FACILITY GROUP

The Facility Data Group provides a convenient way to access limited, common facility information without the need to transfer into specific Waypoint Data Groups for it. Like the Waypoint Data Groups, entry into the Facility Group is by means of the ICAO; the full ICAO is always the passport within the fs9gps module. There are no indexed variables within the Facility Data Group.

As one example of the usefulness of the Facility Group, consider the results of an ICAO Search of Ident = 'CI' using the all-facility IcaoSearchStartCursor = 'AVNW':

```
     ICAO     111
Idx 123456789012
0    VNZ     CI
1    NK3KCIDCI
2    NK5KCIUCI
3    NNZ     CI
4    NRJ     CI
5    NNZ     CI
6    WVC     CI
```

7 ICAO's are returned (IcaoSearchMatchedIcaosNumber = 7) with a mix of VOR, NDB, and Waypoint facilities. Ahead of time, the user may not know what type of facility might be returned by the ICAO Search. Consequently, if common information like Latitude and Longitude of a selected facility is needed, then the following ICAO transfers,

```
(L:ICAO_Index_Selected, enum) (>@c:IcaoSearchMatchedIcao)
(@c:IcaoSearchCurrentIcao) (>@c:WaypointAirportIcao)
(@c:IcaoSearchCurrentIcao) (>@c:WaypointVorIcao)
(@c:IcaoSearchCurrentIcao) (>@c:WaypointNdbIcao)
(@c:IcaoSearchCurrentIcao) (>@c:WaypointIntersectionIcao)
```

followed by,

```
(@c:WaypointAirportLatitude, degrees)
(@c:WaypointAirportLongitude, degrees)
(@c:WaypointVorLatitude, degrees)
(@c:WaypointVorLongitude, degrees)
(@c:WaypointNdbLatitude, degrees)
(@c:WaypointNdbLongitude, degrees)
(@c:WaypointIntersectionLatitude, degrees)
(@c:WaypointIntersectionLongitude, degrees)
```

could be used to cover all the bases.

Alternatively, the much simpler Facility Data Group could be used instead:

```
(L:ICAO_Index_Selected, enum) (>@c:IcaoSearchMatchedIcao)
(@c:IcaoSearchCurrentIcao) (>@c:FacilityIcao)
(@c:FacilityLatitude, degrees)
(@c:FacilityLongitude, degrees)
```

No matter if the Ident belongs to an Airport, VOR, NDB, Intersection, or Runway, FacilityICAO can be used to obtain to certain, limited information.

The Facility Group Variables:

- **FacilityICAO (string) [Get, Set]**

The ICAO of the facility.

- **FacilityCode (string) [Get]**

FacilityCode is a single letter string representing Facility type:

- **A** = Airport
- **V** = VOR / ILS / LOC
- **N** = NDB
- **W** = Waypoint / Intersection
- **M** = Marker
- **R** = Runway

Note that no 'M' Marker facilities appear to exist in the fs9gps database.

- **FacilityIdent (string) [Get]**

The one to five letter Ident of the Facility.

- **FacilityValid (bool) [Get]**

FacilityValid is a check of the ICAO passed to FacilityICAO. If the ICAO is a valid fs9gps ICAO, FacilityValid returns 1, otherwise, 0. In the gps_500 gauge, FacilityValid is used to check for a valid ICAO before allowing certain calculations and subsequent gauge display to occur (gps_500 line 3105 for example).

❑ **FacilityName (string) [Get]**

The name of the facility. The International Airport in Jakarta, Indonesia (Ident = WIII) is "Soekarno-Hatta Intl", for example.

❑ **FacilityCity (string) [Get]**

FacilityCity returns the City Name for Airport Facilities. Only Airports have a City Name in fs9gps; FacilityCity for VOR, NDB, and Intersections is a blank string in fs9gps. Runway Waypoints (part of Approach procedures and not included in the WaypointAirport Group) do not have City Names but do have ICAOs with a Region Code.

