, then the observation is assumed to use only one mode and overheads will be calculated using the mode supplied. If a list is supplied, Spike assumes that the first mode in the list is used at the beginning of the observation and the last mode in the list is used at the end of the observation. All modes supplied in between the first and last in the list are ignored. The first mode is used to calculate overheads before the observation is executed. The last mode is used to calculate overheads after the observation is executed.
- sun-twilight-angle - If the angle separation between the meridian and the sun is

- less than this value, then the sun is up and the sun suitability is 0.
 Angle in degrees. Default is nil. If non-nil,
 overrides the global setting from DEF-ST-SCHEDULER-PARAMETERS
 If not supplied, the global default is used.
- setup-time - overhead associated with task in seconds. Default is 0
- viewing-time - one of “night”, “day”, “all-time”. Default is “night”. Used to restrict
 when the calibration should be scheduled. If “night”, then night.
 If “day”, then schedule in day. If “all-time”, schedule day or night.
 The constraint is basically a sun constraint. The same sun algorithm
 is used for a normal observation block
- task-info - property list of specific info pertaining to the observation block.
 this could be things like ‘(:council-rank 1 :merit-rank 9 ...).
 This is also where you specify a known altitude/azimuth for the
 calibration block if known ‘(:altitude 40 :azimuth 180). Altitude is
 expected to be between 0 and 90 degrees. Azimuth is assumed to
 be between 0 and 360 degrees.
- override-scheduling-intervals
 - Either t or nil. Default is nil. If T then any absolute constraint can be
 violated. This flag allows the user to manually assign the observation
 to any time even if it violates an absolute constraint. If nil then no
 absolute constraints can be violated.

DOCUMENTATION

Defines one calibration observation block. SHORT TERM SCHEDULER ONLY.

(defun SET-OBS-BLOCK-PRIORITY (&rest arguments &key name priority))

ARGS:

- name - a string which names the task, required arg
- priority - priority of task, lower numbers mean higher priority, positive nums,
 required arg

DOCUMENTATION:

Sets the priority of an observation block of an existing observation block. SHORT TERM ONLY.

(defun DEF-OBS-RUN (&rest arguments &key name (program-id “”) observing-mode priority
 telescope instrument (night-demand “whole”) task-duration moon-pref scheduled-date task-info
 override-scheduling-intervals &allow-other-keys))

ARGS:

- name - a string which names the task, required arg
- program-id - a string which identifies the program id, required arg. Not really
 applicable for “service” mode runs. Default is “”
- observing-mode - one of “service”, “classical” required arg
- priority - priority of task, lower numbers mean higher priority, positive nums,
 required arg
- telescope - string naming the telescope that is needed, required arg
- instrument - string describing the instrument used, required arg

- night-demand - how to schedule this task in the night. One of “whole”, “first-half”, “second-half”, “any-half”, “first-third”, “second-third”, “third-third”, or “any-third”. Default is “whole”
- task-duration - number of nights needed. When night-demand = “whole” this is clear what happens. Basically, you want X consecutive whole nights. For fractional nights, lets say that you specify night-demand = “first-half” and task-duration = 2, then you are asking for two consecutive nights where this task is scheduled in the first-half of the night.
- target-ra-list - list of target ra for the obs run in degrees. Can be one or more RA in the list. Also can be nil. Default is nil. This is used to calculate a suitability function for the obs run based upon how far the target(s) are from the meridian. MERIDIAN-TO-SUIT-MAP defined in DEF-LT-PLANNING-PARAMETERS, is used to map RA difference from meridian to a suitability value. If more than one target RA is specified, the max function is used to combine the component RA suitabilities. For example, if two RA’s were specified. One at 100 and the other at 110. The 100 RA had a 0 suitability and the 110 had a .25 suitability at the sampled time, the resultant suitability would be .25 at the sampled time.
- moon-pref - one of “D”, “G”, or “N” for dark, gray, or no preference for task no preference is the default
- scheduled-date - a date when the obs-run is bscheduled. Used to assign a obs-run to a time period. See <Time-Spec> definition. If none supplied not scheduled.
- task-info - property list of specific info pertaining to the observation block. this could be things like ‘(:council-rank 1 :merit-rank 9)
- override-scheduling-intervals - Either t or nil. Default is nil. If T then any absolute constraint can be violated. This flag allows the user to manually assign the observation to any time even if it violates an absolute constraint. If nil then no absolute constraints can be violated.

DOCUMENTATION

Defines one observation run. LONG TERM SCHEDULER ONLY.

```
(defun DEF-ORDER (&rest arguments &key task1 task2 &allow-other-keys))
```

ARGS:

- task1 - a string which names task1
- task2 - a string which names task2

DOCUMENTATION

task1 is before task2

```
(defun DEF-ORDER-AND-OFFSET (&rest arguments &key task1 task2 offset &allow-other-keys))
```

ARGS:

task1 - a string which names task1
task2 - a string which names task2
offset - minimum end to start separation between task1 & task2 in days

DOCUMENTATION

task1 is before task2 by offset

(defun DEF-GROUP-WITHIN (&rest arguments &key time tasks &allow-other-keys))

ARGS:

time - all tasks within this time in days
tasks - list of tasks, where each task is a string ->("tn1" ... "tnN")

DOCUMENTATION

All tasks should be grouped within the specified time

(defun DEF-MIN-TIME-SEPARATION (&rest arguments &key task1 task2 min-time &allow-other-keys))

ARGS:

task1 - a string which names task1
task2 - a string which names task2
min-time - minimum time separation between task1 & task2 in days

DOCUMENTATION

task1 and task2 should be separated by a minimum of min-time

(defun DEF-WINDOW (&rest arguments &key early-start late-start task preference &allow-other-keys))

ARGS:

early-start - earliest time the task can start. See <TIME-SPEC> above.
late-start - latest time the task can start. See <TIME-SPEC> above.
task - a string which names task
preference - a preference for this window. Number between 0 and 1.0

DOCUMENTATION:

task should start between early-start and late-start with preference. So we could say schedule task between early-start and late-start with preference. The total window definition is the intersection of all the DEF-WINDOW forms.

(defun DEF-PHASE (&rest arguments &key start-time period phase-min phase-max task &allow-other-keys))

ARGS:

start-time - time when the periodic phenomena starts. See <TIME-SPEC> above.
period - period of the phenomena in days
phase-min - start point in the period that you are interested in number between 0 and 1. Percent of period
phase-max - end point in the period that you are interested in number between 0 and 1. Percent of period
task - a string which names the task

DOCUMENTATION

Defines a periodic phenomena for task and what part of the period (ie., phase) the task is inter-

ested in observing.

```
(defun DEF-LINK-RESTRICTION (&rest arguments &key restriction tasks &allow-other-keys))
```

ARGS:

- restriction - type of link restriction. One of :schedule-partial or :schedule-whole
If :schedule-whole, then all the tasks are scheduled or non of them should be scheduled
If :schedule-partial, then here is how the things will work
- if tasks = a, b, c and a and c are scheduled but b is not, then c should also not be scheduled. Basically make sure that no tasks are scheduled after the first unscheduled task
- tasks - list of tasks, where each task is a string ->("tn1" ... "tnN")

DOCUMENTATION

Defines scheduling restrictions on a group of tasks.

