FS9GPS MODULE

an empirical xml guide v.1.1

BOB McELRATH

JUNE, 2011

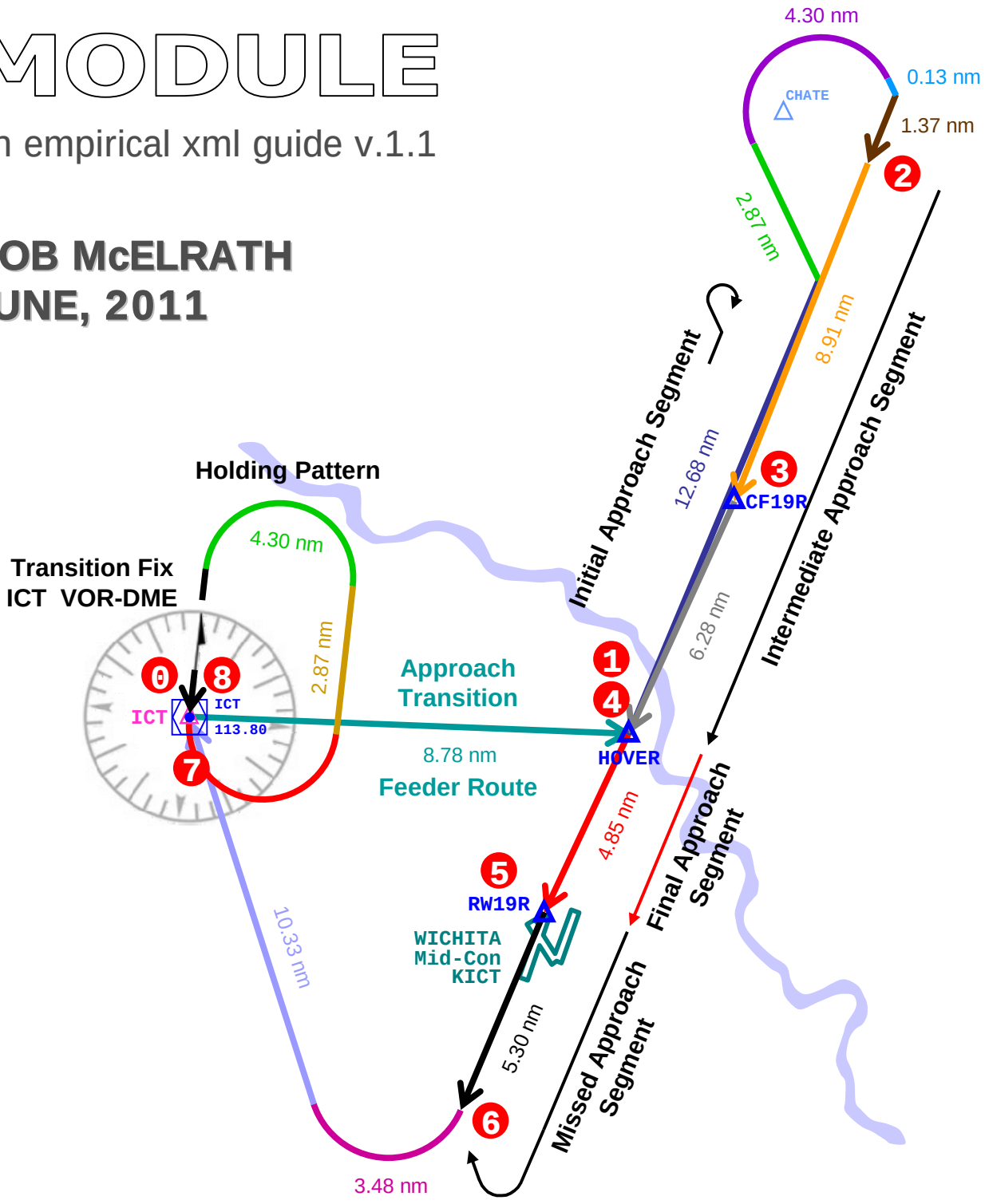


TABLE OF CONTENTS

INTRODUCTION

This is an empirical guide for working with the FS9 gps module. It's a collection of 'notes to myself' on utilization of the fs9gps module, definition and examples of its 320 variables, and discussion of some xml coding techniques that are used when working with the fs9gps. Started out as the GPSViewer spreadsheet then grew into a book.

What it's not - This isn't a discussion of the stock gps_500 xml gauge. None of the 80+ custom @g functions found in that gauge are discussed (however, *none* of those are necessary to fully use all of the capabilities of the gps module). Nor is this a pilot's handbook on the operation of a gps unit. It doesn't directly address the FSX gps module, although in my opinion, if you know FS9 gps, you will have little difficulty with FSX which seems easier to work with (less need for ICAO transfers, for example). Finally, it does not explain, per se, how to write sophisticated gauges such as the gps_500 or the Garmin 1000 or autopilots that use the gps module.

I have learned much from forum posts by several gps veterans such as Jan Van Harten, Paul (PVE), and especially, Tom Aguilo, Roman Stoviak, and Ed Wilson. Luckily, our community has some professional and experienced programmers that have always been willing to offer advice. Bill Leaming heads that list. Still, a significant number of things having to do with the fs9gps are not addressed by MSFT or in the forums, and that which is out there is scattered around … and vanishing sometimes, so I have attempted to fill in the 'white space' and pull it together into one document.

A warning is that I am a layman, neither a long time FS gauge hobbyist or a professional programmer. Frankly, I'm not an especially strong programmer for that matter and I realize there may be alternative approaches to those I suggest. Also, the pros will find some mistakes in here relating to nomenclature, syntax, or basic understanding of the way Flight Simulator and the gps.dll module really work. When you do, I'd appreciate receiving an email or a forum response so I can correct the errors.

I have relied a lot on some basic tools such as BlackBox2, GPSViewer, and many fit for purpose xml scripts to study the fs9gps … test, test, test, and re-test. I mention the tools to acknowledge the contribution of my son, Robbie McElrath – the author of BlackBox (and the professional programmer of the family).

Bob McElrath
Jakarta, Indonesia
March, 2011

PRELIMINARIES

Some odds and ends:

- FS9 Version 9.01, Build 40901.01 was used in development of this guidebook.
- Stock FS9 gps database. Not updated with third party navaid, terrain mesh, or magnetic declination files.
- I indicate gps variables using blue font, NearestNdbCurrentIdent, and xml code typically in green Courier New font **`(A:PLANE LONGITUDE, degrees)`**.
- Very often, I abbreviate gps variable names by omitting the Group Name. For example, in the Flight Plan Data Group, I refer to FlightPlanWaypointAproachTarget simply as WaypointApproachTarget.
- I use FS9 xml syntax throughout.
- I do not get into the Airport Design Editor world when discussing Approaches. I discuss only the stock fs9gps variables and approach segment definitions.
- The aircraft used for sim testing was predominantly Flight 1's Cessna C421 twin. The aircraft configuration remains unchanged from the default settings except for correction of the max_indicated_speed and cruise_speed reversal. Flight simulation testing was performed with wind speed set to zero and gyro drift disabled.
- An Autopilot was used on all flight testing. FS9's stock Bendix-King Radio AP.
- Aviation nomenclature used is predominantly USA standard from Federal Aviation Administration resources (Aeronautical Information Manual, Instrument Flying Handbook, Instrument Procedures Handbook, in particular) http://www.faa.gov/library/manuals/aviation/, or Microsoft's FS9 Help section.
- English units were used in preparation of this guidebook.
- Flight Sim tools and fs9gps resources used: BlackBox2, GPSViewer1.2, numerous specific use xml gauge scripts, and the gauge and panel sections of several flight sim forums (AVSIM, FS Developer, Freeflight Design, FlightSim, Simviation).

GET, SET, and UNITS

GET AND SET

Fs9gps variables are classified as follows:

- **Get.** Read-only. Strings and numbers associated with these variable types can be displayed in <Elements>.

- **Set.** Write-only. These are a little more difficult to work with because you cannot directly display the values entered into these variables in display Elements. The use of a 'shadowing' L:Var is sometimes necessary in order to view what value was Set. As an example, I use the following to view confirmation of what was actually *entered* into the Set-only variable, FlightPlanLoadApproach:

  ```
  (L:LoadApprEnum, enum) d (>c:fs9gps:FlightPlanLoadApproach)
  (>L:LoadApprEnumEntered, enum)
  ```

  and then display `(L:LoadApprEnumEntered, enum)` in an <Element>.

  I have LoadApprEnum to begin with, but if I want to confirm what was really *entered*, I need LoadApprEnumEntered because FlightPlanLoadApproach cannot be displayed.

- **Get and Set.** Read and Write capable.

UNITS

Like most other FS9 variables (e.g., A:, L:, P:, E:), gps variables have units. Throughout this guidebook, however, I omit specifying the units for string, enum and bool variables, but include units for distance, direction, time, speed, and frequency-type gps variables.