In North America, FacilityCity returns a string adding State / Province:
City Name, State / Province.

Finally, for some but not all NDBs, a City Name appears in parentheses as part of the FacilityName (e.g., "HI" NDB = ABATE (PORTLAND)).

❑ **FacilityRegion (string) [Get]**

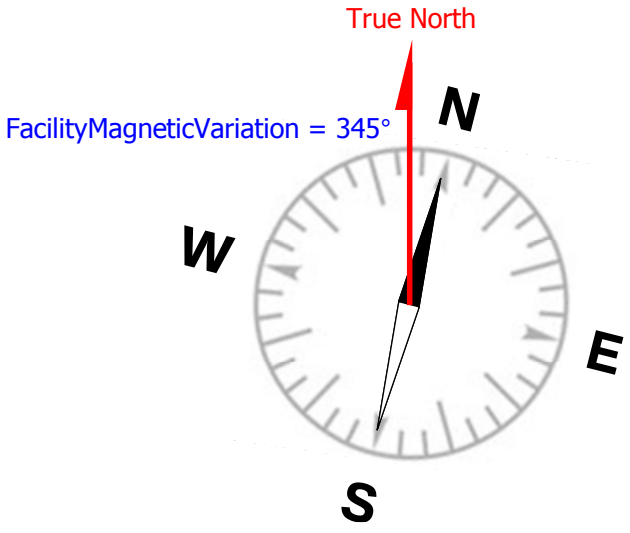
FacilityRegion returns the two letter Region Code. Region Codes don't exist for Airports.

❑ **FacilityLatitude**
❑ **FacilityLongitude (degrees, radians) [Get]**

Latitude and Longitude of the Facility.

❑ **FacilityMagneticVariation (degrees) [Get]**

FacilityMagneticVariation is the compass direction of true north.



GEOCALC GROUP

The GeoCalc Group is fs9gps's spherical geometry calculator, determining distance and bearing from latitude and longitude pairs, or extrapolating latitude and longitude from distance and bearing.

GeoCalc calculations are all performed within the same gauge update cycle. GeoCalc does not extract information from the fs9gps database, so there is no need to add code to skip cycles waiting on GeoCalc results.

- **GeoCalcLatitude1**
- **GeoCalcLongitude1 (degrees or radians) [Get, Set]**

The latitude and longitude of reference point 1. The units of Lat/Lon can be degrees (formatted +/-ddd.dddd where S16 degrees 30 minutes would be written as -16.5000) or radians (d.dddd). Typically, the current aircraft location is set as Latitude1 and Longitude1 within an <Update> section:

```
<Update>
   (A:PLANE LATITUDE, degrees) (>@c:GeoCalcLatitude1, degrees)
   (A:PLANE LONGITUDE, degrees) (>@c:GeoCalcLongitude1, degrees)
</Update>
```

A reminder about Units: GPS variables are sometimes coded without indicating Units, for example, `(C:fs9gps:NearestNdbCurrentLine)`. This usually will not cause difficulty when the Units are either enum or string, as many gps variables are. However, it is quite important to remember Units for gps variables that are not enum or string, such as degrees, feet, knots, nmiles, MHz, etc. variables.

- **GeoCalcLatitude2**
- **GeoCalcLongitude2 (degrees or radians) [Get, Set]**

The latitude and longitude of reference point 2.

- **GeoCalcAzimuth1 (degrees) [Get, Set]**

The bearing (true) from the reference point 1 (GeoCalcLatitude1, GeoCalcLongitude1) to reference point 2 (GeoCalcLatitude2, GeoCalcLongitude2).

- **GeoCalcAzimuth2 (degrees) [Get, Set]**

This variable does not appear to be active in FS9.

❑ GeoCalcLength (nmiles) [Get, Set]

GeoCalcLength is a distance from reference point 1. It is used together with GeoCalcAzimuth1 to calculate ExtrapolationLatitude and ExtrapolationLongitude.