Global Scheduler Parameters

```
(defun DEF-LT-SCHEDULER-PARAMETERS (&rest arguments &key sched-start sched-end  
time-quantum moon-dark-map moon-gray-map moon-no-pref-map dark-before-new-moon  
dark-after-new-moon gray-before-new-moon gray-after-new-moon suitability-scale default-  
priority &allow-other-keys))
```

ARGS:

- sched-start - start time for the scheduler. See <TIME-SPEC> above. required arg
- sched-end - end time for the scheduler. See <TIME-SPEC> above. required arg
- time-quantum - the time discretization interval for the scheduler in days.
- moon-dark-map - used by the dark moon preference constraint to create a dark suitability pcf. format is list like below
(:dark <dark-suit> :gray <gray-suit> :nopref <nopref-suit>)
If the moon is in dark use <dark-suit>,
If the moon is in gray use <gray-suit>,
nopref use <nopref-suit>
Default -> (:dark 1.0 :gray 0.25 :nopref .01)
- moon-gray-map - used by the gray moon preference constraint to create a gray suitability pcf. format is list like below
(:dark <dark-suit> :gray <gray-suit> :nopref <nopref-suit>)
If the moon is in dark use <dark-suit>,
If the moon is in gray use <gray-suit>,
nopref use <nopref>
Default -> (:dark .5 :gray 1 :nopref .01)
- moon-nopref-map - used by the nopref moon preference constraint to create a nopref suitability pcf. format is list like below
(:dark <dark-suit> :gray <gray-suit> :nopref <nopref-suit>)
If the moon is in dark use <dark-suit>,
If the moon is in gray use <gray-suit>,
nopref use <nopref>
Default -> (:dark 0.01 :gray 0.01:nopref 1)

meridian-to-suit-map - maps RA distance from meridian to a suitability value.

format is as follows

(<angle-degrees> <suit> <angle-degrees> <suit>)

<angle-degrees> is RA angular separation from meridian

<suit> is a suitability between 0 and 1

Default is (0 1 90 0)

dark-before-new-moon - the number of days considered dark before the new moon.

Default is 7.

dark-after-new-moon - the number of days considered dark after the new moon. Default is 5

gray-before-new-moon - the number of days considered gray before the new moon.

Default is 11.

gray-after-new-moon - the number of days considered gray after the new moon. Default is 9

suitability-scale - Multiply suitability values (0 to 1) by this value and round

to get integral preference values. default 100

default-priority - default priority value for tasks. default 100

DOCUMENTATION:

Defines a set of global scheduler parameters for the long term scheduler

(defun DEF-ST-SCHEDULER-PARAMETERS (&rest arguments &key sched-start
sched-end time-quantum sun-twilight-angle moon-twilight-angle moon-phase-sample-inter-
val meridian-steps meridian-suitability-map meridian-efficiency-map min-efficiency suitabil-
ity-scale default-priority abs-adjust rel-adjust min-segment-size &allow-other-keys))

ARGS:

sched-start - start time for the scheduler. See <TIME-SPEC> above. required arg

sched-end - end time for the scheduler. See <TIME-SPEC> above. required arg

time-quantum - the time discretization interval for the scheduler in days.

sun-twilight-angle - if the angle separation between the meridian and the sun is less than
this value, then the sun is up and the sun suitability is 0. Angle is degrees
Default is 108.

moon-twilight-angle - if the angle separation between the meridian and the moon is less than
this value, then the moon is up and the moon suitability is 0. Angle is
degrees. Default is 108.

moon-phase-sample-rate - the default sample interval for sampling the moon phase pcf.
greater than 0. 0.5 = 12hrs. Default is 0.2 or about every 1 degree of
moon movement

meridian-steps - then number of sampling steps between sched-start & sched-end
used in calculating the angles between the meridian and sun, moon,
and target. Default is 100

meridian-suitability-map- table that maps angular separation between the meridian and the
line of sight of the target to suitability values. A PCF. Values are
suitabilities between 0.0 and 1.0. What usually is time is the angular
separation in degrees.

Default is '(0 1.0 31 0.8 46 0.5 61 0.25 71 0.1 76 0)

meridian-efficiency-map- table that maps angular separation between the meridian and the
line of sight of the target to efficiency values. A PCF. Values are
efficiencies between 0.0 and 1.0. What usually is time is the angular

- separation in degrees.
Default is '(0 1.0 31 0.75 61 0.5 71 0.3 76 0)
- min-efficiency - the minimum efficiency value. Any efficiency value less than this number will make the task suitability at a particular time 0. Default is 0.5
 - suitability-scale - Multiply suitability values (0 to 1) by this value and round to get integral preference values. default 100
 - default-priority - default priority value for tasks. default 100
 - abs-adjust - time in seconds by which exposure time can be adjusted without impact default 2 mins
 - rel-adjust - fraction of exposure time which can be added/subtracted without impact default 0.2 = +/- 20% of exp time can be changed
 - min-segment-size - time in seconds of minimum exposure time segment on target, if exposure must be interrupted. default 300 = 5min
 - average-suitability - either t or nil. If t, each observation's start time suitability is the average of all the tasks suitability windows from start to end. If nil, each observation's start time suitability is the suitability at the start time only.

DOCUMENTATION:

Defines a set of global scheduler parameters for the short term scheduler

```
(defun DEF-TELESCOPE-PARAMETERS (&rest arguments &key name telescope-unavail
  instrument-unavail instr-chg-tbl current-seeing sky-transparency blongitude latitude altitude
  &allow-other-keys))
```

ARGS:

- name - telescope name -> string. required arg
- telescope-unavail - specifies when the telescope is not available for scheduling
format is a list of time intervals when the telescope is not available
((reason ((t1 t2) (t3 t4)))....)
Each sublist contains 2 elements, an instrument and a list of unavailable times. The reason is a string.
First time in each pair is inclusive, 2nd is exclusive
t1-tN are times. See <TIME-SPEC> above
Default is the telescope is always available
- instrument-unavail - specifies when instruments are unavailable for the telescope
format is a list of time intervals when the instruments are not available
((instr1 ((t1 t2) (t3 t4)))....)
Each sublist contains 2 elements, an instrument and a list of unavailable times. The instrument is a string. Times see <TIME-SPEC> above
First time in each pair is inclusive, 2nd is exclusive
All instruments are available always as a default.
- no-instrument-change-over - specifies when the telescope is not available for instrument change overs. This is a hard constraint requiring that if the telescope is using a particular instrument it must continue to do so throughout an interval.
format is a list of time intervals when instrument change overs are

