AX-TOOLBOX

Flight Analysis Scripting Language

Table of Contents

0 INTRODUCTION

0.1 Language basics

The flight analysis scripting language have two kind of instructions: settings and object constructors. The settings are used to set up some program options and variables and the constructors are used to define the flight tasks.

0.1.1 Settings

The setting instructions have the following syntax:

```
SET <key> = <one or more values>
```

The distinct values are separated by commas.

0.1.2 Constructors

The constructor instructions have the following syntax:

```
<class> <name> = <type>(<zero or more parameters>)
```

The distinct parameters are separated by commas.

0.1.3 Comments and blank characters

Any line whose first non space characters are double slashes '//' will be ignored. All blank lines are also ignored as well as all blank characters (spaces or tabs) before or after any symbol (instruction, name, key, type, value or parameter).

0.2 Notation Conventions

0.2.1 Names

Names can be any sequence of characters. All characters are allowed except commas ','. Names are case sensitive.

0.2.2 Distances and altitudes

Distance and altitude definitions are double precision floating point numbers with dot '.' decimal separator. The known units of measure are meters 'm', kilometers 'km', feet 'ft', miles 'mi' or nautical miles 'nm'.

Examples:

```
500m
1640.42ft
```

0.2.3 Time definitions

Time definitions are in the format 'HH:MM:SS' (hours, minutes and seconds). The seconds part is optional.

Applies to absolute times (time of day) as well to relative times (time span).

Examples:

```
10:30:00
10:30
```

0.2.4 Colors

Only the following colors are allowed: blue, brown, gray, green, orange, pink, red, violet, white and yellow.

1 SET INSTRUCTION

```
SET <key> = <value>[, <value>...]
```

Allows to set different options.

1.1 Definition

1.1.1 Key TITLE: competition title

```
set TITLE = <title>
```

Sets the competition title (used in automatic flight report)

1.1.2 Key SUBTITLE: competition title

```
set SUBTITLE = <subtitle>
```

Sets the competition subtitle (used in automatic flight report)

1.1.3 Key LOGFILE: log file

```
set LOGFILE = <filename>
```

Sets the file where to log all scripting operations

<filename> is the name of the log file

1.1.4 Key ALTITUDECORRECTIONSFILE: logger altitude corrections file

```
set ALTITUDECORRECTIONSFILE = <filename>
```

Sets the file with the logger altitude corrections

<filename> is the name of the file

1.1.5 Key DATETIME: flight date and time

```
set DATETIME = <date>, <time>
```

Sets the flight date and time

<date> is the flight date in format yyyy/mm/dd

<time> is either AM or PM

1.1.6 Key DATUM

```
set DATUM = <datumName>
```

Sets the default datum.

<datumName> is the datum name

Supported datums: Corrego Alegre, European 1950, NAD27 CONUS, OSGB36, WGS72, WGS84.

1.1.7 Key UTMZONE

```
set UTMZONE = <zone>
```

Sets the default UTM zone.

<zone> is the UTM zone

1.1.8 Key TASKSINORDER

```
set TASKSINORDER = <value>
```

Sets whether the tasks must be flown in order or not. If this setting is not present it is assumed as true.

<value> is true or false

1.1.9 Key QNH

```
set QNH = <value>
```

Sets the QNH value

<value> is the qnh value

1.1.10 Key SMOOTHNESS

```
set SMOOTHNESS = <value>
```

Sets the smoothness factor (moving average values) used in automatic launch and landing detection. If undefined, 3 will be assumed.

<value> is the smoothness factor

1.1.11 Key MINSPEED

```
set MINSPEED = <speed>
```

Sets the minimum speed (in m/s) considered as movement in automatic launch and landing detection. If undefined, 0.5 will be assumed.

<speed> is the minimum speed

1.1.12 Key MAXACCELERATION

```
set MAXACCELERATION = <acceleration>
```

Sets the maximum acceleration (in m/s^2) allowed in normal flight. Track points with greater acceleration will be discarded as bogus (spikes). If undefined, 0.3 will be assumed.