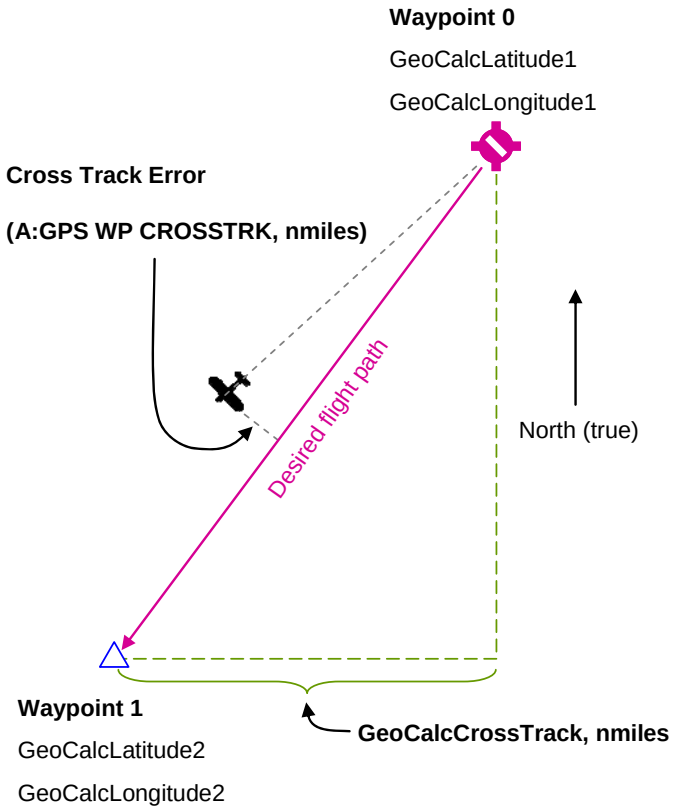
GeoCalcDistance is not the variable to be used for this.

❑ GeoCalcCrossTrack (nmiles) [Get]

GeoCalcCrossTrack returns the East-West base of the triangle set up by two points, GeoCalcLatitude1, GeoCalcLongitude1 and GeoCalcLatitude2, GeoCalcLongitude2 where the sides of the triangle are north-south longitude and east-west latitude lines. GeoCalcCrossTrack is simply the difference in longitude between point 1 and point 2, expressed as a *distance*, and measured along latitude at point 2.

GeoCalcCrossTrack is not the same as standard aeronautical navigation Cross Track Error. Cross Track Error is the distance the aircraft is from the desired flight path track, measured perpendicular to the desired flight path.

Cross Track Error can be accessed using `(A:GPS WP CROSSTRK, nmiles)` where aircraft positions left of the desired flight path are positive Cross Track values and right of the desired flight path are negative.



❑ GeoCalcBearing (degrees) [Get]

GeoCalcBearing is the direction (true) from point 1 defined by GeoCalcLatitude1, GeoCalcLongitude1 to point 2 defined by GeoCalcLatitude2, GeoCalcLongitude2.

❑ GeoCalcDistance (nmiles) [Get]

GeoCalcDistance is the Great Circle distance between two points defined by GeoCalcLatitude1, GeoCalcLongitude1 and GeoCalcLatitude2, GeoCalcLongitude2.

With Latitude and Longitude expressed in radians, the Great Circle Distance Formula formula is:

Distance = (R+Alt)*arccos[sin(Lat1)*sin(Lat2)+cos(Lat1)*cos(Lat2)*cos(Lon2-Lon1)]

Where R is the radius of the Earth in whatever units you desire and Alt is the average aircraft altitude (asl) expressed in the same units. Obviously, Alt does little to change the accuracy of the calculation given its small value compared to Earth radius.