Examples:

- (c:fs9gps:NearestVorCurrentFilter) - enum variable
- (c:fs9gps:NearestVorCurrentDistance, NMiles) – distance variable
- (c:fs9gps:NearestVorCurrentFrequency, MHz) – frequency variable
- (c:fs9gps:NearestVorCurrentLongitude, degrees) – direction variable
- (c:fs9gps:NearestVorCurrentICAO) – string variable

fs9gps VARIABLE GROUPS

There are 320 gps variables in the fs9gps gps module that are organized into the following functional data groups:

- ❑ Waypoint Data Groups:
  - WaypointAirport Group
  - WaypointIntersection Group
  - WaypointNdb Group
  - WaypointVor Group
- ❑ Search Data Groups:
  - Nearest Search Groups:
    - o NearestAirport Group
    - o NearestIntersection Group
    - o NearestVor Group
    - o NearestNdb Group
    - o NearestAirspace Group
  - ICAOSearch Group
  - NameSearch Group
- ❑ Message Data Group
- ❑ FlightPlan Data Group
- ❑ Facility Data Group
- ❑ GeoCalc Data Group

The FSX gps module uses the same functional groups but contains additional variables. Many of the new variables simply augment the Nearest Groups with variables from the Waypoint Groups making Nearest searches more robust and reducing the need for ICAO Transfer to access some Waypoint Group information.

FS REGIONS

The FS9 world is divided into approximately 285 Regions following **I**nternational **C**ivil **A**viation **O**rganization format. The first letter of the two letter Region code is generally determined by continent. The second letter generally represents a country within that continent. The exception is some larger countries that have single-letter country codes such as the USA which uses country code "K".

The Region code is part of the fs9gps ICAO Identifier, in character positions 2 and 3.

International **C**ivil **A**viation **O**rganization Region Code

http://en.wikipedia.org/wiki/International_Civil_Aviation_Organization_airport_code

A - Western South Pacific

AG	Solomon Islands
AN	Nauru
AY	Papua New Guinea

B - Iceland/Greenland and Kosovo

BG	Greenland
BI	Iceland
BK	Kosovo

C - Canada

C	Canada (Multiple)

D - West Africa

DA	Algeria

N - South Pacific

NC	Cook Islands
NF	Fiji, Tonga
NG	Kiribati (Gilbert Islands), Tuvalu
NI	Niue
NL	Wallis and Futuna
NS	Samoa, American Samoa
NT	French Polynesia
NV	Vanuatu
NW	New Caledonia
NZ	New Zealand, Antarctica

O - Southwest Asia (excluding Israel and Turkey), Afghanistan and Pakistan

DB Benin
DF Burkina Faso
DG Ghana
DI Côte d'Ivoire
DN Nigeria
DR Niger
DT Tunisia
DX Togolese Republic

E - Northern Europe

EB Belgium
ED Germany (civil)
EE Estonia
EF Finland
EG United Kingdom
EH Netherlands
EI Ireland
EK Denmark
EL Luxembourg
EN Norway
EP Poland
ES Sweden
ET Germany (military)
EV Latvia
EY Lithuania

F - Southern Africa

FA South Africa
FB Botswana
FC Republic of the Congo
FD Swaziland
FE Central African Republic
FG Equatorial Guinea
FH Ascension Island
FI Mauritius
FJ British Indian Ocean Territory
FK Cameroon
FL Zambia
FM Comoros, Madagascar, Mayotte, Réunion
FN Angola
FO Gabon
FP São Tomé and Príncipe
FQ Mozambique
FS Seychelles
FT Chad
FV Zimbabwe
FW Malawi
FX Lesotho
FY Namibia
FZ Democratic Republic of the Congo

G - Northwestern Africa

OA Afghanistan
OB Bahrain
OE Saudi Arabia
OI Iran
OJ Jordan and the West Bank
OK Kuwait
OL Lebanon
OM United Arab Emirates
OO Oman
OP Pakistan
OR Iraq
OS Syria
OT Qatar
OY Yemen

P - Eastern North Pacific

PA Alaska only
PB Baker Island
PC Kiribati (Canton Airfield, Phoenix Islands)
PF Fort Yukon, Alaska
PG Guam, Northern Marianas
PH Hawaii only
PJ Johnston Atoll
PK Marshall Islands
PL Kiribati (Line Islands)
PM Midway Island
PO Oliktok Point, Alaska
PP Point Lay, Alaska
PT Federated States of Micronesia, Palau
PW Wake Island

R - Western North Pacific

RC Republic of China (Taiwan)
RJ Japan (most of country)
RK South Korea
RO Japan (Okinawa Prefecture and Yoron)
RP Philippines

S - South America

SA Argentina
SB Brazil (also SD, SI, SJ, SN, SS and SW)
SC Chile
SD Brazil (also SB, SI, SJ, SN, SS and SW)
SE Ecuador
SF Falkland Islands
SG Paraguay
SI Brazil (also SB, SD, SJ, SN, SS and SW)
SJ Brazil (also SB, SD, SI, SN, SS and SW)
SK Colombia
SL Bolivia
SM Suriname
SN Brazil (also SB, SD, SI, SJ, SS and SW)

GA Mali
GB The Gambia
GC Canary Islands (Spain)
GE Ceuta and Melilla (Spain)
GF Sierra Leone
GG Guinea-Bissau
GL Liberia
GM Morocco
GO Senegal
GQ Mauritania
GS Western Sahara
GU Guinea
GV Cape Verde

H - Northeastern Africa

HA Ethiopia
HB Burundi
HC Somalia (including Somaliland because of disputes)
HD Djibouti (also HF)
HE Egypt
HF Djibouti (also HD)
HH Eritrea
HK Kenya
HL Libya
HR Rwanda
HS Sudan
HT Tanzania
HU Uganda

K - United States (excluding Alaska and Hawaii)

K Contiguous United States (K1, K2, K3 …K7)

L - Southern Europe, Israel and Turkey

LA Albania
LB Bulgaria
LC Cyprus
LD Croatia
LE Spain
LF France, including Saint-Pierre and Miquelon
LG Greece
LH Hungary
LI Italy
LJ Slovenia
LK Czech Republic
SO French Guiana
SP Peru
SS Brazil (also SB, SD, SI, SJ, SN and SW)
SU Uruguay
SV Venezuela
SW Brazil (also SB, SD, SI, SJ, SN and SS)
SY Guyana

T - Caribbean

TA Antigua and Barbuda
TB Barbados
TD Dominica
TF Guadeloupe
TG Grenada
TI U.S. Virgin Islands
TJ Puerto Rico
TK Saint Kitts and Nevis
TL Saint Lucia
TN Netherlands Antilles, Aruba
TQ Anguilla
TR Montserrat
TT Trinidad and Tobago
TU British Virgin Islands
TV Saint Vincent and the Grenadines
TX Bermuda

U - Russia and former Soviet States

U Russia (except UA, UB, UD, UG, UK, UM and UT)
UA Kazakhstan, Kyrgyzstan
UB Azerbaijan
UD Armenia
UG Georgia
UK Ukraine
UM Belarus and Kaliningrad, Russia
UT Tajikistan, Turkmenistan, Uzbekistan

V - South Asia (except Afghanistan and Pakistan), mainland Southeast Asia, Hong Kong and Macau

VA India (West Zone, Mumbai Center)
VC Sri Lanka
VD Cambodia
VE India (East Zone, Kolkata Center)
VG Bangladesh
VH Hong Kong
VI India (North Zone, Delhi Center)

LL Israel
LM Malta
LN Monaco
LO Austria
LP Portugal, including the Azores
LQ Bosnia and Herzegovina
LR Romania
LS Switzerland
LT Turkey
LU Moldova
LV Areas Under the Control of the Palestinian Authority
LW Macedonia
LX Gibraltar
LY Serbia and Montenegro
LZ Slovakia

M - Central America and Mexico

MB Turks and Caicos Islands
MD Dominican Republic
MG Guatemala
MH Honduras
MK Jamaica
MM Mexico
MN Nicaragua
MP Panama
MR Costa Rica
MS El Salvador
MT Haiti
MU Cuba
MW Cayman Islands
MY Bahamas
MZ Belize
VL Laos
VM Macau
VN Nepal
VO India (South Zone, Chennai Center)
VQ Bhutan
VR Maldives
VT Thailand
VV Vietnam
VY Myanmar

W - Maritime Southeast Asia (except the Philippines)