- restricted. ((reason ((t1 t2) (t3 t4)))....)
- Each sublist contains 2 elements, a reason why change overs are restricted and a list of unavailable times. The reason is a string. Times see <TIME-SPEC> above
- First time in each pair is inclusive, 2nd is exclusive
- Default is the telescope is always available for instrument change overs.
- Applies only to the short term schedule.
- instr-chg-tbl - list of the costs associated with switching between instruments/modes
- Format -> (((inst1 mode1) (inst2 mode2) time1)
 ((inst2 mode2) (inst1 mode1) time2)
 (:default timeN)
- inst1-instN - string naming the instrument
- mode1-modeN - string naming the mode. this can be null string and then just instrument to instrument is modeled
- :default - default instr chg overhead
- time1-timeN - seconds
- required arg
- EX/ (((("inst1" "mode1") ("instr2" "mode2") 10) ;; STS format
 (("inst3" "") ("inst4" "")) 20) ;; LTS format
 (:default 5))
- The first line is an example of instr/mode to instr/mode overhead
- The second line is an example of instrument to instrument overhead
- Note, that the mode is a null string. The observation should also have its mode set to the null string
- The third line is the default time used if an entry can't be found in the table. The default does not have to be specified.
- current-seeing - a pcf which represents the telescope seeing over time. Times are specified in <TIME-SPEC> format. Values are between 0-1. This pcf is used to create observation block seeing pcf's based on the following algorithm.
- For each telescope seeing value in the telescope pcf
- If the obs block's seeing is >= telescope seeing value
- the obs block suitability for time interval is 1.0
- Else
- the obs block suitability for time interval is 0.0
- Default is nil, which means that no observation block will take seeing into account. Basically ignore current seeing constraint.
- Applies to SHORT TERM ONLY.
- sky-transparency - a pcf which represents the telescope transparency over time. Times are specified in <TIME-SPEC> format. Values are between 0-1. This pcf is used to create observation block transparency pcf's based on the following algorithm.
- For each telescope transparency value in the telescope pcf
- If the obs block's transparency is >= telescope transparency value
- the obs block suitability for time interval is 1.0
- Else

the obs block suitability for time interval is 0.0
 Default is nil, which means that no observation block will take transparency into account. Basically ignore transparency constraint. Applies to SHORT TERM ONLY.

- longitude - geographic longitude in degrees, 0 to 360, measured EASTWARD. required arg.
- latitude - geographic latitude in degrees, -90 to 90. required arg
- altitude - altitude in meters above sea level, must be between -100 and 10000 meters. required arg
- max-speed - maximum speed of the telescope in degrees/sec
- acceleration - acceleration of the telescope in degrees/sec²

DOCUMENTATION:

Defines a set of global telescope parameters

5.0 Scheduling Engine Output

This section defines a set of functions that generate output for the scheduling engine

(defun SAVE-SCHEDULE (&rest arguments &key file (overwrite t) (which-scheduler :lt) schedule-name &allow-other-keys)

ARGS:

- file - file location for schedule. required arg
- overwrite - if schedule exists overwrite old schedule if t else don't. Default overwrite
- which-scheduler - either :lt or :st for long term or short term respectively default is :lt
- schedule-name - name of the schedule, a string. required arg. When the schedule is restored, this is the name that it will be called.

DOCUMENTATION:

Saves current schedule to file (ie., set of assignments, locked, ignored tasks).

(defun SCHEDULE-REPORT-TEXT (&rest arguments &key file (overwrite t) (which-scheduler :lt) &allow-other-keys)

ARGS:

- file - file location for report. required arg
- overwrite - if file exists overwrite old file if t else don't. Default overwrite
- which-scheduler - either :lt or :st for long term or short term respectively default is :lt

DOCUMENTATION:

Save a textual schedule report to a file. These functions also return a formatted list of keyword/value pairs describing the schedule.

(defun SCHEDULE-REPORT-PS (&rest arguments &key file (overwrite t) (which-scheduler :lt) &allow-other-keys)

ARGS:

- file - file location for report. required arg

overwrite - if file exists overwrite old file if t else don't. Default overwrite
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Save a graphical schedule report to a file. Format is postscript.

(defun TASK-REPORT-TEXT (&rest arguments &key tasks file (overwrite t) (which-scheduler :lt) &allow-other-keys)

ARGS:

task - a list of tasks where each task is a string. required arg
file - file location for report. required arg
overwrite - if file exists overwrite old file if t else don't. Default overwrite
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Save a textual report for a set of tasks to a file.

(defun TASK-REPORT-PS (&rest arguments &key task file (overwrite t) (which-scheduler :lt) start-time end-time &allow-other-keys)

ARGS:

task - name of task - string, required arg
file - file location for report, required arg
overwrite - if file exists overwrite old file if t else don't. Default overwrite
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt
start-time - start point for printing. Must be \geq to schedule start. See <TIME-SPEC> above. Default is schedule start
end-time - end point for printing. Must be \leq to schedule end. See <TIME-SPEC> above. Default is schedule end

DOCUMENTATION:

Save a postscript file showing the different task scheduling components.

(defun GAP-REPORT-TEXT (&rest arguments &key file (overwrite t) (which-scheduler :lt) &allow-other-keys)

ARGS:

file - file location for report, required arg
overwrite - if file exists overwrite old file if t else don't. Default overwrite
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Save a textual report showing the schedule gaps and which tasks can fill the gap.

(defun OBSERVATORY-SUN-MOON-ZENITH-PS (&rest arguments &key observatory file (overwrite t) &allow-other-keys)

ARGS:

observatory - name of the observatory - string, required arg

file - file location for report, required arg
overwrite - if file exists overwrite old file if t else don't. Default overwrite
DOCUMENTATION:
Saves a postscript file plotting the sun and moon zenith angles for an observatory.
Short Term Only”

(defun RA-DISTRIBUTION (&rest arguments &key tasks file (overwrite t)
(cluster-interval 10) &allow-other-keys)

ARGS:

tasks - list of tasks to generate a distribution for, list of strings, required arg
file - file location for report, required arg
overwrite - if file exists overwrite old file if t else don't. Default overwrite
cluster-interval - cluster targets in cluster-interval degrees. Default is 10 degrees.

RETURN-VALUES:

Return a formatted list of keyword/value pairs describing the ra distribution.

DOCUMENTATION:

Save a textual RA distribution report to a file. Reports things like the number of targets in each cluster interval. Average RA, etc. These functions also return a formatted list of keyword/value pairs describing the ra distribution. SHORT TERM ONLY

6.0 Scheduling Engine Functionality

This section defines a set of functions that the scheduling engine supports

Management of Schedules in Memory

The scheduling engine allows for different schedules to be generated and manipulated in memory at once. This section describes this functionality. A schedule in memory is typically a set of assignments.

(defun SELECT-SCHEDULE (&rest arguments &key name &allow-other-keys)

ARGS:

name - name of the schedule whose assignments should be made current - string

DOCUMENTATION:

Makes the assignments for this schedule the current set of assignments in memory

(defun DELETE-SCHEDULE (&rest arguments &key name &allow-other-keys)

ARGS:

name - name of the schedule that should be removed from memory - string

DOCUMENTATION:

This schedule and its assignments are removed from memory

(defun ALL-SCHEDULES (&rest arguments &key which-scheduler)

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Returns a list of schedules known in memory. List of strings

Creating Schedules

Creates new schedules in memory with different auto schedule algorithms

(defun LARGEST-TASK-FIRST (&rest arguments &key name (tasks :all) (which-scheduler :lt) &allow-other-keys)

ARGS:

- name - name of the new schedule -> string, required arg
- tasks - either a list of tasks to schedule with this algorithm -> list of strings
or :all for all unignored and unlocked tasks. Default is :all.
- which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

LONG TERM SCHEDULE ONLY.

The following algorithm is used.