- ❑ R = 3437.74677 (nautical miles)
- ❑ R = 6378.7 (kilometers)
- ❑ R = 3963.0 (statute miles)
- ❑ Lat1, Lat2, Lon1, Lon2 = GeoCalcLatitude1 & 2, GeoCalcLongitude1 & 2 expressed in radians

If Latitude and Longitude values are expressed in degrees, then the degrees-to-radians conversion must be included in the calculation. The Great Circle Distance Formula using decimal degrees becomes:

Distance = (R+Alt)*arccos[sin(Lat1/57.2958)*sin(Lat2/57.2958)+cos(Lat1/57.2958) *cos(Lat2/57.2958)*cos(Lon2/57.2958-Lon1/57.2958)]

Finally, GeoCalcDistance is not slant line distance like DME distance. When flying directly over a point at an elevation 1 nmile above it, GeoCalcDistance is 0.0 and DME Distance is 1.0.

❑ GeoCalcIsIntersect (bool) [Get]

I have not been able to decipher GeoCalcIsIntersect. As far as I can tell, it always returns value=1, and I have not yet been able to cause it to be 0, or any other value. It's not included in the gps_500 xml code, so it is a little difficult for me to figure out, and it may actually not be a working variable.

- **GeoCalcIntersectionLatitude (degrees) [Get]**

Likewise, GeoCalcIntersectionLatitude is a confusing variable. As far as I can tell, it always simply returns GeoCalcLatitude1. With no example in the gps_500 gauge, the GeoCalcIntersection variables remain an enigma. But I note Susan Ashlock's warning that variables not used in the GPS *may not work at all*. If GeoCalcIntersection really is functional, I would certainly like to find out how it is used and what it returns.

GeoCalcIntersectionLongitude

No such variable exists in the gps.dll module. I include it here only for those (like me, initially) who suspect that it may exist because GeoCalcIntersectionLatitude is listed twice in the SDK, leading one to wonder if the second instance is a typo and Longitude intended instead. Anyway, GeoCalcIntersection does not appear to be implemented in fs9gps.

- **GeoCalcExtrapolationLatitude**
- **GeoCalcExtrapolationLongitude (degrees) [Get]**

GeoCalcExtrapolationLatitude and GeoCalcExtrapolationLongitude is the computed Lat & Lon of a point located GeoCalcLength nmiles at GeoCalcAzimuth1 degrees (true) from reference point 1 (GeoCalcLatitude1, GeoCalcLongitude1).

Note that GeoCalcAzimuth2 should not be used. Even when a non-zero degree bearing value is entered into GeoCalcAzimuth2, the resulting GeoCalcExtrapolation Lat and Lon will turn out to be **due north** of point 1, meaning that GeoCalcAzimuth2 is actually zero, regardless of input – in other words, it is not an active variable in FS9, just zero values.

Additionally, the reference point must always be GeoCalcLatitude1 and GeoCalcLongitude1. Using GeoCalcLatitude2 and GeoCalcLongitude2 will not work.

KEYBOARD DIRECT ENTRY

This guide is not an XML code reference, however, in my opinion, Keyboard Direct Entry is a particularly useful fs9gps related XML technique that is not thoroughly covered in the forums and warrants additional discussion here.

User input is sometimes required by gauges that use the fs9gps module. Particularly when ICAO and Name searches are initiated, it may be more convenient using the keyboard to enter strings rather than use of a mouse to click knob or keyboard images in your gauge to produce string entry.

EXAMPLE

The example Keyboard Direct Entry XML code shown below will accept a single character key stroke entry and store it into IcaoSearchStartCursor.

```
1  <Keys>
2     <On Key="AlphaNumeric">
3        <Visible>(L:KeyEntry, enum) 1 ==</Visible>
4           (M:Key) chr (>C:fs9gps:IcaoSearchStartCursor)
5     </On>
6  </Keys>
```

- **Lines 1 and 6:** The Keyboard Direct Entry code must be placed within a `<Keys>` section. `<Keys>` is a stand alone section that should not be placed within `<Element>`, `<Mouse>`, or `<Update>` sections.
- **Line 2:** The choices are `Key="AlphaNumeric"` or `Key="Ascii"`.