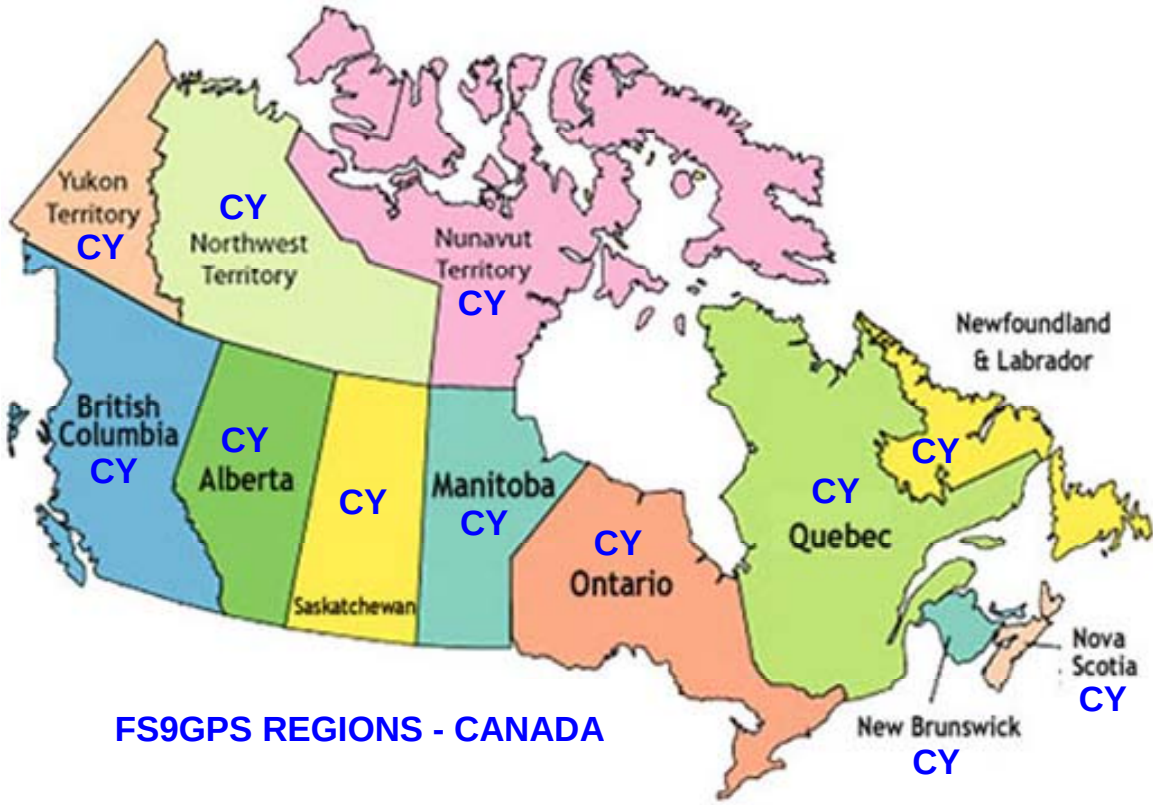
WA Indonesia (also WI, WQ and WR)
WB Malaysia (East Malaysia), Brunei
WI Indonesia (also WA, WQ and WR)
WM Malaysia (Peninsular Malaysia)
WP Timor-Leste
WQ Indonesia (also WA, WI and WR)
WR Indonesia (also WA, WI and WQ)
WS Singapore

Y - Australia

Y Australia

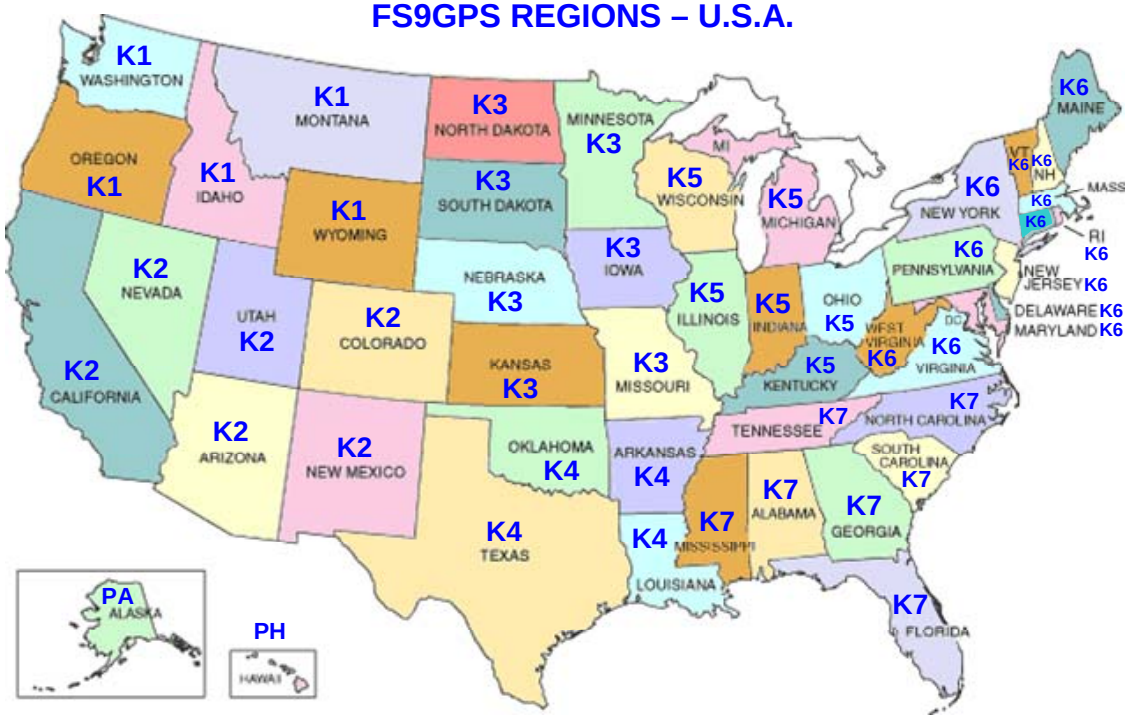
Z - East Asia (excluding Hong Kong, Japan, Macau, South Korea and Taiwan)

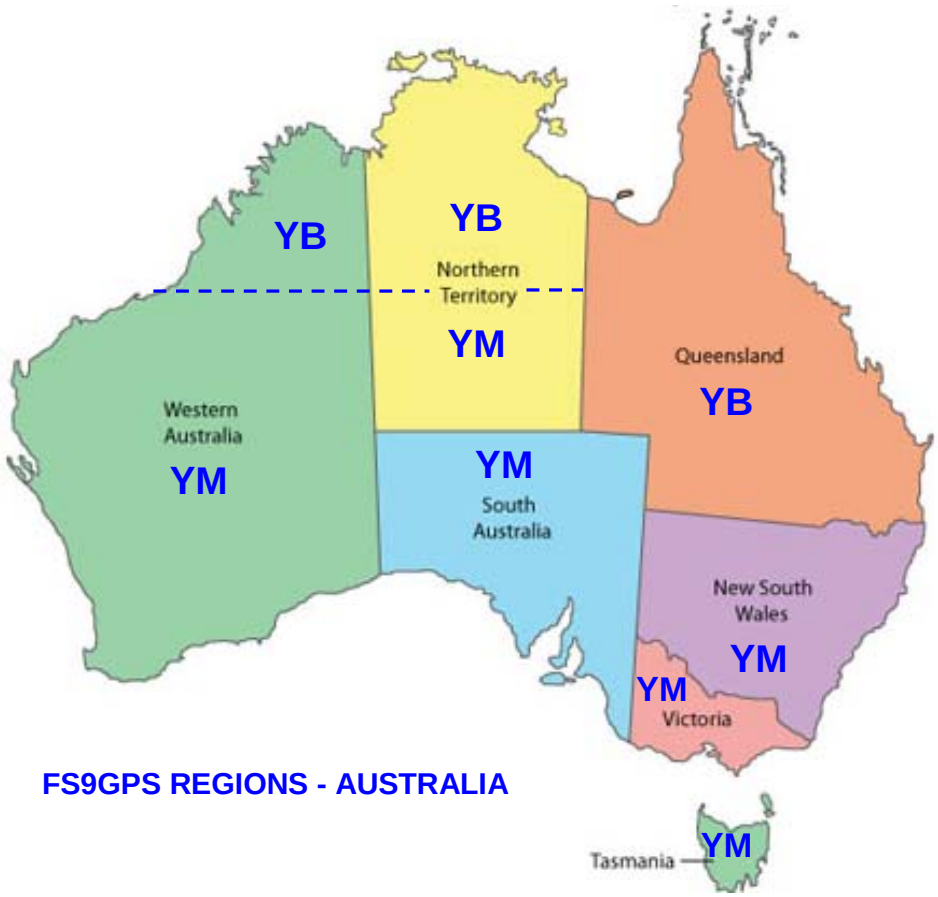
Z People's Republic of China (except ZK and ZM)
ZK North Korea
ZM Mongolia



FS9GPS REGIONS - CANADA

FS9GPS REGIONS – U.S.A.





FS9GPS REGIONS - AUSTRALIA

ICAO

The ICAO is a 12 character* long unique identification string for all facilities in the fs9gps database. It is required for access to almost all of the variables within the Waypoint Groups, where most of the fs9gps database information is located. The exception is that latitude, longitude and Ident of the facility the ICAO represents are accessible without the need to retrieve Lat and Lon from the specific Waypoint Group.

All single point facilities (Airport, VOR, NDB, Intersection) in the database have a unique ICAO. Airspaces are not single point facilities and consequently do not have an ICAO.