Step 1 - Initial Guess

- For all unassigned tasks, pick a task with the maximum duration
- if more than one task, pick the task who can go earliest with minimum conflicts
- if more than one task, pick the task with the minimum constraint violations
- if more than one task, pick the first in the list
- pick a time when there is a minimum number of conflicts
- if more than one time, pick the highest preference time
- if more than one time, pick the first time

Step 2 - Repair

- For all tasks with conflicts, pick a task with the most conflicts
- if more than one task, pick the first task in the list
- pick a time when there is a minimum number of conflicts
- if more than one time, pick the highest preference time
- if more than one time, pick the first time

Step 3 - Unassign conflicted tasks

- For all tasks with conflicts, unassign the tasks in the following order
- pick a task with the minimum duration
- if more than one task, pick the task that has the most conflicts on its assignment
- if more than one task, pick the task that has the minimum preference on its assignment
- if more than one time, pick the first task in the list of remaining tasks

Step 4 - Fill in any gaps

- For all unassigned tasks with a time that has no conflicts, pick a task with the maximum max preference with minimum conflicts
- if more than one task, pick a task at random
- pick a time when when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.

Step 5 - Respect Link Restrictions

- For all tasks which are assigned and cause a def-link-restriction violation, unassign them
- Gather all unassigned tasks with no conflicts
- Remove any tasks that were just unassigned for the list of candidate tasks to schedule.
- For all remaining tasks, pick a task with the maximum max preference with minimum conflicts
- if more than one task, pick a task at random
- pick a time when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.

(defun EARLY-GREEDY (&rest arguments &key name (tasks :all) (which-scheduler :lt) &allow-other-keys)

ARGS:

- name - name of the new schedule -> string, required arg
- tasks - either a list of tasks to schedule with this algorithm -> list of strings or :all for all unignored and unlocked tasks. Default is :all.
- which-scheduler - either :lt or :st for long term or short term respectively default is :lt

DOCUMENTATION:

The following algorithm is used.

Step 1 - Early assignment

- For all unassigned tasks that have one time with no conflicts, pick a task who can go the earliest without any conflicts
- if more than one task, pick the task with the earliest time with no conflicts
- if more than one task, pick the highest priority task
- if more than one task, pick the task with the maximum preference on the earliest time with not conflicts
- if more than one task, pick the first task in the remaining list
- pick a time with the minimum conflicts
- if more than one time, pick the first time

Step 2 - Respect Link Restrictions

- For all tasks which are assigned and cause a def-link-restriction violation, unassign them
- Gather all unassigned tasks with no conflicts
- Remove any tasks that were just unassigned for the list of candidate tasks to schedule.
- For all remaining tasks, pick a task with the maximum max preference with minimum conflicts
- if more than one task, pick a task at random
- pick a time when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.

(defun EARLY-LEAST-MINIMAL (&rest arguments &key name (tasks :all) (which-scheduler :lt) &allow-other-keys)

ARGS:

- name - name of the new schedule -> string, required arg
- tasks - either a list of tasks to schedule with this algorithm -> list of strings

or :all for all unignored and unlocked tasks, default is :all
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

The following algorithm is used

Step 1 - Initial guess

- For all unassigned tasks, pick the task that has the minimum number of conflicts on its first schedulable time.
- if more than one task, pick a task that has the minimum number of minimum conflicted times
- if more than one task, pick the first task in the remaining list
- pick a value that has the minimum number of conflicts
- if more than one value, pick the first time

Step 2 - Repair

- For all tasks with conflicts, pick a task with the most conflicts on its assignment
- if more than one task, pick the first task in the list
- pick a time when there is a minimum number of conflicts
- if more than one time, pick the highest preference time
- if more than one time, pick the first time

Step 3 - Unassign conflicted task

- For all tasks with conflicts, unassign the tasks in the following order
- pick the task with the most conflicts on its assignment
- if more than one task, pick a task with minimum preference on its assignment
- if more than one task, pick the first task in the remaining list of tasks

Step 4 - Fill in any gaps

- For all unassigned tasks with a time that has no conflicts, pick a task with the maximum max preference with minimum conflicts
- if more than one task, pick a task at random
- pick a time when when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.

Step 5 - Respect Link Restrictions

- For all tasks which are assigned and cause a def-link-restriction violation, unassign them
- Gather all unassigned tasks with no conflicts
- Remove any tasks that were just unassigned for the list of candidate tasks to schedule.
- For all remaining tasks, pick a task with the maximum max preference with minimum conflicts
- if more than one task, pick a task at random
- pick a time when when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.

(defun MAX-PREF (&rest arguments &key name (tasks :all) (which-scheduler :lt) &allow-other-keys)

ARGS:

- name - name of the new schedule -> string, required arg
- tasks - either a list of tasks to schedule with this algorithm -> list of strings

or :all for all unignored and unlocked tasks. Default is :all.
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

The following algorithm is used

Step 1 - Initial Guess

- For all unassigned tasks, pick the task that has the maximum preference on its minimum conflicted time.

- if more than one task, pick the first task from the remaining list of tasks

- pick a time that has a minimum number of conflicts

- if more than one time, pick the time with the maximum preference

- if more than one time, pick the first time

Step 2 - Repair

- For all tasks with conflicts, pick a task with the most conflicts on its assignment

- if more than one task, pick the first task in the list

- pick a time when there is a minimum number of conflicts

- if more than one time, pick the highest preference time

- if more than one time, pick the first time

Step 3 - Unassign conflicted task

- For all tasks with conflicts, unassign the tasks in the following order

- pick the task with the most conflicts on its assignment

- if more than one task, pick a task with minimum preference on its assignment

- if more than one task, pick the first task in the remaining list of tasks

Step 4 - Fill in any gaps

- For all unassigned tasks with a time that has no conflicts, pick a task with the maximum max preference with minimum conflicts

- if more than one task, pick a task at random

- pick a time when when there is a minimum number of conflicts

- if more than one time, pick a time that has maximum preference

- if more than one time, pick the first time.

Step 5 - Respect Link Restrictions

- For all tasks which are assigned and cause a def-link-restriction violation, unassign them

- Gather all unassigned tasks with no conflicts

- Remove any tasks that were just unassigned for the list of candidate tasks to schedule.

- For all remaining tasks, pick a task with the maximum max preference with minimum conflicts

- if more than one task, pick a task at random

- pick a time when when there is a minimum number of conflicts

- if more than one time, pick a time that has maximum preference

- if more than one time, pick the first time.

```
(defun HI-PRIOR-MAX-PREF (&rest arguments &key name (tasks :all) (which-scheduler :lt)
&allow-other-keys)
```

ARGS:

name - name of the new schedule -> string, required arg

tasks - either a list of tasks to schedule with this algorithm -> list of strings

or :all for all unignored and unlocked tasks. Default is :all
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

The following algorithm is used.

Step 1 - Initial Guess

- For all unassigned tasks, pick the task with the highest priority
- if more than one task, pick the task with the maximum preference on its minimum conflicted time
- if more than one task, pick the first task in the remaining list of tasks
- pick a time that has the maximum preference
- if more than one time, pick a time that has the minimum number of conflicts
- if more than one time, pick the earliest time.

Step 2 - Repair

- For all tasks with conflicts, pick a task with the most conflicts on its assignment
- if more than one task, pick the first task in the list
- pick a time when there is a minimum number of conflicts
- if more than one time, pick the highest preference time
- if more than one time, pick the first time

Step 3 - Unassign conflicted task

- For all tasks with conflicts, unassign the tasks in the following order
- pick the task with the lowest priority
- if more than one task, pick the task with the maximum conflicts on its assignment
- if more than one task, pick the task with the minimum preference on its assignment
- if more than one task, pick the first task in the remaining list of tasks

Step 4 - Fill in any gaps

- For all unassigned tasks with a time that has no conflicts
- pick the highest priority task
- if more than one task, pick the task with the maximum preference on its minimum conflicted time
- if more than one task, pick at random from the remaining tasks
- pick a time that has the minimum number of conflicts
- if more than one time, pick a time that has the maximum preference
- if more than one time, pick the earliest time.