<acceleration> is the maximum acceleration

1.2 Examples

```
set DATETIME = 2010/09/19, AM
set DATUM = European 1950
```

```
set QNH = 1013
set MINSPEED = 0.3
set MAXDISTTOCROSSING = 300m
```

2 MAP CLASS

```
MAP <mapName> = <mapType>(<map params>)
```

There must be one and only one map file per script. If more than one is defined, only the last is used. The area covered by the map is assumed to be the competition area.

2.1 Constructors

```
<mapType>(<map params>)
```

2.1.1 Type BITMAP: Bitmap file

```
BITMAP(<fileName>)
```

<fileName> is a bitmap file name. Compatible bitmap types are bmp, gif, jpg, png, tif. A corresponding world file must exist. See http://en.wikipedia.org/wiki/World_file for more information.

2.1.2 Type BLANK: Blank map

```
Blank(<topLeft>, <bottomRight>)
```

<topLeft> is a previously defined point name designing the top-left corner of the map.
<bottomRight> is a previously defined point name designing the bottom-right corner of the map.

2.2 Display modes

```
<displayMode>(<display params>)
```

2.2.1 Mode GRID

```
GRID(<width>)
```

<width> is the desired grid spacing between lines.
Draws a grid over the map.

2.3 Examples

```
MAP CompetitionMap = BITMAP(map.png) grid(1000)
MAP OfficialMap = Blank(TopLeftPoint,BottomRightPoint)
```

3 TASK CLASS

```
TASK <taskName> = <taskType>(<number>)
```

<taskType> is one of the AXMER allowed task type.
<number> is the task number in the competition.

3.1 Constructors

```
<taskType>(<number>)
```

3.1.1 PDG - Pilot Declared Goal

```
PDG(<number>)
```

3.1.2 JDG - Judge Declared Goal

```
JDG(<number>)
```

3.1.3 HWZ - Hesitation Waltz

```
HWZ(<number>)
```

3.1.4 FIN - Fly In

```
FIN(<number>)
```

3.1.5 FON - Fly On

```
FON(<number>)
```

3.1.6 HNH - Hare And Hounds

```
HNH(<number>)
```

3.1.7 WSD - Watership Down

```
WSD(<number>)
```

3.1.8 GBM - Gordon Bennett Memorial

```
GBM(<number>)
```

3.1.9 CRT - Calculated Rate Of Approach

```
CRT(<number>)
```

3.1.10 RTA - Race To An Area

```
RTA(<number>)
```

3.1.11 ELB - Elbow

```
ELB(<number>)
```

3.1.12 LRN - Land Run

```
LRN(<number>)
```

3.1.13 MDT - Minimum Distance

```
MDT(<number>)
```

3.1.14 SFL - Shortest Flight

```
SFL(<number>)
```

3.1.15 MDD - Minimum Distance Double Drop

```
MDD(<number>)
```

3.1.16 XDT - Maximum Distance Time

```
XDT(<number>)
```

3.1.17 XDI - Maximum Distance

```
XDI(<number>)
```

3.1.18 XDD - Maximum Distance Double Drop

```
XDD(<number>)
```

3.1.19 ANG - Angle

```
ANG(<number>)
```

3.1.20 3DT - 3D Shape Task

```
3DT(<number>)
```

3.2 Examples

```
TASK Task3HWZ = HWZ(5)
TASK Task2 = JDG(3)
```

4 AREA CLASS

```
AREA <name> = <areaType>(<area params>) [<displayMode>(<display
params>)]
```

4.1 Constructors

```
<areaType>(<area params>)
```

4.1.1 Type CYLINDER: Circular or cylindrical area

```
CYLINDER(<center>, <radius>[, <upperLimit>[, <lowerLimit>]])
```

 is a point name used as the circle center
<radius> is the circle radius
<upperLimit> is the upper altitude limit (above is outside)
<lowerLimit> is the lower altitude limit (below is outside)