ICAO Format

fs9gps ICAO - Examples

		Character position									1	1	1
FACILITY	SLEN	1	2	3	4	5	6	7	8	9	0	1	2
Airport	12	A							E	G	N	X	
Airport	12	A							W	3	6		
Airport ILS	12	V			K	A	S	T	I	A	S	T	
VOR	12	V	E	G					M	I	D		
Intersection	12	W	K	1					E	G	R	E	T
Waypoint	12	W	V	T	V	T	B	D	C	F	0	3	R
NDB	12	N	P	A					C	M	Q		
NDB	12	N	E	B	E	B	B	R	O	Z			
		Type	Region		"Owning" Airport Ident				Facility Ident				

The ICAO is assembled from four parts that are concatenated to form the 12 character ICAO identifier. The four parts are:

- **Type**. Character position 1. A single letter representing the type of facility.
  - "A" = Airport
  - "V" = VOR, ILS, LOC
  - "N" = NDB
  - "W" = Waypoint, Intersection
  - "R" = Runway
  - "X" = NDB
- **Region**. Character positions 2 and 3. The two letter FS Region code. Note that the Airport Group, which includes ILS and LOC, does not include Region in the ICAO (which is why WaypointAirportRegion always returns a blank string).

- **Owning Airport Ident**. Character positions 4 through 7. For navigation facilities (ILS, NDB) and points (Waypoints, Intersections) that are part of an approach procedure in the fs9gps database, the Ident of the airport to which the procedure belongs is indicated. Without Owning Airport Ident, the ICAO would not be unique for Computer Navigation Fix and unnamed waypoints. As an example, 'CF19R' is the Ident of a computer fix waypoint which may be part of an approach at multiple airports having a runway 19R. Without including the associated, or owning, airport, the ICAO "WK3____CF19R" would probably not be unique. However, adding the owning airport Ident makes it unique, "WK3KICTCF19R".

- **Ident**. Character positions 8 through 12. The 1 to 5 character long Ident of the facility. Note that while ICAOs are unique, Idents are not necessarily unique other than airport Idents. For example, there are many occurrences in the database of VORs having the same 3 letter Ident.

***** The String Length (SLEN) of an ICAO is always 12, even in cases where the Ident is not 5 characters long. However, when directly entering an ICAO, such as:

```
'A      W36' (>C:fs9gps:WaypointAirportICAO)
```

it is acceptable to omit the trailing spaces.

GPS DATABASE SEARCH: Search> Index> Display

Most information needed from the fs9gps database must be obtained through a sequence of 1) Search, 2) Index, and then 3) Display. It is important to understand that the Search step can be computationally extensive. Database information cannot be manipulated or displayed until it is retrieved from the database. More than one gauge update cycle is often required to *retrieve* data from the database. Consequently, the gps module has been described by some as having asynchronous behavior – data cannot be retrieved from the database and then displayed or otherwise utilized in the same gauge update cycle.

SEARCH (and EXTRACT)

Search is the action of requesting and receiving a small (tiny) portion of the global data base for subsequent manipulation by the gauge – manipulation that can be as simple as display of the information.

In general, the first step is to define 1) the type of information desired, 2) the geographic location of interest, and 3) the amount of information (the limit) the user wants to extract from the gps data base. For example, if a list of VORs nearest the aircraft is needed, a NearestVor search can be initiated using the following statements:

```
(A:PLANE LATITUDE, degrees)
(>c:fs9gps:NearestVorCurrentLatitude, degrees)

(A:PLANE LONGITUDE, degrees)
(>c:fs9gps:NearestVorCurrentLongitude, degrees)

20 (>c:fs9gps:NearestVorMaximumItems, enum)
100 (>c:fs9gps:NearestVorMaximumDistance, nmiles)
```

From these instructions, the gps engine knows the type of information requested (nearest VORs), the geographic location of interest (lat and lon of the aircraft) and the maximum amount of information desired (max items and max search distance, or radius).

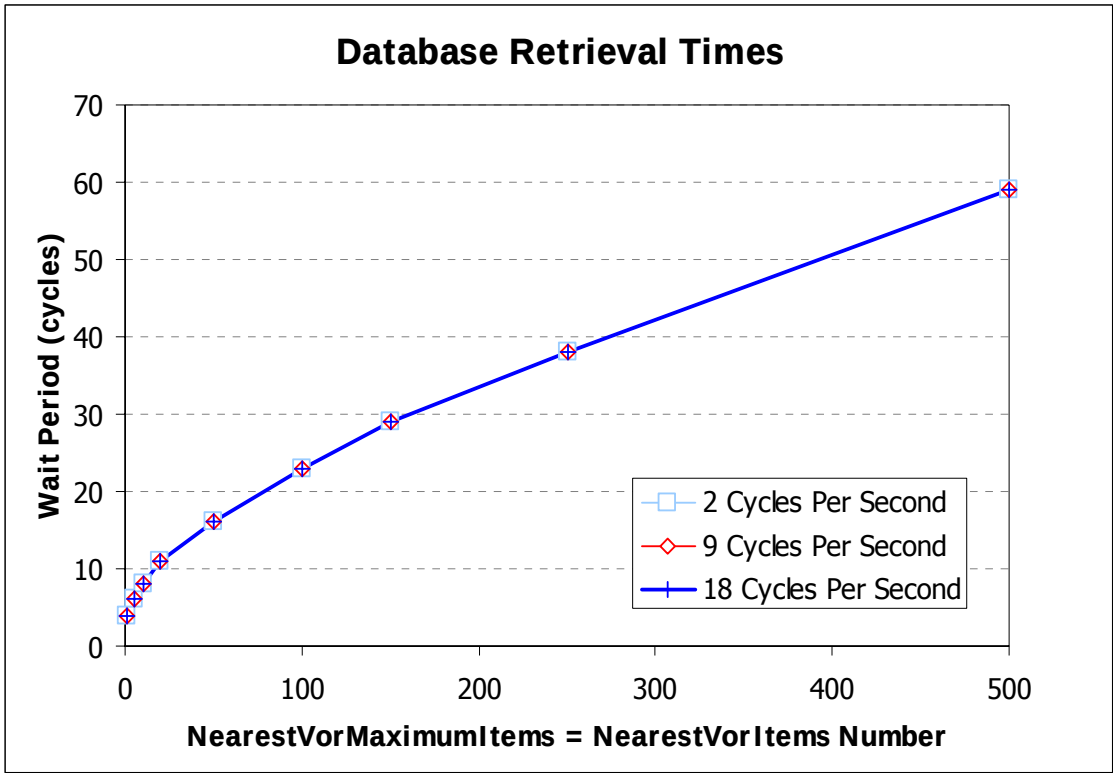
As soon as these statements are executed, a gps database search automatically begins. Some amount of time, however small, is required to extract the data, so you must wait for database search results to be delivered. How many gauge update cycles you wait is predominantly a function of the amount information requested: max distance and max items. In the stock gps_500 xml gauge, max items and max distance values are set realistically low (e.g., lines 416-419, max items = 9) and it seems that data are sometimes displayed *almost* instantly. However, if one wants a list of nearest airports within 1000 nmiles of Chicago O'Hare airport with a max item limit of 10,000, be prepared to wait. That order will take a minute or so to prepare. Regardless of the gauge update cycle rate, several, and in unrealistically large searches, thousands of update cycles may pass before search and extract is complete.

The figure below shows the gauge update cycle wait plotted against MaximumItems for an arbitrary NearestVor search. Three things are noteworthy:

- ❑ First, retrieval of just <u>one</u> item in this particular NearestVor search required 4 gauge update cycles before the retrieved data could be accessed and stored as an L:Var, or even just displayed on the screen.

- ❑ Second, the same number of gauge update cycles are required for the search regardless of gauge update cycle frequency. So, the key is to wait on update cycles, not absolute elapsed time, for data retrieval.

- ❑ Third, the bigger the search, the longer the wait.



The **multiple update cycle** database operations include:

1) **Nearest searches**, and

2) **ICAO transfer**

Does this matter? Often, no. Waiting on data extraction may not harm the function of the gauge you are building. This is especially true if searches are simple and retrieved data are only displayed on the screen, which describes most searches used in the gps_500 gauge. For example, a display loop within an <Element> could, in effect, just

sit there displaying blank lines, waiting for search data to become available, and it does not matter how few or how many update cycles pass before that happens. Often, the delay is short, almost imperceptible, and of no consequence.

However, there are situations where the user must wait until data have been retrieved from the database before executing subsequent code. These situations, as well as knowing how many gauge update cycles to wait, are the topics of the **Asynchronous Operation** section.

INDEX

Much of the information retrieved from the gps database is structured, that is, organized in the form of lists: the list of nearest airports or VORs, the list of radio frequencies or runways at an airport, the list of waypoints in a Flight Plan. In a list of nearest airports, for example, all retrieved data from a specific, individual airport can be thought of as occupying one line of the list. The lines are numbered, or indexed, and to display or access data from any particular line, the line number must first be specified. This is accomplished by assigning a number to an index pointer such as the CurrentLine or Index variable.

```
2 (>c:fs9gps:NearestVorCurrentLine, enum)
```

selects the third VOR of a NearestVor list (indices and line numbers start at 0 for the first line).