  Using **AlphaNumeric**, only lower case alphabet letters, space, and number 0 through 9 keystrokes are accepted. These produce upper case letters, space and numbers 0 through 9. Shift+letter combinations are not accepted. Caps Lock letters are accepted and produce upper case letters. As an example, typing a lower case "**a**" produces ascii decimal value 65, which is actually the ascii value of an upper case "**A**". **AlphaNumeric** is a good choice for ICAOSearchStartCursor because only alphabet letters "**V**", "**A**", "**N**", "**W**", and "**X**" are valid entries for StartCursor. Because Idents contain no special characters, **AlphaNumeric** is also a good choice for IcaoSearchEnterChar.

  Ascii is similar to **AlphaNumeric** except that characters "**+**", "**-**", "**,**", and "**.**" (plus, minus, comma and period) are also accepted. Note that the gps_500 gauge uses **Ascii** for NameSearch entry (see line 3947).

- **Line 3:** This is an important toggle that enables or disables keyboard entry execution according to the M:Key instruction that follows. Only when the <Visible> condition is true will code lines that follow the <Visible> statement be executed.

- The visibility condition (in this example, L:KeyEntry = 1) is usually established by code elsewhere in the gauge, for example, by means of a mouse click that opens a screen page that requires alphanumeric input.
- If the <Visible> statement is omitted, then by default, line 4 will be executed whenever there is a keyboard entry. In this situation, you may lose the use of normal keyboard assignments such as "**G**" for Landing Gear toggle.
- Therefore, you need the ability to turn on and turn off execution of Line 4, limiting its use to the specific situation where you want keyboard entry executed according to the M:Key statement.

❑ **Line 4:** The keyboard direct entry code is executed with each individual keystroke entry. When a keystroke occurs and the <Visible> condition is "true", M:Key generates an ascii decimal number that is mapped to the specific keyboard character. Refer to the Ascii table below:

ASCII TABLE

Character	Ascii Decimal	Character	Ascii Decimal	Character	Ascii Decimal	Character	Ascii Decimal
Space	32	8	56	P	80	h	104
!	33	9	57	Q	81	i	105
"	34	:	58	R	82	j	106
#	35	;	59	S	83	k	107
$	36	<	60	T	84	l	108
%	37	=	61	U	85	m	109
&	38	>	62	V	86	n	110
'	39	?	63	W	87	o	111
(	40	@	64	X	88	p	112
)	41	A	65	Y	89	q	113
*	42	B	66	Z	90	r	114
+	43	C	67	[	91	s	115
,	44	D	68	\	92	t	116
-	45	E	69	]	93	u	117
.	46	F	70	^	94	v	118
/	47	G	71	_	95	w	119
0	48	H	72	`	96	x	120
1	49	I	73	a	97	y	121
2	50	J	74	b	98	z	122
3	51	K	75	c	99	{	123
4	52	L	76	d	100	|	124
5	53	M	77	e	101	}	125
6	54	N	78	f	102	~	126
7	55	O	79	g	103	DEL	127

"**chr**" is an XML string operator that converts the ascii decimal number back to a character or symbol equivalent, which is subsequently stored into the gps string variable IcaoSearchStartCursor.

SHIFT REGISTER

In special situations*, the user may want to concatenate and "store" multi-character alphanumeric keystroke entries. This can be accomplished through the use of a shift register.

The example below accommodates repetitive typing of up to 5 keystrokes, storing the ascii code value of each typed character into a separate L:Var, subsequently retrieving the L:Vars, converting the ascii code back to characters, concatenating the characters, then ultimately storing the string into FlightPlanNewWaypointIdent.

```
1  <Keys>
2     <On Key="AlphaNumeric">
3        <Visible>(L:KeyEntry, enum) 39 ==</Visible>
4           (L:Num394,enum) (>L:Num395,enum)
5           (L:Num393,enum) (>L:Num394,enum)
6           (L:Num392,enum) (>L:Num393,enum)
7           (L:Num391,enum) (>L:Num392,enum)
8           (M:Key) (>L:Num391,enum)
9           (L:Num395,enum) chr
10          (L:Num394,enum) chr scat
11          (L:Num393,enum) chr scat
12          (L:Num392,enum) chr scat
13          (L:Num391,enum) chr scat
14          (>@c:FlightPlanNewWaypointIdent, string)
15    </On>
16 </Keys>
```