Step 5 - Respect Link Restrictions

- For all tasks which are assigned and cause a def-link-restriction violation, unassign them
- Gather all unassigned tasks with no conflicts
- Remove any tasks that were just unassigned for the list of candidate tasks to schedule.
- For all remaining tasks, pick a task with the maximum max preference with minimum conflicts
- if more than one task, pick a task at random
- pick a time when when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.inst

defun MIN-INSTRUMENT-CHANGE (&rest arguments &key name (tasks :all) (which-sched-

uler :lt) &allow-other-keys)

ARGS:

- name - name of the new schedule -> string, required arg
- tasks - either a list of tasks to schedule with this algorithm -> list of strings or :all for all unignored and unlocked tasks. Default is :all
- which-scheduler - either :lt or :st for long term or short term respectively default is :lt

DOCUMENTATION:

The following algorithm is used.

Step 1 - Initial Guess

- For all unassigned tasks
- group tasks by instrument usage
- order tasks by largest instrument group to smallest instrument group
- loop over the ordered tasks
- pick a task that has the minimum number of conflicted start times
- if more than one task, pick the first task in the remaining list of tasks
- pick a time with the minimum number of conflicts
- if more than one time, pick a time with the minimum instrument overhead
- if more than one time, pick a time gives the minimum distance between the window for this task and the window for an assigned task which has a zero instrument change overhead between the assigned task and this task. Used a criteria to group tasks by instrument.
- if more than one time, pick the earliest time

Step 2 - Repair

- For all tasks with conflicts, pick a task with the most conflicts on its assignment
- if more than one task, pick the first task in the list
- pick a time when there is a minimum number of conflicts
- if more than on time, pick the highest preference time
- if more than one time, pick the first time

Step 3 - Unassign conflicted task

- For all tasks with conflicts, unassign the tasks in the following order
- pick the task with the most conflicts on its assignment
- if more than one task, pick a task with minimum preference on its assignment
- if more than one task, pick the first task in the remaining list of tasks

Step 4 - Fill in any gaps

- For all unassigned tasks with a time that has no conflicts, pick a task with the maximum max preference with minimum conflicts
- if more than one task, pick a task at random
- pick a time when when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.

Step 5 - Respect Link Restrictions

- For all tasks which are assigned and cause a def-link-restriction violation, unassign them
- Gather all unassigned tasks with no conflicts
- Remove any tasks that were just unassigned for the list of candidate tasks to schedule.
- For all remaining tasks, pick a task with the maximum max preference with minimum conflicts

- if more than one task, pick a task at random
- pick a time when there is a minimum number of conflicts
- if more than one time, pick a time that has maximum preference
- if more than one time, pick the first time.

Schedule Operations

This section describes a set of functionalities supported by the scheduler

(defun SCHEDULE-STATUS (&key (which-scheduler :lt))

 ARGS:

 which-scheduler - either :lt or :st for long term or short term respectively
 default is :lt

 RETURN-VALUES:

Returns a list of keyword value pairs containing information about the schedule.

Format (:key1 val1 :key2 val2 ...)

Currently defined key/value pairs

key	value
:name	name of schedule -> string
:sched-start	start time for the scheduler. See <TIME-SPEC> above.
:sched-end	end time for the scheduler. See <TIME-SPEC> above.
:time-quantum	the time discretization interval for the scheduler in days
:all-tasks	list of all tasks -> list of strings
:locked-tasks	list of locked tasks -> list of strings
:ignored-tasks	list of ignored tasks -> list of strings
:assigned-tasks	list of assigned tasks -> list of lists of the form ((task-name <TIME-SPEC>) ...) task-name is a string
:conflicted-tasks	list of tasks with conflicts -> list of strings
:tasks-violating-linking	list of tasks that are assigned but need to be unassigned to respect the link restriction which they are part of. -> list of strings
:resource-usage	list of lists. each sublist describes a resource usage specification Format ((name spec)...) name - string naming the resource Spec Format -> ((t1 t2 ru1 ra1) ... (tN-1 tN ruN raN)) Resource used <ru1> out of resource available <ra1> between t1 and t2 t1 inclusive, t2 exclusive t1-tN are times. See <TIME SPEC> above. ru1-ruN - resource used in units specific to the resource ra1-raN - resource available in units specific to the resource
:top-prior-fraction	distribution of tasks by priority Format ((prior1 percent1)...(priorN percentN)) <percent1> of <prior1> tasks have been scheduled prior1-priorN - priority numbers percent1-percentN - percent of priority scheduled

:total-gaps total time of schedule gaps in days
 :total-sched-dur For all assigned tasks, the summed actual duration for the tasks. This value is the scheduled duration
 :total-min-dur For all assigned tasks, the summed minimum duration for the tasks. This value is the best achievable duration
 :mean-preference the average preference for assigned tasks
 :summed-preference the summed preference for assigned tasks
 :mean-conflicts the average number of conflicts for assigned tasks
 :summed-conflicts the summed conflicts for assigned tasks

DOCUMENTATION:

Returns data describing the current state of the schedule

(defun TELESCOPE-STATUS (name)

 ARGS:

 name - name of telesc-ope -> string, required arg

 RETURN-VALUES:

 Returns a list of keyword value pairs containing information about a telescope.

 Format (:key1 val1 :key2 val2 ...)

 Currently defined key/value pairs

 key value

 :name name of telescope -> string

 :longitude geographic latitude in degrees -90 to 90

 :latitude geographic longitude in radians, 0 to 360, measured EASTWARD

 :altitude altitude in meters above sea level, must be between -100 and 10000 meters

 :max-speed maximum speed of the telescope for slewing in deg/sec

 :acceleration acceleration of the telescope for slewing in deg/sec²

 :telescope-unavail specifies when the telescope is not available for scheduling
 format is a list of time intervals when the telescope is not available
 ((t1 t2) (t3 t4).....)

 First time in each pair is inclusive, 2nd is exclusive

 t1-tN are times. See <TIME-SPEC> above

 Default is the telescope is always available

 :instrument-unavail specifies when instruments are unavailable for the telescope
 format is a list of time intervals when the instruments are not available
 ((instr1 ((t1 t2) (t3 t4))).....)

 Each sublist contains 2 elements, an instrument and a list of unavailable times. The instrument is a string. Times see <TIME-SPEC> above

 First time in each pair is inclusive, 2nd is exclusive

 All instruments are available always as a default.

 :instr-chg-tbl list of the costs associated with switching instruments

 Format -> ((inst1 inst2 time1) (inst2 inst1 time2).....)

 Switching from inst1 to inst2 takes time1

 inst1-instN - string naming the instrument

 time1-timeN - seconds

 :current-seeing specifies the seeing values over time. See <PCF-SPEC> above

PCF values are current seeing values
Default is no restrictions
:sky-transparency specifies the transparency values over time. See <PCF-SPEC> above
PCF values are transparency values.