The CurrentLine / Index pointers and Total Number variables in fs9gps include:

Index Pointer (all are enum)	**Total Number** (all are enum)
WaypointAirportCurrentFrequency	WaypointAirportFrequenciesNumber
WaypointAirportCurrentRunway	WaypointAirportRunwaysNumber
WaypointAirportCurrentApproach	WaypointAirportApproachesNumber
WaypointAirportApproachCurrentTransition	WaypointAirportApproachTransitionsNumber
NearestAirportCurrentLine	NearestAirportItemsNumber
NearestIntersectionCurrentLine	NearestIntersectionItemsNumber
NearestVorCurrentLine	NearestVorItemsNumber
NearestNdbCurrentLine	NearestNdbItemsNumber
NearestAirspaceCurrentLine	NearestAirspaceItemsNumber
FlightPlanWaypointIndex	FlightPlanWaypointsNumber
IcaoSearchMatchedIcao	IcaoSearchMatchedIcaosNumber
MessageCurrentLine	MessageItemsNumber
FlightPlanWaypointApproachIndex	FlightPlanApproachWaypointsNumber

The table below shows the results of a NearestVor search. It demonstrates the line-by-line list nature of the retrieved data and the different types of VOR information available from a FS9 NearestVor search - the ICAO, VOR Ident, Type, Frequency, Distance and True Bearing to VOR. The *nearest* VOR to the reference latitude and longitude is CurrentLine=0 (Index 0), the FFA VOR, which is 12.4 nmiles distant.

NEAREST VOR SEARCH

51.4943	:Current Lat	10	:Max Items	10	:Items Num
-1.6551	:Current Lon	100	:Max Dist	62	:Filter

NearestVorCurrent

	ICAO	111					
Line	123456789012		Ident	Type	Freq	Dist	Brg
0	VEG	FFA	FFA	3	113.40	12.4	334
1	VEG	LYE	LYE	3	109.80	12.6	274
2	VEG	BZN	BZN	3	111.90	15.4	7
3	VEG	CPT	CPT	2	114.35	16.3	90
4	VEG	BDN	BDN	3	108.20	21.0	190
5	VEG	OX	OX	3	117.70	23.8	31
6	VEG	GOS	GOS	3	115.55	30.6	322
7	VEG	ODH	ODH	3	109.60	30.8	120
8	VEG	BLC	BLC	3	116.20	32.0	108
9	VEG	SAM	SAM	2	113.35	34.4	160

DISPLAY (use of the extracted data)

Extracted data are either displayed to the pilot on the screen of a gauge like the gps_500, or used in calculations in the gauge's code. In either case, the Index pointer / CurrentLine must first be specified to access any information that is indexed, that is in the form of a list. It's important to note that defining the index value and then display or other calculation using the extracted data occur in the same gauge update cycle. Only the database search or an ICAO Transfer consume multiple gauge update cycles.

Simple displays of extracted lists are common. The easiest way to do this is through the use of {loop} … {next} in a <String> section within an <Element>. The xml used to display the NearestVor list is shown below:

```
<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}
                            %(11 (>@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
                                %(11 ++ s1 12 &lt;)
                        %{next}
                    %{end}
            </String>
        </FormattedText>
</Element>
```

The display is constructed one line at a time as the required index pointer, NearestVorCurrentLine, is incremented each pass through the loop. Line 82 is the indexing action, lines 83 through 89 are the display instructions for the list of VORs, and line 90 is the 'incrementer'. Display Loops are discussed further in the <ELEMENT> DISPLAY LOOPS chapter.

Similarly, data used in calculations rather than screen display must first be extracted from the database, then an index pointer specified.

```
2 (>c:fs9gps:NearestVorCurrentLine, enum)
```

places the index pointer to the third line of the NearestVor data extraction. Then, in the case of the NearestVor search above,

```
(C:fs9gps:NearestVorCurrentFrequency, MHz)
```

returns 111.90.

In another gauge in the panel, `(c:fs9gps:NearestVorCurrentFrequency, MHz)` will return 0.00 (or be empty for a string value) because indexed gps variables are local to the host gauge. As Tom Aguilo pointed out in one of his forum posts (edited to address NearestVor):

"the visibility/status of an indexed gps var is local to the gauge itself - its "instance" - In the other gauge case, NearestVorCurrentFrequency is a different "instance" of the same NearestVorCurrentFrequency found in the gauge containing the NearestVor search, therefore it needs to be initialized too.

There are other gpsvars that are "public", or visible through the entire panelset, for example FlightPlanIsActiveFlightPlan, FlightPlanTitle, etc.

It is important to add also that gpsvar names are case sensitive."

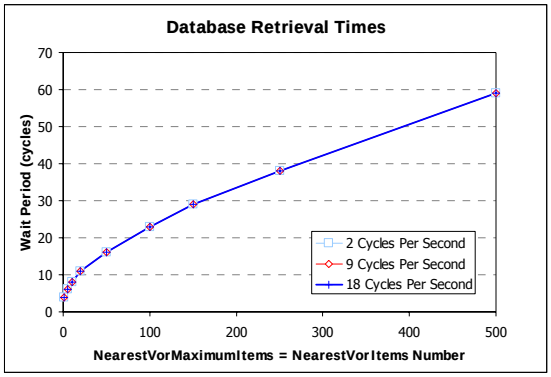
ASYNCHRONOUS OPERATION

Cycle skipping techniques

Information requested in an fs9gps database search is usually not available for use by the gauge, even for simple data display, in the same gauge update cycle that the search is initiated. Often, this causes no problems and it does not need to be addressed. There are some situations, however, where it *may or probably will* adversely affect subsequent code if that code requires the results of the database search before its execution begins. Because of this, it may be necessary to use cycle skipping techniques to delay execution of code that uses the requested data until they become available.

Recapping some discussion from the GPS Database Search section:

- ❑ even small searches can require several cycles for data retrieval
- ❑ waiting on gauge update cycles rather than absolute elapsed time for data retrieval is the key
- ❑ the bigger the search (i.e., the more items), the longer the wait



WHEN IS CYCLE SKIPPING NECESSARY?

I believe the question is not whether a particular fs9gps operation is multi-cycle, that is, whether it requires multiple gauge update cycles to complete. It is whether or not execution of *subsequent* code that uses search data must be carefully timed to <u>not</u> begin until search data are retrieved.

Consider a situation where one may want to find the nearest seaplane base. There are a couple of ways to approach this depending upon what you want to do afterward. For the purposes of this section, one way is to perform a NearestAirport search, then in <Update>, loop through the NearestAirport search results list incrementing the Index pointer (NearestAirportCurrentLine) by 1 each gauge update cycle, searching for an airport with NearestAirportCurrentAirportKind = 3, which is a seaplane base.

The following xml <Update> examples demonstrate what happens if the loop through the NearestAirport list begins before the NearestAirport search has returned data, and two cycle skipping techniques that can be used to "wait" for the Nearest search to conclude before starting the loop through the search result list.

Following that is a discussion of the cycle skipping requirements associated with ICAO Transfers.

EXAMPLE 1 – Nearest Searches

The following shows the aircraft position and the results of a NearestAirport search. Search parameters specify 100 maximum items and a 100 nmile maximum distance. The search is completed, returning the maximum list of 100 airports, on the 6th update cycle. The nearest Airport is Snohomish County (KPAE), while the nearest seaplane base is Kenmore Air Harbor (S60).


```
NEAREST AIRPORT SEARCH
  47.8335 :Current Lat  100 :Max Items 100 :Items Num
-122.3760 :Current Lon  100 :Max Distance

Current  ICAO     111 ---------------------- Current -------------------------
Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com     Freq Length
0        A      KPAE   KPAE    1  179   5.8   41   13   ILS  twr  120.20   9005
1        A      S13     S13    1  180   5.8   73    0              0.00   1680
2        A      S60     S60    3  180   6.7  135    0        CTF  122.70  10000
3        A      96WA   96WA    2  180   9.7   66    0              0.00   1100
4        A      W10     W10    1  180  11.4  347    0        CTF  122.90   2400
5        A      S43     S43    1  160  11.8   68    0        CTF  123.00   2660
6        A      W55     W55    3  180  12.4  173    0        CTF  122.90   5000
7        A      83Q     83Q    3  150  12.5  241    0        CTF  122.90  12000
8        A      0W0     0W0    3  201  12.5  172    0        CTF  122.90   9500
9        A      76WA   76WA    2   91  14.2   43    0              0.00   2500
```

(truncated after 10 airports to fit on this page)

What happens when no cycle skipping code is used?

The following code initiates the NearestAirport and begins inspecting the search results, looking for a seaplane base, in the same gauge update cycle that the Nearest search begins:

```
4       <Macro Name="c">@C</Macro>
5       <Macro Name="C">@C</Macro>
6
7   <Update Frequency="18" Hidden="No">
8   <!-- NearestAirport Search RESET -->
9       (L:Asynchronous Example 1 Gauge Reset, bool) 0 ==
10          if{
11              1 (>L:Asynchronous Example 1 Gauge Reset, bool)
12              -1 (>L:IndexPointer, enum)
13              1 (>L:LoopingThroughNearestAirportList, bool)
14
15
16          }
17
18  <!-- Set Search variables -->
19      100 (>@C:NearestAirportMaximumItems, enum)
20      100 (>@C:NearestAirportMaximumDistance, nmiles)
21
22  <!-- Set Aircraft position reference point -->
23      (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
24      (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)
25
26  <!-- Loop through the NearestAirport List searching for Seaplane Bases, AirportKind = 3 -->
27
28
29
30              (L:LoopingThroughNearestAirportList, bool) 1 ==
31                  if{
32                      (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
33                      (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
34                      (@C:NearestAirportCurrentAirportKind) 3 ==
35                          if{
36                              (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
37                              (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
38                              0 (>L:LoopingThroughNearestAirportList, bool)
39                          }
40                  }
41
42  </Update>
```

- **Lines 9–16:** Search RESET. The gauge that this code snippet is from contains a Gauge Reset mouse click button. Clicking it performs the following:

  ```
  0 (>L:Asynchronous Example 1 Gauge Reset, bool)
  (>K:RELOAD_PANELS)
  ```

  Then, in the subsequent gauge update cycle, lines 11 through 15 are executed.