- **Lines 1-3, 15-16:** As before.
- **Lines 4-7:** The "shift register". Before converting the current key entry to ascii and storing that number into L:Num391 (Line 8), the ascii value of the *previous* keystroke entry, which was initially stored in L:Num391, is stored into L:Num392 (Line 7). Preceding that (Line 6), the value in L:Num392 is shifted up to (stored into) L:Num393, and so forth. After 5 keystrokes, the ascii value of first keystroke entered will end up being stored in L:Num395.
- **Line 8:** The current keystroke is converted to an ascii decimal value (M:Key) and stored into L:Num391. *Note that each keystroke must be converted to an ascii decimal value before being stored into an L:Var. String data cannot be stored in L:Vars – only numers can be stored in L:Vars.*
- **Lines 9–13:** The L:Var ascii values are recalled in a last-in, first-out order, converted back to symbols using the '**chr**' operator, and concatenated using the '**scat**' operator to form a string.
- **Line 14:** The concatenated string is entered into FlightPlanNewWaypointIdent.
- Before using a shift register to store keyboard entry ascii values, you will need to first "clear" all of the old L:Var values by storing a zero (which is the ascii null value) into each of the L:Vars.

The following are the cumulative results of typing "**a**", "**b**", "**space**", "**Shift+c**", "**d**", "**.**", "**5**", "**f**":

Entry #	Keystroke	L:KeyEntry	Num391	Num392	Num393	Num394	Num395	Concatenated string
1	a	39	65	0	0	0	0	A
2	b	39	66	65	0	0	0	AB
3	space	39	32	66	65	0	0	AB then "space"
4	Shift+c	39	32	66	65	0	0	AB then "space"
5	d	39	68	32	66	65	0	AB D
6	.	39	68	32	66	65	0	AB D
7	5	39	53	68	32	66	65	AB D5
8	f	39	70	53	68	32	66	B D5F

- **Keyboard Entry #4:** Shift+c is ignored by M:Key and does not produce an ascii value. The existing Shift+c keyboard assignment, if any, will be performed as usual.
- **Keyboard Entry #6:** With <On Key="**AlphaNumeric**">, periods are ignored. The existing keyboard assignment for period (usually, Brake Release) will be performed as usual.

To display what was just typed using an <Element>:

```
<Element Name="Entry Box 39: FlightPlanNewWaypointIdent">
  <Position X="775" Y="86"/>
     <FormattedText X="101" Y="20" Adjust="left"
        Font="Courier New" Color="#111111" FontSize="10"
        LineSpacing="16" Bright="Yes">
        <String>
           %(
           (L:Num395,enum) chr
           (L:Num394,enum) chr scat
           (L:Num393,enum) chr scat
           (L:Num392,enum) chr scat
           (L:Num391,enum) chr scat
           )%!s!
        </String>
     </FormattedText>
</Element>
```

* Special situations that would be beyond the basic requirements of the gps_500 gauge. The fs9gps module automatically concatenates Alphanumeric and Ascii keyboard entries that are required for IcaoSearch and NameSearch. The gps_500 gauge does not use shift registers. As well, special @g functions such as @g:enteringInput are not required for concatenation of keyboard entry in use of the gps_500 gauge.

<ELEMENT> DISPLAY LOOPS

The <Element> display loop is an xml "must-know" for working with the gps module. It is covered in the gps_500 xml gauge, in the forums, a few places in this guidebook, and also covered again in this section.