4.1.2 Type SPHERE: Spherical area

```
SPHERE(<center>, <radius>)
```

 is a point name used as the sphere center
<radius> is the sphere radius

4.1.3 Type PRISM: Prismatic area

```
PRISM(<fileName>[, <upperLimit>[, <lowerLimit>]])
```

<fileName> is the name of a track file containing the polygon (trk, igc) used as base
<upperLimit> is the upper altitude limit (above is outside)
<lowerLimit> is the lower altitude limit (below is outside)

4.2 Display modes

```
<displayMode>(<display params>)
```

The display mode is optional. If it is not specified, DEFAULT(blue) is used.

4.2.1 Mode NONE

```
NONE()
```

The area will not be displayed.

4.2.2 Mode DEFAULT

```
DEFAULT([<color>])
```

Draws the area according to its type.

<color> is one of blue, brown, gray, green, orange, pink, red, violet, white, yellow. If no color is specified, blue will be used.

4.3 Examples

```
area Area1 = CYLINDER(P10, 100m) DEFAULT(green)
area Area2 = PRISM(area2.trk, 100m, 1000m)
area Area3 = SPHERE(P10, 500) NONE()
```

5 FILTER CLASS

Filter track points. Points filtered out are not considered in further computations. Successive filters are applied in definition order.

If the filter is defined before any task, it is considered as the competition area definition and applied automatically to all tasks.

```
FILTER <name> = <filterType>(<filter params>)
```

5.1 Constructors

5.1.1 INSIDE

```
INSIDE(<areaName>)
```

Selects all the track points inside the specified area.

<areaName> is the name of a previously defined area

5.1.2 OUTSIDE

```
OUTSIDE(<areaName>)
```

Selects all the track points outside the specified area.

<areaName> is the name of a previously defined area

5.1.3 BEFORETIME

```
BEFORETIME(<time>)
```

Selects all the track points earlier than specified time.

<time> desired time in hh:mm:ss

5.1.4 AFTERTIME

```
AFTERTIME(<time>)
```

Selects all the track points later than specified time.

<time> desired time in hh:mm:ss

5.1.5 BEFOREPOINT

```
BEFOREPOINT(<pointName>)
```

Selects all the track points earlier than specified point.

<pointName> is the name of a previously defined point

5.1.6 AFTERPOINT

```
AFTERPOINT(<pointName>)
```

Selects all the track points later than specified point.

<pointName> is the name of a previously defined point

5.1.7 ABOVE

```
ABOVE(<altitude>)
```

Selects all the track points above a given altitude

<altitude> is the desired altitude limit in feet

5.1.8 BELOW

```
BELOW(<altitude>)
```

Selects all the track points below a given altitude

<altitude> is the desired altitude limit in feet

5.1.9 NONE

```
NONE()
```

Clears all the active filters for the current task. Previous computations are not affected.

5.2 Examples

```
FILTER InArea = INSIDE(GBMArea)
FILTER Before10:00 = BEFORETIME(10:00:00)
```

6 POINT CLASS

```
POINT <name> = <pointType>(<point params>) [<displayMode>(<display
params>)]
```

6.1 Static point (S) constructors

The S constructors return a statically defined point.

6.1.1 Type SLL: WGS84 lat/long

```
SLL(<lat>, <long>, <altitude>)
```

Returns a statically defined point using WGS lat/lon coordinates.

<lat> is the latitude coordinate
<long> is the longitude coordinate
<altitude> is the altitude (in feet)

6.1.2 Type SUTM: UTM

```
SUTM(<easting>, <northing>, <altitude>)
```

Returns a statically defined point using UTM coordinates in the default datum and zone.

<easting> is the easting coordinate
<northing> is the northing coordinate
<altitude> is the altitude (in feet)

6.2 Point from a list (L) constructors

The L constructors return a point chosen form a list of predefined points.