- **Lines 27–41:** Loop through the NearestAirport search results list looking for the first occurrence of NearestAirportCurrentAirportKind = 3. When found, LVars `L:NrstArptCurDist` and `L:NrstArptCurTruBrg` are written and the looping stops, `0 (>L:LoopingThroughNearestAirportList, bool)`.

In this example, looping through the NearestAirport list looking for a seaplane base begins in the same gauge update cycle as the start of the NearestAirport search itself. The problem is that 5 update cycles are consumed before the NearestAirport list is returned and by that time, the loop that checks the search results is already at L:IndexPointer = NearestAirportCurrentLine = 4. Starting at this point in the search results list, the first seaplane base (CurrentAirportKind = 3) found will be in CurrentLine 6, Airport Ident W55, which is not the nearest seaplane base to the aircraft.

TECHNIQUE 1 - Cycle Counting

The next code employs a cycle counting routine that delays execution of the loop that checks for the first seaplane base until a prescribed number of gauge update cycles have passed since the NearestAirport search was initiated. The goal is to give the NearestAirport search time to return results before looking through them for a seaplane base.

```
    <Macro Name="c">@C:</Macro>
    <Macro Name="C">@C:</Macro>

<Update Frequency="18" Hidden="No">
<!-- NearestAirport Search RESET -->
    (L:Asynchronous Example 1 Gauge Reset, bool) 0 ==
        if{
            1 (>L:Asynchronous Example 1 Gauge Reset, bool)
            -1 (>L:IndexPointer, enum)
            1 (>L:LoopingThroughNearestAirportList, bool)
            4 (>L:NrstArptSearchCyclesToSkip, enum)
            0 (>L:NrstArptSearchCycleSkipCounter, enum)
        }

<!-- Set Search variables -->
    100 (>@C:NearestAirportMaximumItems, enum)
    100 (>@C:NearestAirportMaximumDistance, nmiles)

<!-- Set Aircraft position reference point -->
    (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
    (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)

<!-- Loop through the NearestAirport List searching for Seaplane Bases, AirportKind = 3 -->
    (L:NrstArptSearchCycleSkipCounter, enum) 1 + (>L:NrstArptSearchCycleSkipCounter, enum)
    (L:NrstArptSearchCycleSkipCounter, enum) (L:NrstArptSearchCyclesToSkip, enum) &gt;
        if{
            (L:LoopingThroughNearestAirportList, bool) 1 ==
                if{
                    (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
                    (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
                    (@C:NearestAirportCurrentAirportKind) 3 ==
                        if{
                            (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
                            (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
                            0 (>L:LoopingThroughNearestAirportList, bool)
                        }
                }
        }
</Update>
```

- **Lines 9–16:** Same comments apply, as above.
- **Lines 27–41:** Same comments apply, as above.
- **Line 14:** The pre-selected number of update cycles to skip.
- **Line 15:** The cycle skip counter. It is reset to 0 when the Search RESET mouse button is clicked.
- **Line 27:** The cycle skip counter is incremented by 1 each update cycle.
- **Lines 28-29:** When the cycle skip counter is greater than the number of cycles to skip, looping through the NearestAirport search list is allowed to begin (lines 30–40).

When `L:NrstArptSearchCyclesToSkip` is set too low, lines 30 - 40 begin execution before the NearestAirport search has returned data, and the same thing happens as in the first example, and the nearest seaplane base may be missed because the L:IndexPointer = NearestAirportCurrentLine may be already past it before NearsetAirport results are available.

In this example, when `L:NrstArptSearchCyclesToSkip` = 4 or less, the first seaplane base found is ident W55, which is incorrect. When `CyclesToSkip` is set to 5 or more, the first seaplane base found is ident S60, which is correct.

TECHNIQUE 2 - Let fs9gps tell you when it's ready

In this example, progress of the NearestAirport search is checked each gauge update cycle. Line 28 checks if NearestAirportItemsNumber, which is the total number of airports found in the search, is greater than zero. When starting the database search, this value is zero, but when the NearestAirport search returns data, ItemsNumber reflects the number of airports found. Assuming the NearestAirport search found at least one airport within the specified search distance, ItemsNumber becomes greater than zero, and only then are lines 30 – 40 executed. This time, the NearestAirport list is available before the loop begins and the nearest seaplane base, S60, is not bypassed.

```
4       <Macro Name="c">@C</Macro>
5       <Macro Name="C">@C</Macro>
6
7   <Update Frequency="18" Hidden="No">
8   <!-- NearestAirport Search RESET -->
9       (L:Asynchronous Example 1 Gauge Reset, bool) 0 ==
10          if{
11              1 (>L:Asynchronous Example 1 Gauge Reset, bool)
12              -1 (>L:IndexPointer, enum)
13              1 (>L:LoopingThroughNearestAirportList, bool)
14
15
16          }
17
18  <!-- Set Search variables -->
19      100 (>@C:NearestAirportMaximumItems, enum)
20      100 (>@C:NearestAirportMaximumDistance, nmiles)
21
22  <!-- Set Aircraft position reference point -->
23      (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
24      (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)
25
26  <!-- Loop through the NearestAirport List searching for Seaplane Bases, AirportKind = 3 -->
27
28      (@C:NearestAirportItemsNumber) 0 &gt;
29          if{
30              (L:LoopingThroughNearestAirportList, bool) 1 ==
31                  if{
32                      (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
33                      (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
34                      (@C:NearestAirportCurrentAirportKind) 3 ==
35                          if{
36                              (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
37                              (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
38                              0 (>L:LoopingThroughNearestAirportList, bool)
39                          }
40                  }
41          }
42  </Update>
```

- **Lines 9–16:** Same comments apply, as above.
- **Lines 27–41:** Same comments apply, as above.

This technique is employed in the FS9 gps_500 gauge where screen display loop of Nearest search results does not begin until ItemsNumber value is non-zero (for example, see line 2016). As a side comment, it isn't actually necessary that the gps_500 does that in this particular situation – to delay a *display* loop until the search concludes. If the user is only *displaying* the Nearest list, the display script will simply display blank values until the search concludes, at which point, the full list will be displayed as normal.

Edit line 2016 to read:

```
%((@c:NearestAirportItemsNumber) s2 )
```

and delete lines 2017 and 2034 to check this.

EXAMPLE 2 – Cycle Counting Technique for ICAO Transfers

ICAO Transfers are another multi-cycle operation. Data from the Waypoint Group are not available during the same update cycle that the ICAO Transfer is executed. In some circumstances, data may not be accessible even in the subsequent cycle and a cycle counting technique to skip more than one cycle is needed.

Building on the NearestAirport-seaplane base search example, the following statements have been added to the example code: a NearestAirport to WaypointAirport ICAO Transfer, cycle counting code following the ICAO Transfer, and a WaypointAirport Group write-to-LVar statement.

- **Line 41:** The ICAO Transfer. NearestAirport to WaypointAirport Group.
- **Line 42:** Write WaypointAirportLatitude to an LVar, same cycle as ICAO Xfer.
- **Line 43:** Write WaypointAirportLongitude to an LVar, same cycle as ICAO Xfer.
- **Lines 50 –59:**
  - If Line 44 `LoopingThroughNearestAirportList` is set to 2, then Lines 52 – 57 will be executed, otherwise they won't.
  - Starting in Line 52, the `ICAOXferCycleSkipCounter` is incremented by one each update cycle.
  - Only when it is equal to or greater than the prescribed `ICAOTransferCyclesToSkip` (Line 53, 54),
  - are the WaypointAirport Group variables Latitude and Longitude written to LVars (Lines 55, 56).
  - Finally, Line 57 sets `LoopingThroughNearestAirportList` to zero, terminating the entire Nearest Search and ICAO Transfer process.
- **Lines 14 –19:** Resets cycle counters and LVar values.