The following script produces a list of the Nearest VORs extracted from the fs9gps database in a NearestVor search.

```
<Update Frequency="18" Hidden="No">
    20 (>C:fs9gps:NearestVorMaximumItems, enum)
    100 (>C:fs9gps:NearestVorMaximumDistance, nmiles)
    62 (>C:fs9gps:NearestVorCurrentFilter)
    (A:PLANE LATITUDE, degrees) (>C:fs9gps:NearestVorCurrentLatitude, degrees)
    (A:PLANE LONGITUDE, degrees) (>C:fs9gps:NearestVorCurrentLongitude, degrees)
</Update>

<Element Name="NEAREST VOR LOOP DISPLAY">
    <Position X="10" Y="25"/>
    <FormattedText X="800" Y="800" Font="Courier New" FontSize="12"
    LineSpacing="12" Color="#101010" Bright="Yes" >
    <Color Value="blue"/>
    <Color Value="darkgreen"/>
        <String>
        \{clr2}NEAREST VOR SEARCH\n\n\{clr3}
        %((C:fs9gps:NearestVorCurrentLatitude, degrees))%!9.4f! :Current Lat
        %((C:fs9gps:NearestVorMaximumItems, enum))%!5d! :Max Items
        %((@c:NearestVorItemsNumber))%!4d! :Items Num\n
        %((C:fs9gps:NearestVorCurrentLongitude, degrees))%!9.4f! :Current Lon
        %((C:fs9gps:NearestVorMaximumDistance, nmiles))%!5d! :Max Dist
        %((@c:NearestVorCurrentFilter))%!5d! :Filter
        \n\n\{clr2}
        ----------------- NearestVorCurrent -----------------\n
       %      ICAO     111\n
        Line  123456789012  Ident Type     Freq    Dist   Brg\n
            %((@c:NearestVorItemsNumber) s2 0 !=)
                %{if}
                    %(0 sp1)
                    %{loop}
                        %(l1 (>@c:NearestVorCurrentLine))
                            \{clr}%((@c:NearestVorCurrentLine))%!-5d!
                            %((@c:NearestVorCurrentICAO))%!13s!
                            %((@c:NearestVorCurrentIdent))%!7s!
                            %((@c:NearestVorCurrentType))%!5d!
                            %((@c:NearestVorCurrentFrequency, mhz))%!9.2f!
                            %((@c:NearestVorCurrentDistance, nmiles))%!8.1f!
                            %((@c:NearestVorCurrentTrueBearing, degrees))%!6d!\n
                            %(l1 ++ s1 l2 &lt;)
                    %{next}
                %{end}
        </String>
    </FormattedText>
</Element>
```

```
NEAREST VOR SEARCH

  38.0739 :Current Lat   20 :Max Items  10 :Items Num
 -97.8707 :Current Lon  100 :Max Dist   62 :Filter

----------------- NearestVorCurrent -----------------
      ICAO     111
Line  123456789012  Ident Type     Freq     Dist   Brg
0     VK3     HUT      HUT    2   116.80      5.5   213
1     VK3     ICT      ICT    2   113.80     23.9   145
2     VK3     IAB      IAB    3   116.50     39.7   133
3     VK3     SLN      SLN    2   117.10     52.4    13
4     VK3     ANY      ANY    2   112.90     56.7   195
5     VK3     FRI      FRI    1   109.40     71.7    41
6     VK3     HYS      HYS    2   110.40     80.8   306
7     VK3     EMP      EMP    2   112.80     82.8    80
8     VK3     MHK      MHK    2   110.20     85.5    41
9     VK4     PER      PER    2   113.20     86.5   157
```

Line:

- ❑ 25 – 35 The header lines for the Nearest VOR display list.
- ❑ 36 Cycle skipping/delay command. A Nearest search always consumes multiple gauge update cycles and display of search results cannot begin until values have been returned. Line 36 delays displaying of the NearestVor list until the Nearest search has returned values as evidenced by NearestVorItemsNumber being greater than zero. This number is then stored into Register #2 which is checked each loop (Line 48) to see if all VORs have been displayed.
- ❑ 37 Condition statement. Used in connection with the cycle skipping command.
- ❑ 38 The value zero is stored into Register #1. "0" is always the value of the first index line.
- ❑ 39 The display loop begins. Variables for an individual VOR are displayed one VOR at a time / one line at a time based on the current Index pointer, the value in Register #1.
- ❑ 40 Register #1 is loaded into the NearestVor index pointer variable.
- ❑ 41 – 47 The NearestVor variables associated with the current Index pointer that will be displayed all on the same line.
- ❑ 48 The "incrementer". After each VOR variable list is displayed, Register #1 is incremented by 1 and Register #2 is checked to see if all of the VOR's have been displayed.