6.2.1 Type LNP: nearest to point

```
LNP(<desiredPoint>, <listPoint1>, <listPoint2>, ...,
<altitudeThreshold>)
```

Returns the point closest to a desired one from a list of predefined points. It uses altitude dependent distance measurement (see DRAD result class).

<altitudeThreshold>: distance 2D measurement is used below this altitude difference. 3D otherwise.

6.2.2 Type LFT: first in time

```
LFT(<listPoint1>, <listPoint2>, ...)
```

Returns the earliest point from a list of points.

6.2.3 Type LLT: last in time

```
LLT(<listPoint1>, <listPoint2>, ...)
```

Returns the latest point from a list of points.

6.2.4 Type LFNN: first not null

```
LFNN(<listPoint1>, <listPoint2>, ...)
```

Returns the first non null point from a list of points.

6.2.5 Type LLNN: last not null

```
LLNN(<listPoint1>, <listPoint2>, ...)
```

Returns the last non null point from a list of points.

6.3 Logger mark (M) constructors

The M constructors return a point defined with a logger mark.

6.3.1 Type MVMD: virtual marker drop

```
MVMD(<number>)
```

Returns the first occurrence of the desired virtual marker drop. The filters applied to track points are enforced because virtual marker drops must coincide with a valid track point.

<number> desired marker number

6.3.2 Type MPDGL: pilot declared goal before launch

```
MPDGL(<number>, <defaultAltitude>)
```

Returns the last occurrence of the desired pilot declared goal within time limits.

<number> desired pilot declared goal number
<defaultAltitude> altitude to apply if not defined in the declaration

6.3.3 Type MPDGF: pilot declared goal in flight

```
MPDGF(<number>, <defaultAltitude>)
```

Returns the last occurrence of the desired pilot declared goal within time limits.

<number> desired pilot declared goal number
<defaultAltitude> altitude to apply if not defined in the declaration

6.4 Track point (T) constructors

The T constructors return a point from the track log.

6.4.1 Type TLCH: launch

```
TLCH()
```

Returns the launch point.

6.4.2 Type TLND: landing

```
TLND()
```

Returns the landing point.

6.4.3 Type TPT: on point time

```
TPT(<pointName>)
```

Returns the track point with the same timestamp as a given point.

<pointName> is the name of a previously defined point

6.4.4 Type TNP: nearest to point

```
TNP(<pointName>, <altitudeThreshold>)
```

Returns the closest valid track point to the specified point.

<pointName> is the name of a previously defined point
<altitudeThreshold>: distance 2D measurement is used below this altitude difference. 3D otherwise.

6.4.5 Type TNL: nearest to point list

```
TNL(<listPoint1>, <listPoint2>, ..., <altitudeThreshold>)
```

Returns the closest valid track point to any the specified points.

<listPoint> is the name of another previously defined point
<altitudeThreshold>: distance 2D measurement is used below this altitude difference. 3D otherwise.

6.4.6 Type TDT: delayed in time

```
TDT(<pointName>, <timeDelay>[, <maxTime>])
```

If <maxTime> is not specified, returns the first valid track point after <timeDelay> minutes have elapsed over the desired point time.

If <maxTime> is specified, returns the last valid track point after <timeDelay> minutes have elapsed over the desired point time but before <maxTime>.

<pointName> is the name of another previously defined point
<timeDelay> desired time delay in minutes
<maxTime> desired time limit. This parameter is optional

6.4.7 Type TDD: delayed in distance

```
TDD(<pointName>, <distanceDelay>[, <maxTime>])
```

If <maxTime> is not specified, returns the first valid track point farther than <distanceDelay> from the desired point.

If <maxTime> is specified, returns the last valid track point before <maxTime> farther than <distanceDelay> from the desired point.

<pointName> is the name of another previously defined point
<distanceDelay> desired distance delay in meters
<maxTime> desired time limit. This parameter is optional

6.4.8 Type TAFI: area first in

```
TAFI(<areaName>)
```

Returns the first valid track point after entering the area for the first time.