Note that in the following <Update> section, LVAR write statements (Lines 42, 43) are included immediately following the ICAO Transfer (Line 41) to illustrate that the WaypointAirport Group variables including Latitude and Longitude are not accessible in the same cycle as the ICAO Transfer.

```
<Update Frequency="2" Hidden="No">
<!-- ICAOSearch RESET -->
    (L:Asynch Example 3 Gauge Reset, enum) 1 ==
        if{
            0 (>L:Asynch Example 3 Gauge Reset, enum)
            -1 (>L:IndexPointer, enum)
            1 (>L:LoopingThroughNearestAirportList, enum)
            -1 (>L:ICAOXferCycleSkipCounter, enum)
            0 (>L:ICAOXferCyclesToSkip, enum)
            0 (>L:NrstArptCurDist, nmiles)
            0 (>L:NrstArptCurTruBrg, degrees)
            0 (>L:WptArptLat, degrees)
            0 (>L:WptArptLon, degrees)
        }

<!-- Set Search variables -->
    10 (>@C:NearestAirportMaximumItems, enum)
    100 (>@C:NearestAirportMaximumDistance, nmiles)

<!-- Set Aircraft position reference point -->
    (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
    (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)

<!-- Loop through the NearestAirport list searching for Seaplane Base, AirportKind = 3 -->
    (L:LoopingThroughNearestAirportList, enum) 1 ==
        if{
            (@C:NearestAirportItemsNumber) 0 !=
                if{
                    (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
                    (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
                    (@C:NearestAirportCurrentAirportKind) 3 ==
                        if{
                            (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
                            (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
                            (@C:NearestAirportCurrentICAO) (>@C:WaypointAirportICAO)
                            (@C:WaypointAirportLatitude, degrees) (>L:WptArptLat, degrees)
                            (@C:WaypointAirportLongitude, degrees) (>L:WptArptLon, degrees)
                            0 (>L:LoopingThroughNearestAirportList, enum)
                        }
                }
        }

<!-- After ICAO Transfer, delay accessing Waypoint Group variables using a Cycle Counting Technique -->
    (L:LoopingThroughNearestAirportList, enum) 2 ==
        if{
            (L:ICAOXferCycleSkipCounter, enum) ++ (>L:ICAOXferCycleSkipCounter, enum)
            (L:ICAOXferCycleSkipCounter, enum) (L:ICAOXferCyclesToSkip, enum) >=
                if{
                    (@C:WaypointAirportLatitude, degrees) (>L:WptArptLat, degrees)
                    (@C:WaypointAirportLongitude, degrees) (>L:WptArptLon, degrees)
                    0 (>L:LoopingThroughNearestAirportList, enum)
                }
        }
</Update>
```

When Line 44 is set to zero, the loop terminates after the LVar write statements (Lines 42, 43) that follow the ICAO Transfer (Line 41). The WaypointAirport variables Latitude and Longitude are not yet accessible, so the LVars remain 0.0000, as shown below.

Similarly, when Line 44 is set to 2, but with `ICAOXferCyclesToSkip` = 0 (Line 15), then the LVar write statements again occur in the *same* update cycle as the ICAO Transfer, the WaypointAirport variables Latitude and Longitude are not yet accessible, and the LVars remain 0.0000, as shown below (see lower right corner of the figure).

```
NEAREST AIRPORT SEARCH
  47.8335 :Current Lat   10 :Max Items  10 :Items Num
-122.3760 :Current Lon  100 :Max Distance

Current  ICAO     111 ---------------------- Current -------------------------
Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com     Freq Length
0        A      KPAE   KPAE    1  179   5.8   41   13   ILS  twr  120.20   9005
1        A      S13     S13    1  180   5.8   73    0              0.00   1680
2        A      S60     S60    3  180   6.7  135    0        CTF  122.70  10000
3        A      96WA   96WA    2  180   9.7   66    0              0.00   1100
4        A      W10     W10    1  180  11.4  347    0        CTF  122.90   2400
5        A      S43     S43    1  160  11.8   68    0        CTF  123.00   2660
6        A      W55     W55    3  180  12.4  173    0        CTF  122.90   5000
7        A      83Q     83Q    3  150  12.5  241    0        CTF  122.90  12000
8        A      OW0     OW0    3  201  12.5  172    0        CTF  122.90   9500
9        A      76WA   76WA    2   91  14.2   43    0              0.00   2500

NEAREST AIRPORT > LVars                  WAYPOINT AIRPORT > LVars

(L:NrstArptCurDist, nmiles):     6.7     (L:WptArptLat, degrees):    0.0000
(L:NrstArptCurTruBrg, degrees):  135     (L:WptArptLon, degrees):    0.0000
```

However, if Line 44 is set to 2 and Line 15 is edited to read

```
1 (>L:ICAOXferCyclesToSkip, enum)
```

then the write statements in Lines 55 & 56 are delayed one update cycle from the ICAO Transfer and the WaypointAirport Lat & Lon for seaplane base S60 are now accessed:

```
NEAREST AIRPORT SEARCH
  47.8335 :Current Lat   10 :Max Items  10 :Items Num
-122.3760 :Current Lon  100 :Max Distance

Current  ICAO     111 ---------------------- Current -------------------------
Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com     Freq Length
0        A      KPAE   KPAE    1  179   5.8   41   13   ILS  twr  120.20   9005
1        A      S13     S13    1  180   5.8   73    0              0.00   1680
2        A      S60     S60    3  180   6.7  135    0        CTF  122.70  10000
3        A      96WA   96WA    2  180   9.7   66    0              0.00   1100
4        A      W10     W10    1  180  11.4  347    0        CTF  122.90   2400
5        A      S43     S43    1  160  11.8   68    0        CTF  123.00   2660
6        A      W55     W55    3  180  12.4  173    0        CTF  122.90   5000
7        A      83Q     83Q    3  150  12.5  241    0        CTF  122.90  12000
8        A      OW0     OW0    3  201  12.5  172    0        CTF  122.90   9500
9        A      76WA   76WA    2   91  14.2   43    0              0.00   2500

NEAREST AIRPORT > LVars                  WAYPOINT AIRPORT > LVars

(L:NrstArptCurDist, nmiles):     6.7     (L:WptArptLat, degrees):   47.7548
(L:NrstArptCurTruBrg, degrees):  135     (L:WptArptLon, degrees): -122.2593
```

An important note is that in some circumstances, **skipping more than one gauge update cycle following an ICAO Transfer may be required** before the Waypoint Group variables are accessible. With ICAO Transfers, unfortunately there is no gps variable that can be used with the "let fs9gps tell you when it is ready" technique, so cycle counting is necessary. Reading through the forums, you may run across examples advocating the use of cycle skipping toggles – very simple code that results in a one cycle skip. But if you unknowingly run into the situation where you need to skip more than one cycle following an ICAO Transfer but have used a single-cycle toggle, then

figuring out what is wrong with your code can be pretty frustrating. Been there, done that. Experiment to determine what you need.

To get a better sense of the sequence of events, slow the update frequency rate (Line 7) down to 2, and watch the display after Gauge Reset (lines 11 – 19). You should be able to discern the slight hesitation during the Nearest search, before the NearestAirport list is displayed. Next, the NearestAirport LVar values appear, followed a moment later by the WaypointAirport LVar values.

Finally, just to demonstrate a point, the LVar assignments could have been pulled out of the loop and placed by themselves elsewhere in the Update section. This results in continuous* writing of WaypointAirportLatitude and Longitude to LVars. That would eliminate the need for specific cycle skipping code because, eventually, the ICAO Transfer is completed, the Waypoint Group variables are finally accessible, and the WaypointAirport Latitude and Longitude will be written to the LVars.

```
7  <Update Frequency="18" Hidden="No">
8  <!-- ICAOSearch RESET -->
9      (L:Asynch Example 3 Gauge Reset, enum) 1 ==
10         if{
11             0 (>L:Asynch Example 3 Gauge Reset, enum)
12             -1 (>L:IndexPointer, enum)
13             1 (>L:LoopingThroughNearestAirportList, enum)
14             0 (>L:NrstArptCurDist, nmiles)
15             0 (>L:NrstArptCurTruBrg, degrees)
16             0 (>L:WptArptLat, degrees)
17             0 (>L:WptArptLon, degrees)
18         }
19
20 <!-- Set Search variables -->
21     10 (>@C:NearestAirportMaximumItems, enum)
22     100 (>@C:NearestAirportMaximumDistance, nmiles)
23
24 <!-- Set Aircraft position reference point -->
25     (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
26     (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)
27
28 <!-- Loop through the NearestAirport list searching for Seaplane Base, AirportKind = 3 -->
29     (L:LoopingThroughNearestAirportList, enum) 1 ==
30         if{
31             (@C:NearestAirportItemsNumber) 0 !=
32                 if{
33                     (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
34                     (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
35                     (@C:NearestAirportCurrentAirportKind) 3 ==
36                         if{
37                             (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
38                             (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
39                             (@C:NearestAirportCurrentICAO) (>@C:WaypointAirportICAO)
40                             0 (>L:LoopingThroughNearestAirportList, enum)
41                         }
42                 }
43         }
44
45 <!-- Write WaypointAirport Latitude and Longitude to LVars each update cycle -->
46     (@C:WaypointAirportLatitude, degrees) (>L:WptArptLat, degrees)
47     (@C:WaypointAirportLongitude, degrees) (>L:WptArptLon, degrees)
48
49  </Update>
```

* It is not a best practice to continuously execute code in the Update section when it is not necessary, so this code snippet is just for the purpose of illustrating a point.

ICAO SEARCH – No Cycle-Skipping Required

ICAOSearch is a single cycle operation, therefore, no cycle skipping code is required after ICAOSearch and before an ICAO Transfer.