BUGS, INOPS, and ISSUES

Not that there is anyone at MSFT that will do anything about these after ACES demise, but here is a list of Bugs / Inops / Issues I have run across.

I recognize that some, perhaps many, have been identified long before now...

Group	Variable	Bug
FlightPlan	FlightPlanIsActiveWaypoint	It is not Settable
FlightPlan	FlightPlanIsDirectTo	It is not Settable
FlightPlan	FlightPlanActiveWaypoint	It is Settable
FlightPlan	FlightPlanNewApproachAddInitialLeg	MSFT ESP web page *suggests* this may be Inop (units Unavailable), but it is operational
FlightPlan	FlightPlanWaypointFrequency	Value returned is not a valid VOR frequency. Already noted by MSFT ACES
FlightPlan	FlightPlanWaypointMinAltitude	Feet units - must specify 'meters' to get feet
FlightPlan	FlightPlanApproachSegmentLength	At least in the case of the KICT ILS19R Initial Approach segment, the sub-segment outbound from the IAF is too long, causing the Procedure Turn to exceed the allowable distance.
FlightPlan	FlightPlanWaypointMinAltitude	Database issue - some segments have an obviously incorrect (too low) MEA, such as 0
FlightPlan	FlightPlanWaypointApproachCourse	Some bearings are True but all should be Magnetic
GeoCalc	GeoCalcAzimuth2	Not active in fs9gps
GeoCalc	GeoCalcCrossTrack	Maybe not a bug as much as it is quite misleading
GeoCalc	GeoCalcIsIntersect	Appears Inop
GeoCalc	GeoCalcIntersectionLatitude	Appears Inop
NearestAirspace	NearestAirspaceCurrentNearDistance	Incorrectly identifies distance with the far, 'occluded' airspaces
WaypointAirport	WaypointAirportRegion	Unnecessary variable. Regions do not exist for Airports
WaypointAirport	WaypointAirportRadarCoverage	Inop
WaypointAirport	WaypointAirportAirspace	Inop
NearestNDB	NearestNdbCurrentFilter	does not exist
Nearest__	Nearest__MaximumDistance	Especially in large searches, MaximumDistance is not strictly adhered to. For some reason apparently having to do with the way the search algorithm works, some searches return items 20 to 25% more distant than MaximumDistance.
WaypointNDB	WaypointNdbCity	Always returns a blank string
WaypointNDB	WaypointNdbWeatherBroadcast	Inop
WaypointVOR	WaypointVorWeatherBroadcast	Inop
WaypointVOR	WaypointVorCity	Always returns a blank string
WaypointIntersection	WaypointIntersectionCity	Always returns a blank string
WaypointIntersection	NearestVOR	Not the nearest VOR

fs9gps GUIDEBOOK UPDATES

Version 1.1

Page	Edit
2	Removed RXP reference
11	Revised Table Title
12	**Fixed wrong ICAO example**
14	Highlighted the multiple update cycle database operations
15	Fixed spill over text
31	**Corrected M:Key xml code**
42	Corrected grammar mistake
54	Removed underline
79	**Corrected statement about A:PLANE and A:GPS update frequency**
81	Corrected MSFT online SDK reference url
82	Corrected reference to the Garmin GNS 500
91	Blue text for a gps var
94	**Corrected M:Key xml code**
104	**Corrected M:Key xml code**
118	Updated xml I/O remark
121	Removed errant tab
157	Changed graphic
169	Added Flight Plan New Approach Table