<areaName> is the name of a previously defined area

6.4.9 Type TAFO: area first out

```
TAFO(<areaName>)
```

Returns the last valid track point before exiting the area for the first time.

<areaName> is the name of a previously defined area

6.4.10 Type TALI: area last in

```
TALI(<areaName>)
```

Returns the first valid track point after entering the area for the last time.

<areaName> is the name of a previously defined area

6.4.11 Type TALO: area last out

```
TALO(<areaName>)
```

Returns the last valid track point before exiting the area for the last time.

<areaName> is the name of a previously defined area

6.5 Display modes

```
<displayMode>(<display params>)
```

The display mode is optional. If it is not specified, WAYPOINT(blue) is used.

6.5.1 Mode NONE

```
NONE()
```

The point will not be displayed.

6.5.2 Mode WAYPOINT

```
WAYPOINT([<color>])
```

Displays a waypoint map overlay.

<color> is one of blue, brown, gray, green, orange, pink, red, violet, white, yellow. If no color is specified, blue will be used.

6.5.3 Mode TARGET

```
TARGET(<radius>[, <color>])
```

Displays a target map overlay with a circular area of given radius.

<radius> circular area radius in meters.
<color> is one of blue, brown, gray, green, orange, pink, red, violet, white, yellow. If no color is specified, blue will be used.

6.5.4 Mode MARKER

```
MARKER([<color>])
```

Displays a flag map overlay.

<color> is one of blue, brown, gray, green, orange, pink, red, violet, white, yellow. If no color is specified, blue will be used.

6.5.5 Mode CROSSHAIRS

```
CROSSSHAIRS([<color>])
```

Displays a crosshairs map overlay.

<color> is one of blue, brown, gray, green, orange, pink, red, violet, white, yellow. If no color is specified, blue will be used.

6.6 Examples

```
point LAUNCH = TLCH() WAYPOINT(green)
point LANDING = TLND() WAYPOINT(red)
point G12 = SLL(42.12, 3.23, 1500) TARGET(100, green)
point G13 = SUTM(31T, 235123, 4612153, 850) TARGET(100, green)
point Best = TNL(G12, G13) MARKER(green)
```

7 RESULT CLASS

```
RESULT <resultName> = <resultType>(<result params>)
```

Computes a task result. The results are assigned to the last defined task.

7.1 Constructors

```
<resultType>(<result params>)
```

7.1.1 Type D2D: distance in 2D

```
D2D(<pointNameA>, <pointNameB>[, <bestPerformance>])
```

Computes the projected distance (in 2D) between two points.

<bestPerformance>: best performance achievable with GPS (R12.22.4). If it's not specified, is assumed to be zero.

7.1.2 Type D3D: distance in 3D

```
D3D(<pointNameA>, <pointNameB>[, <bestPerformance>])
```

Computes the distance in 3D between two points.

<bestPerformance>: best performance achievable with GPS (R12.22.4). If it's not specified, is assumed to be zero.

7.1.3 Type DRAD: relative altitude dependent distance

```
DRAD(<pointNameA>, <pointNameB>, <2DThreshold>[, <bestPerformance>])
```

Computes the distance between two points. If the altitude difference between the two is less than the threshold , 2D distance will be computed (R12.22.3). Otherwise, 3D distance is used.

<bestPerformance>: best performance achievable with GPS (R12.22.4). If it's not specified, is assumed to be zero.

7.1.4 Type DRAD10: relative altitude dependent distance rounded to decameter

```
DRAD10(<pointNameA>, <pointNameB>, <2DThreshold>[, <bestPerformance>])
```

Computes the distance, rounded to the decameter, between two points. If the altitude difference between the two is less than the threshold , 2D distance will be computed (R12.22.3). Otherwise, 3D distance is used.

<bestPerformance>: best performance achievable with GPS (R12.22.4). If it's not specified, is assumed to be zero.