DOCUMENTATION:

Returns data describing the current state of the telescope

(defun IGNORE-ALL (&key (which-scheduler :lt) (permanent t))

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

permanent - either t or nil. default is t.

DOCUMENTATION:

Ignores all tasks in the schedule.

If permanent = t the tasks will be ignored until an explicit unignore
command is executed.

if permanent = nil the tasks are ignored only for the current schedule.

If you execute a new auto scheduler run, these vars are automatically
unignored and rescheduled.

Any tasks which are locked are left locked

Any tasks which are assigned are first make it unassigned and then ignored.

(defun UNIGNORE-ALL (&key (which-scheduler :lt) (permanent t))

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Unignores all ignored tasks in the schedule.

If permanent = t then all the tasks whether they are permanently ignored or
not are unignored

If permanent = nil then all the tasks that are ignored are unignored. If
you run an auto scheduler again, the tasks that are permanently ignored
are marked ignored again.

Any tasks locked are left locked.

Any tasks assigned are left assigned.

(defun LOCK-ALL (&key (which-scheduler :lt))

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Locks all assigned tasks in the scheduled.

(defun LOCK-BEFORE (time &key (which-scheduler :lt))

ARGS:

time - See <TIME-SPEC> above. required arg

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Locks all assigned tasks in the schedule that are scheduled before time.

(defun UNLOCK-ALL (&key (which-scheduler :lt))

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Unlocks all locked tasks in the scheduled.

(defun UNASSIGN-ALL (&key (which-scheduler :lt))

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Unassigns all assigned tasks in the scheduled.

Locked and ignored tasks are not affected.

Task Operations

This section describes a set of functionalities supported on tasks

(defun IGNORE-TASK (task &key (which-scheduler :lt) (permanent t))

ARGS:

task - name of the task -> string, required arg
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt
permanent - t or nil, default is t.

DOCUMENTATION:

Ignores the specified task.

If permanent = t the task will be ignored until an explicit unignore
command is executed.

If permanent = nil the task is ignored only for the current schedule.

If you execute a new auto scheduler run, this task is automatically
unignored and rescheduled.

If the task is locked, nothing happens

If the task is assigned it is first make it unassigned and then ignored.

(defun UNIGNORE-TASK (task&key (which-scheduler :lt) (permanent t))

ARGS:

task - name of the task -> string, required arg
which-scheduler - either :lt or :st for long term or short term respectively
default is :lt
permanent - t or nil, default is t.

DOCUMENTATION:

Unignores the specified task.

If permanent = t then the task is unignored whether it is permanently ignored or not.

If permanent = nil then the task is unignored for now. If you run an auto scheduler again, the task will be marked ignored again

Any tasks locked are left locked.

Any tasks assigned are left assigned.

(defun LOCK-TASK (task &key (which-scheduler :lt))

ARGS:

task - name of the task -> string, required arg

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Locks the specified task. It should already be assigned

(defun UNLOCK-TASK (task &key (which-scheduler :lt))

ARGS:

task - name of the task -> string, required arg

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

DOCUMENTATION:

Unlocks the specified task. It is still assigned, but no longer locked

If task is ingored, no action taken.

If task is assigned but not locked, no action taken

If task is unassigned no action taken

(defun ASSIGN-TASK (task time &key (which-scheduler :lt) (override nil))

ARGS:

task - name of the task -> string, required arg

time - start time where the task should be scheduled. See <TIME-SPEC> above

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

override - t or nil, default is nil. If override = t, then make the assignment
Even if the scheduler says that the task is unschedulable at that time.
If override = nil and scheduler thinks the time specified is unschedulable
no assignment is made.

DOCUMENTATION:

Assigns the specified task to time

If permanent = t then all the tasks whether they are permanently ignored or not are unignored

If permanent = nil then all the tasks that are ignored are unignored. If you run an auto scheduler again, the tasks that are permanently ignored are marked ignored again.

Any tasks locked are left locked.

Any tasks assigned are left assigned.

(defun ASSIGN-TASK-BY-STRATEGY (task strategy &key (which-scheduler :lt))

 ARGS:

 task - name of the task -> string, required arg
 strategy - one of
 :first-possible-time -> assign task at 1st possible time
 :last-possible-time -> assign task at last possible time
 :first-min-conflict-time -> assign task at 1st minimum conflict time
 :last-min-conflict-time -> assign task at last minimum conflict time
 :max-pref-min-conflict-time -> assign task at max preference minimally
 conflicted time
 required arg
 which-scheduler - either :lt or :st for long term or short term respectively
 default is :lt

 DOCUMENTATION:

 Assigns a task based on the strategy selected
 If task is ignored, nothing happens
 If task is locked, nothing happens
 If task is assigned to the same time, nothing happens
 If task is assigned to another time, reassign it if the time specified is
 a valid schedulable time.

(defun UNASSIGN-TASK (task &key (which-scheduler :lt))

 ARGS:

 task - name of the task -> string, required arg
 which-scheduler - either :lt or :st for long term or short term respectively
 default is :lt

 DOCUMENTATION:

 Unassigns the task
 If task locked or ignored, nothing happens.
 If task unassigned, nothing happens

(defun TASK-STATUS (task &key (which-scheduler :lt))

 ARGS:

 task - name of the task -> string, required arg
 which-scheduler - either :lt or :st for long term or short term respectively
 default is :lt

 RETURN-VALUES:

 Returns a list of keyword value pairs containing information about the task.

 Format (:key1 val1 :key2 val2 ...)

 Currently defined key/value pairs

 key value
 :name name of task -> string
 :state one of :locked, :assigned, :ignored, :unassigned
 :assigned-time only valid if state = :assigned | :locked

	time where the task is scheduled
	See <TIME-SPEC> above
:conflicts-on-assign	only valid if state = :assigned :locked number of conflicts at assigned time
:task-duration	only valid if state = :assigned :locked duration of task where it is scheduled, in days
:min-duration	minimum possible duration of task, in days
:telescope-pcf	Specifies the availability of the telescope over time. See <PCF-SPEC> above. PCF values are suitabilities.
:instrument-pcf	Specifies the availability of the instrument over time. See <PCF-SPEC> above. PCF values are suitabilities.
:moon-illum-pcf	Specifies goodness over time related to the moon illumination. See <PCF SPEC> above. PCF values are suitabilities.
:moon-target-exclusion-pcf	Specifies goodness over time related to the moon target angular separation See <PCF SPEC> above. PCF values are suitabilities.
:moon-pcf	Specifies goodness over time related to the moon. See <PCF SPEC> above. PCF values are suitabilities. Not applicable for calibration observations.
:ra-pcf	Specifies goodness over time related to the RA distance from the meridian. See <PCF SPEC> above. PCF values are suitabilities. Not applicable to LTS tasks.
:windows-pcf	Represents the user defined windows. See <PCF-SPEC> above. PCF values are suitabilities.
:phases-pcf	Represents the user defined phases. See <PCF-SPEC> above. PCF values are suitabilities.
:sun-pcf	Specifies goodness over time related to the sun. See <PCF-SPEC> above. PCF values are suitabilities. Only supplied when which-scheduler = :st. Not applicable for calibration observations.
:orb-viewing-pcf	Specifies goodness over time related to target viewing. See <PCF-SPEC> above. PCF values are suitabilities. Only supplied when which-scheduler = :st.
:current-seeing-pcf	Specifies goodness over time related to the current seeing. See <PCF-SPEC> above. PCF values are suitabilities. Only supplied when which-scheduler = :st. Not applicable for calibration observations.
:sky-transparency-pcf	Specifies goodness over time related to transparency, See <PCF-SPEC> above. PCF values are suitabilities. Only supplied when which-scheduler = :st. Not applicable for calibration observations.
:override-pcf	Specifies the override pcf for the task over time. See <PCF-SPEC> above. PCF values are suitabilities.
:airmass-pcf	Specifies goodness over time related to airmass or closeness to meridian. See <PCF-SPEC> above. PCF values are values from the MERIDIAN-SUITABILITY-MAPPING. Only supplied when which-scheduler = :st.
:preference-pcf	Specifies the preference for the task over time. See <PCF-SPEC> above

PCF values are preferences.