7.1.5 Type DACC: accumulated distance

```
DACC(<pointNameA>, <pointNameB>)
```

Computes the sum of the distances between consecutive valid track points from point 1 to point 2. In other words, It computes the total traveled distance between the two points.

7.1.6 Type TSEC: time in seconds

```
TSEC(<pointNameA>, <pointNameB>)
```

Computes the time elapsed in seconds between the two points.

7.1.7 Type TMIN: time in minutes

```
TMIN(<pointNameA>, <pointNameB>)
```

Computes the time elapsed in minutes between the two points.

7.1.8 Type ATRI: area of triangle

```
ATRI(<pointNameA>, <pointNameB>, <pointNameC>)
```

Computes the area in km^2 of the triangle delimited by the three points.

7.1.9 Type ANG3P: angle between 3 points

```
ANG3P(<pointNameA>, <pointNameB>, <pointNameC>)
```

Computes the direction change between AB and BC (180 - the angle ABC).

7.1.10 Type ANGN: angle to the north

```
ANGN(<pointNameA>, <pointNameB>)
```

Computes the angle between the line AB and the line north-south.

7.1.11 Type ANGSD: angle to a set direction

```
ANGSD(<pointNameA>, <pointNameB>, <setDirection>)
```

Computes the angle between the line AB and a set direction. 0 is north-south.

7.2 Examples

```
RESULT Task3Result=DRAD(Task3Marker,Task3Target,500)
```

8 RESTRICTION CLASS

```
RESTRICTION <restrictionName> = <restrictionType>(<restriction
params>)
```

Computes a restriction. If a restriction is infringed, the pilot will not achieve a result for the task (group B).

8.1 Constructors

```
<restrictionType>(<restriction params>)
```

8.1.1 Type DMAX: maximum distance

```
DMAX(<pointNameA>, <pointNameB>, <distance>, <description>)
```

The 2D distance between point A and point B must not be greater than the specified distance.

<description> is the description of the infringement (rule number will be included automatically when known).

8.1.2 Type DMIN: minimum distance

```
DMIN(<pointNameA>, <pointNameB>, <distance>, <description>)
```

The 2D distance between point A and point B must not be lesser than the specified distance.

<description> is the description of the infringement (rule number will be included automatically when known).

8.1.3 Type DVMAX: maximum vertical distance

```
DVMAX(<pointNameA>, <pointNameB>, <height>, <description>)
```

The vertical distance between point A and point B must be greater than the specified height.

<description> is the description of the infringement (rule number will be included automatically when known).

8.1.4 Type DVMIN: minimum vertical distance

```
DVMIN(<pointNameA>, <pointNameB>, <height>, <description>)
```

The vertical distance between point A and point B must be lesser than the specified height.

<description> is the description of the infringement (rule number will be included automatically when known).

8.1.5 Type TMAX: maximum time

```
PBP(<pointNameA>, <pointNameB>, <time>, <description>)
```

The point B must not happen more than the the specified time in minutes later than point A.

<description> is the description of the infringement (rule number will be included automatically when known).

8.1.6 Type TMIN: minimum time

```
TMIN(<pointNameA>, <pointNameB>, <time>, <description>)
```

The point B must happen more than the the specified time in minutes later than point A.

<description> is the description of the infringement (rule number will be included automatically when known).

8.1.7 Type TBTOD: before time of day

```
TBTOD(<pointNameA>, <time>, <description>)
```

The point A must happen before the specified time of day.

<description> is the description of the infringement (rule number will be included automatically when known).

8.1.8 Type TATOD: after time of day

```
TATOD(<pointNameA>, <time>, <description>)
```

The point A must happen after the specified time of day.

<description> is the description of the infringement (rule number will be included automatically when known).

8.2 Examples

```
RESTRICTION t2_dmin = DMIN(launch, task02_target, 1000m, "launch too
close to target")
RESTRICTION t3_declTime = TATOD(declaration_point, 10:00:00, "late
declaration")
```

9 PENALTY CLASS

```
PENALTY <penaltyName> = <penaltyType>(<penalty params>)
```

Computes a penalty. The penalty is applied to each affected task. The penalty definitions should be placed after the last task.