:conflicts-pcf Specifies the conflicts for the task over time. See <PCF-SPEC> above. PCF values are number conflicts

:efficiency-pcf Specifies efficiency over time. See <PCF-SPEC> above. PCF values are efficiency values. Only supplied when which-scheduler = :st

:computed-efficiency-pcf Specifies efficiency over time. See <PCF-SPEC> above. PCF values are efficiency values. Only supplied when which-scheduler = :st. Not applicable for Calibration observations.

:duration-pcf Specifies task duration over time. See <PCF-SPEC> above. PCF values are task durations, in seconds. Only supplied when which-scheduler= :st

:rel-constraints-pcf Specifies goodness over time for a user specified relative constraints. Only present if specified by proposer. See <PCF-SPEC> above. PCF values suitabilities.

:abs-constraints-pcf Specifies goodness over time for a user specified absolute constraints. See <PCF-SPEC> above. PCF values suitabilities.

:suitability-pcf Specifies goodness over time for the rel-constraints and abs-constraints. See <PCF-SPEC> above. PCF values are suitabilities

:total-suitability-pcf Specifies the task's overall suitability over time. This includes suitability-pcf, telescope-pcf, instrument-pcf.

DOCUMENTATION:

Returns status information about the task

(defun EXPLAIN-CONFLICTS (&rest arguments &key (which-scheduler :lt))

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
 default is :lt

RETURN-VALUES:

For each assigned task in the scheduler that has conflicts with other assigned tasks, explains why there are conflicts and in what time periods. If no conflicts or no assigned tasks, nil is returned.

Format

```
((:affected-task <affected-task-id> :assignment <TIME-SPEC>
  <affecting-task-list1> ... <affecting-task-listN>)
  ....)
<affecting-task-list> has the following format
(:affecting-task <affecting-task-id> :reason <reason> :conflicted-value <TIME-SPEC>)
```

<affected-task-id> is the id for the task whose assignment has conflicts the each of the affecting task's assignments
 <TIME-SPEC> - see above for definition
 <affecting-task-id> is the id for the task that has conflicts with <affected-task-id>'s assignment

<reason> is the reason why the conflict occurs. There are three primary reason returned now.

:overlap -> means that the tasks overlap in time or that the night demand conflicts (ie., both want whole nights)

:rel-long-term-scheduler - a relative timing constraint violation in the long term scheduler (ie., after violated, etc)

:rel-short-term-scheduler - a relative timing constraint violation in the short term scheduler (ie., after violated, etc)

EXAMPLE OUTPUT:

```
((:AFFECTED-TASK |OR 1| :ASSIGNMENT (1996 6 1 0 0 0 0)
 (:AFFECTING-TASK OR2 :REASON :OVERLAP :CONFLICTED-VALUE (1996 6 3
0 0 0 0)))
 (:AFFECTED-TASK OR2 :ASSIGNMENT (1996 6 3 0 0 0 0)
 (:AFFECTING-TASK |OR 1| :REASON :OVERLAP :CONFLICTED-VALUE (1996 6 1
0 0 0 0))))
```

DOCUMENTATION:

Explains all the conflicts for all assigned tasks

(defun EXPLAIN-UNSCHEDULABLES (&rest arguments &key (which-scheduler :lt))

ARGS:

which-scheduler - either :lt or :st for long term or short term respectively
default is :lt

RETURN-VALUES:

For each unschedulable observation, determines why the observation is unschedulable. Returns a list of lists where each sublist is a list of explanation strings explaining why an observation is unschedulable.

Example;

```
((:TASK-NAME "ob2" :REASONS ("Combination of relative and absolute constraints for ob2 cause ob2 to
be unscheduable"))
 (:TASK-NAME "ob5" :REASONS
 ("Absolute Constraints ('Moon' 'Current Seeing') for ob5 cause ob5 to be unscheduable."
 "Absolute Constraints ('Moon Target Exclusion' 'Current Seeing') for ob5 cause ob5 to be unscheduable."
 "Absolute Constraints ('Moon Illumination' 'Current Seeing') for ob5 cause ob5 to be unscheduable."
 "Absolute Constraints ('Sky Transparency' 'Current Seeing') for ob5 cause ob5 to be unscheduable."))))
```

Target Operations

(defun SUN-POSITION-FOR-TARGET (ra dec time1 time2)

ARGS:

ra - right ascension of target in J2000, required arg
dec - declination of target in J2000, required arg
time1 - start time for calculating position info -> See <TIME-SPEC> above.
required arg
time2 - end time for calculating position info -> See <TIME-SPEC> above.
required arg

RETURN-VALUES:

Returns a list of lists. Each sublist has the following format

((t1 ra1 dec1 a1) ... (tN raN decN aN))

The sun is at position ra1 and dec1 at t1. The distance between the sun and the target is a1
t1-tN - are times. See <TIME-SPEC> above.

ra1-raN - sun right ascension

dec1-decN - sun declination

a1-aN - angular distance between sun and target

DOCUMENTATION:

Loop from time1 to time2 calculating the position of the sun and the angular distance between the sun and the target.

(defun SUN-POSITION-FOR-TASK (task time1 time2 &key (which-scheduler :st))

ARGS:

task - name of task -> string, required arg

time1 - start time for calculating position info -> See <TIME-SPEC> above.
required arg

time2 - end time for calculating position info -> See <TIME-SPEC> above.
required arg

which-scheduler - either :lt or :st for long term or short term respectively
short term supported only. Long term has no targets

RETURN-VALUES:

Returns a list of lists. Each sublist has the following format

((t1 ra1 dec1 a1) ... (tN raN decN aN))

The sun is at position ra1 and dec1 at t1. The distance between the sun and the target is a1
t1-tN - are times. See <TIME-SPEC> above.

ra1-raN - sun right ascension

dec1-decN - sun declination

a1-aN - angular distance between sun and target

DOCUMENTATION:

Loop from time1 to time2 calculating the position of the sun and the angular distance between the sun and the task's target.

(defun MOON-POSITION-FOR-TARGET (ra dec time1 time2)

ARGS:

ra - right ascension of target in J2000, required arg

dec - declination of target in J2000, required arg

time1 - start time for calculating position info -> See <TIME-SPEC> above.
required arg

time2 - end time for calculating position info -> See <TIME-SPEC> above.
required arg

RETURN-VALUES:

Returns a list of lists. Each sublist has the following format

((t1 ra1 dec1 a1) ... (tN raN decN aN))

The moon is at position ra1 and dec1 at t1. The distance between the moon and target is a1
t1-tN - are times. See <TIME-SPEC> above.

ra1-raN - moon right ascension

dec1-decN - moon declination

a1-aN - angular distance between moon and target

DOCUMENTATION:

Loop from time1 to time2 calculating the position of the moon and the angular distance between the moon and the target.

```
(defun MOON-POSITION-FOR-TASK (task time1 time2 &key (which-scheduler :st))
```

ARGS:

- task - name of task -> string, required arg
- time1 - start time for calculating position info -> See <TIME-SPEC> above.
required arg
- time2 - end time for calculating position info -> See <TIME-SPEC> above.
required arg
- which-scheduler - either :lt or :st for long term or short term respectively
short term supported only. Long term has no targets

RETURN-VALUES:

Returns a list of lists. Each sublist has the following format

((t1 ra1 dec1 a1) ... (tN raN decN aN))

The moon at position ra1 and dec1 at t1. The distance between the moon and the target is a1
t1-tN - are times. See <TIME-SPEC> above.

ra1-raN - moon right ascension

dec1-decN - moon declination

a1-aN - angular distance between moon and target

DOCUMENTATION:

Loop from time1 to time2 calculating the position of the moon and the angular distance between the moon and the task's target

7.0 Concept of Operation Example

This section provides some examples of how this interface might be used.

Example 1 - Assign a few new tasks time periods with max-pref algorithm

```
(prep-obs-set :name "lts-1") ;; calculate all out of date products, maybe nightly  
(init-system :name "lts-1") ;; loads and inits system - little overhead since prep done  
(restore-schedule :name "baselined-lts-1" ;; load baselined assignments  
(max-pref :name "test-1" :tasks ("task1" "task2")) ;; sched set of tasks with max pref  
(schedule-status) ;; get info about current schedule  
(lock-task "task1") ;; lock assignment  
(lock-task "task2") ;; lock assignment  
(schedule-report-ps :file "../report.ps") ;; save ps plot  
(save-schedule :name "baselined-lts-2") ;; save new baselined schedule
```

Example 1 - Assign a unassigned tasks time periods

```
(init-system :name "lts-1") ;; loads and inits system - little overhead since prep done  
(restore-schedule :name "baselined-lts-1" ;; load baselined assignments  
(schedule-status) ;; get info about current schedule - especially unassigned tasks  
(max-pref :name "test-1" :tasks unassigned-tasks) ;; schedule with max-pref
```

(schedule-status) ;; get info about current schedule
 (hi-prior-max-pref :name "test-2" :tasks unassigned-tasks) ;; schedule with hi-prior-max-pref
 (schedule-status) ;; get info about current schedule
 (early-greedy :name "test-3" :tasks unassigned-tasks) ;; schedule with early-greedy
 (schedule-status) ;; get info about current schedule
 (select-schedule :name "test-2") ;; like test2 assignments the best
 (lock-all-tasks) ;; lets lock all the test-2 assignments
 (schedule-report-ps :file "../report.ps") ;; save ps plot
 (save-schedule :name "baselined-lts-2") ;; save new baselined schedule

8.0 Astronomical Constraints for the Short Term Scheduler

1) sun-pcf = Calculate times when the sun is up. This is done by calculating the angular distance from the zenith to the sun. If the sun angle is < the specified angular distance (ie., see **SUN-TWILIGHT-ANGLE** in **DEF-ST-SCHEDULER-PARAMETERS**), then the sun is up and this is an invalid scheduling time.

2) moon-pcf = Calculate times when the moon is up. This is done by calculating the angular distance from the zenith to the moon. If the moon angle is < the specified angular distance (ie., see **MOON-TWILIGHT-ANGLE** in **DEF-ST-SCHEDULER-PARAMETERS**), then the moon is up. Times when the moon is set are valid for scheduling. When the moon is up, need to check target/moon separation and moon phase. Both constraints need to be satisfied for a valid scheduling time.

2.1) moon-target-exclusion-pcf = Calculate moon target angular separation for times when the moon is up. Remove periods where the angular separation is less than the allowable minimal separation specified for the observaion block.

2.2) moon-illum-pcf = Calculate moon phase (0 new moon, 1 full moon) for times when the moon is up. Remove periods where the moon phase is larger than the maximal allowed phase specified on the observation block.

3) Calculate the task's suitability pcf and efficiency pcf.

3.1) abs-suitability-pcf = Calculate a suitability pcf (ie., suitability over time pcf) based on how far from the zenith the target is. Uses the **MERIDIAN-SUITABILITY-MAP** on the **DEF-OBS-BLOCK** first and then looks for the default specified in the **DEF-ST-SCHEDULER-PARAMETERS** to determine the suitabilities. This map translates angular distance from zenith to suitability value.

3.2) Calculate an efficiency pcf (ie., efficiency over time pcf) based on how far from the zenith the target is. Uses the **MERIDIAN-EFFICIENCY-MAP** on the **DEF-OBS-BLOCK** first and then looks for the default specified in the **DEF-ST-SCHEDULER-PARAMETERS** to determine the efficiencies. This map translates angular distance from zenith to efficiency value.

3.3) Remove times when sun is up from both the abs-suitability-pcf and efficiency-pcf.
Can't schedule when sun is up

$\text{abs-suitability-pcf} = \text{abs-suitability-pcf} * \text{sun-pcf}$ (sun-pcf is 0 when sun is up)
 $\text{efficiency-pcf} = \text{efficiency-pcf} * \text{sun-pcf}$ (sun-pcf is 0 when sun is up)

3.4) Remove times when the moon is up and either the moon/target separation or moon phase constraints are violated. If moon is set, moon does not affect scheduling
 $\text{abs-suitability-pcf} = \text{abs-suitability-pcf} * \text{moon-illum-pcf} * \text{moon-target-exclusion-pcf}$
 $\text{efficiency-pcf} = \text{efficiency-pcf} * \text{moon-illum-pcf} * \text{moon-target-exclusion-pcf}$

3.5) Don't schedule when efficiency value is < min-efficiency
 $\text{efficiency-pcf} = \text{set all periods to 0 (ie., bad times) when efficiency} < \text{min-efficiency}$
(see **MIN-EFFICIENCY** in **DEF-ST-SCHEDULER-PARAMETERS**).

4) Compute a duration pcf which says how long a task will take if it starts at time X. This is calculated based on the abs-suitability-pcf and efficiency-pcf

Example.

- task takes 60 seconds (see **TASK-DURATION** in **DEF-OBS-BLOCK**)
- If efficiency value = 1 then task duration = 60 sec
- If efficiency value = .5 then task duration = 120 sec
- If efficiency value = 0 then task duration = 0 -> invalid time

5) Add in any task specific constraints into abs-suitability-pcf.

Task specific constraints are defined by **DEF-WINDOW** or **DEF-PHASE**

$\text{abs-suitability-pcf} = \text{abs-suitability-pcf} * \text{window-pcf} * \text{phase-pcf}$
(ie., window-pcf & phase-pcf are 0 when illegal to schedule)

6) Set up a rel-suitability-pcf based on task specific relative constraints.

Task specific constraints are defined by **DEF-ORDER**, **DEF-ORDER-AND-OFFSET**, **DEF-GROUP-WITHIN**.

7) A tasks overall suitability-pcf is the combination of the abs-suitability-pcf, and rel-suitability-pcf.