9.1 Constructors

```
<penaltyType>(< penalty params>)
```

9.1.1 Type BPZ: blue PZ

```
BPZ(<area>, <description>)
```

If the flight path enters an BPZ area, a penalty is applied to each affected task.

9.1.2 Type RPZ: red PZ

```
RPZ(<area>, <description>)
```

If the flight path enters an RPZ area, a penalty is applied to each affected task.

//TODO: explain better

9.1.3 Type VSMAX: maximum vertical speed

```
VSMAX(<verticalSpeed>, <sensitivity>)
```

This class actually does not apply any paramateres. It is intended for information purposes only.

It adds a remark when the vertical speed is over the given speed for over *sensitivity* seconds.

//TODO: check the value of the applied penalty

10 FULL TASK EXAMPLES

10.1 Judge Declared Goal (nearest track point)

```
TASK Task02JDG = JDG(2)
POINT Task02target=UTM(31T,325000,4612000,650)
FILTER Task02scoringPeriod1=AFTER(09:00:00)
FILTER Task02scoringPeriod2=BEFORE(10:00:00)
POINT Task02marker=TNP(Task02target)
RESULT Task02result=DRAD(Task02target,Task02marker,500)
```

The first line defines the task type, JDG. The second line defines the target. The next two lines define the scoring period from 9:00 to 10:00. The fifth line defines the nearest track point to the target. The last line computes the result, a relative altitude dependent distance with an altitude limit of 500ft.

10.2 Hesitation Waltz (logger marker drop)

```
TASK Task03HWZ=HWZ(3)
POINT Task03t1=UTM(31T,325000,4612000,650)
POINT Task03t2=UTM(31T,325500,4612000,650)
POINT Task03t3=UTM(31T,325000,4612500,650)
POINT Task03marker=MVMD(3)
POINT Target=LNP(Task03marker,Task03t1,Task03t2,Task03t3)
RESULT Result=D3D(Marker,Target)
```

The first line defines the task type, HWZ. The next three lines define the targets. The fifth line defines the logger marker drop #3. The sixth line defines the closest target to the marker. The last line computes the result, a distance in 3D.

10.3 Gordon Bennett Memorial (logger marker drop)

```
TASK Task1=GBM(1)
AREA T1area=Area(task1area.trk)
POINT T1goal=SUTM(31T,3166732,4613455,350)
FILTER T1scoringPeriod=BEFORE(10:00:00)
POINT T1marker=MVMD(1)
RESULT=DRAD(T1Goal,T1marker,500)
```

The first line defines a GBM task. The second line defines the scoring area from a track file. The third line defines the goal, a fixed UTM point. The next line defines a scoring period until 10:00. The next defines the logger marker #1 (note that the scoring period filter also applies to VMDs). The last line computes a relative altitude dependent distance with a threshold of 500ft.

10.4 Maximum Distance Double Drop (first in / last out)

```
TASK Task4=XDD(4)
AREA T4area=Area(task4area.trk)
FILTER T4scoringPeriod=BEFORE(10:00:00)
POINT T4A=TAFI(T4area)
POINT T4B=TALO(T4area)
RESULT t4result=D2D(T4A,T4B)
```

The first line defines the task type, XDD. The second line defines the task area from a track file. The third line applies a filter: the valid track points are those before 10:00:00 (task scoring period up to 10:00). The next line defines the first scoring point, the first-in point. The next defines the second scoring point, the last-out point. The last line computes the result, a distance in 2D.

11 BUGS AND IDEAS

The point constructors LNP, TNP and TNL do not specify what kind of distance should be used (d2d, d3d or drad). Consider constructors that specify the distance method.

Example: LNPD2D, LNPD3D, LNPDRAD, TNPD2D, TNPD3D, TNPDRAD, etc.

Think about a method to score a 3DS task allowing entering and exiting the area multiple times.