CONDITIONAL TEXT DISPLAY

The section, "When Is Cycle Skipping Necessary?", began by saying that depending upon what the user wants to do next, there are alternative approaches to finding the nearest seaplane base. If simple display of the list of nearest seaplane bases is all that is required and no ICAO Transfer is needed, then conditional text display statements can be used to display only NearestAirportCurrentAirportKind = 3 airports, thus eliminating the need for the loop through the NearestAirport list in the <Update> section.
The display <Element> I use to display all of the nearest airports is:

```
62  <Element Name="NEAREST AIRPORT LOOP DISPLAY">
63      <Position X="10" Y="5"/>
64      <FormattedText X="800" Y="700" Font="Courier New" FontSize="12"
65      LineSpacing="12" Color="#100000" Bright="Yes">
66          <Color Value="blue"/>
67          <Color Value="darkgreen"/>
68          <String>
69          \{clr2}NEAREST AIRPORT SEARCH\n
70          \{clr3}%((C:fs9gps:NearestAirportCurrentLatitude, degrees))%!9.4f! :Current Lat
71          %((C:fs9gps:NearestAirportMaximumItems, enum))%!5d! :Max Items
72          %((@c:NearestAirportItemsNumber))%!4d! :Items Num\n
73          %((C:fs9gps:NearestAirportCurrentLongitude, degrees))%!9.4f! :Current Lon
74          %((C:fs9gps:NearestAirportMaximumDistance, nmiles))%!5d! :Max Distance\n
75          \n
76          \{clr2}Current  ICAO     111 ---------------------- Current -------------------------\n
77          \{clr2}Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com    Freq Length\n\{clr)
78              %((@c:NearestAirportItemsNumber) s2 0 !=)
79                  %{if}
80                      %(0 sp1)
81                      %{loop}
82                          %(l1 (>@c:NearestAirportCurrentLine))
83
84
85                                      %((@c:NearestAirportCurrentLine))%!-5d!
86                                      %((@c:NearestAirportCurrentICAO))%!16s!
87                                      %((@c:NearestAirportCurrentIdent))%!6s!
88                                      %((@c:NearestAirportCurrentAirportKind))%!5d!
89                                      %((@c:NearestAirportCurrentLongestAirportDirection, degrees))%!5d!
90                                      %((@c:NearestAirportCurrentDistance, nmiles))%!6.1f!
91                                      %((@c:NearestAirportCurrentTrueBearing, degrees))%!5d!
92                                      %((@c:NearestAirportCurrentBestApproachEnum))%!4d!
93                                      %((@c:NearestAirportCurrentBestApproach))%!6s!
94                                      %((@c:NearestAirportCurrentComFrequencyName))%!5s!
95                                      %((@c:NearestAirportCurrentComFrequencyValue, mhz))%!8.2f!
96                                      %((@c:NearestAirportCurrentLongestRunwayLength, feet))%!7d!\n
97
98                                  %(l1 ++ s1 l2 &lt;)
99                      %{next}
100                 %{end}
101         </String>
102     </FormattedText>
103 </Element>
```

To display only AirportKind = 3, Lines 83, 84, and 97 are added as follows:

```
<Element Name="NEAREST AIRPORT LOOP DISPLAY">
    <Position X="10" Y="5"/>
    <FormattedText X="800" Y="700" Font="Courier New" FontSize="12"
    LineSpacing="12" Color="#100000" Bright="Yes">
        <Color Value="blue"/>
        <Color Value="darkgreen"/>
        <String>
        \{clr2}NEAREST AIRPORT SEARCH\n
        \{clr3}%((C:fs9gps:NearestAirportCurrentLatitude, degrees))%!9.4f! :Current Lat
        %((C:fs9gps:NearestAirportMaximumItems, enum))%!5d! :Max Items
        %((@c:NearestAirportItemsNumber))%!4d! :Items Num\n
        %((C:fs9gps:NearestAirportCurrentLongitude, degrees))%!9.4f! :Current Lon
        %((C:fs9gps:NearestAirportMaximumDistance, nmiles))%!5d! :Max Distance\n
        \n
        \{clr2}Current  ICAO     111 ---------------------- Current -------------------------\n
        \{clr2}Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com    Freq Length\n\{clr}
            %((@c:NearestAirportItemsNumber) s2 0 !=)
                %{if}
                %(0 sp1)
                    %{loop}
                        %(l1 (>@c:NearestAirportCurrentLine))
                        %((@c:NearestAirportCurrentAirportKind) 3 ==)
                            %{if}\{nr}
                                %((@c:NearestAirportCurrentLine))%!-5d!
                                %((@c:NearestAirportCurrentICAO))%!16s!
                                %((@c:NearestAirportCurrentIdent))%!6s!
                                %((@c:NearestAirportCurrentAirportKind))%!5d!
                                %((@c:NearestAirportCurrentLongestAirportDirection, degrees))%!5d!
                                %((@c:NearestAirportCurrentDistance, nmiles))%!6.1f!
                                %((@c:NearestAirportCurrentTrueBearing, degrees))%!5d!
                                %((@c:NearestAirportCurrentBestApproachEnum))%!4d!
                                %((@c:NearestAirportCurrentBestApproach))%!6s!
                                %((@c:NearestAirportCurrentComFrequencyName))%!5s!
                                %((@c:NearestAirportCurrentComFrequencyValue, mhz))%!8.2f!
                                %((@c:NearestAirportCurrentLongestRunwayLength, feet))%!7d!\n
                            %{end}
                        %(l1 ++ s1 l2 &lt;)
                    %{next}
                %{end}
        </String>
    </FormattedText>
</Element>
```

which results in the following screen display:

```
NEAREST AIRPORT SEARCH
  47.8335 :Current Lat  100 :Max Items 100 :Items Num
-122.3760 :Current Lon  100 :Max Distance

Current  ICAO     111 ---------------------- Current -------------------------
Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com    Freq Length
2        A       S60      S60     3   180   6.7  135    0        CTF  122.70  10000
6        A       W55      W55     3   180  12.4  173    0        CTF  122.90   5000
7        A       83Q      83Q     3   150  12.5  241    0        CTF  122.90  12000
8        A       0W0      0W0     3   201  12.5  172    0        CTF  122.90   9500
19       A       W36      W36     3   140  20.4  163    0        CTF  124.70   5000
41       A       2WA2    2WA2     3   200  27.4  186    0                0.00  15000
45       A       WN19    WN19     3   110  30.9  196    0                0.00   6000
69       A       21H      21H     3   156  40.9  343    0        CTF  128.25   5000
74       A       W37      W37     3    40  42.2  190    0        CTF  122.90   5500
91       A       WA81    WA81     3    70  46.4  332    0                0.00   4000
```

ICAO SEARCH EXAMPLE

The following demonstrates the ICAOSearch process of retrieving a facility ICAO given the StartCursor search filter and facility Ident. It's a situation that could arise if the user needs the latitude and longitude of a particular VOR when the Ident is known, something that cannot be done simply with A:Vars. A solution would be to 1) perform ICAOSearch to retrieve the VOR's ICAO, then, 2) transfer the ICAO to the WaypointVor or Facility Group to gain access the VOR's Lat and Lon.

As an example, "What's the Lat and Lon of the Corvallis, Oregon, USA VOR (Ident = "**CVO**")?"

STEP 1 - ICAOSearch. ICAOSearch has two required inputs, the search filter IcaoSearchStartCursor, and the facility Ident which is entered using the variable IcaoSearchEnterChar.

Before entry begins, all strings and enums in the ICAOSearch Group are blank and zero:

1.1 – First, enter the ICAOSearch filter, IcaoSearchStartCursor. ICAOSearch always begins by entering IcaoSearchStartCursor. In this example, it is '**V**' for VOR. A keyboard direct entry statement would be in the form of:

```
<On Key="AlphaNumeric">
      <Visible>(L: AlphaNumericEntryEnable, enum) 101 ==</Visible>
      (M:Key) chr (>C:fs9gps:IcaoSearchStartCursor)
</On>
```

An equivalent statement:

```
      'V' (>C:fs9gps:IcaoSearchStartCursor)
```

After the ICAO search filter has been entered, ICAOSearch immediately returns the first ICAO that matches the criterion. In this case, CurrentIcaoType is "**V**", and the first VOR Ident in the database is **1CD** (CurrentIdent). The associated ICAO is **VCY**_ _ _ _**1CD**_ _. The Region is **CY**, and is part of the 12 character ICAO. There is only one VOR with the Ident **1CD** in the fs9gps database, hence MatchedIcaosNumber = 1.

```
MatchedIcaosNumber: 1

          ICAO      111
Index     123456789012
0         VCY     1CD
```

At this point, only StartCursor has been entered. No Ident search string, or portion of the Ident, has been entered yet.

1.2 – Next, begin entering the Ident using IcaoSearchEnterChar. "**C**" is entered.

```
'C' (>C:fs9gps:IcaoSearchEnterChar)
```

IcaoSearchEnterChar is the heart of the ICAOSearch. It is how Ident is entered. The ICAOSearch engine subsequently searches the database and returns an ICAO that matches the ICAO Type defined by StartCursor and the Ident defined by EnterChar.

After "**C**" is entered, the following automatically occurs:

- The CursorPosition advances one place to Position 1, ready for the next character of the Ident to be entered.
- The first VOR Ident in the fs9gps database that begins with "**C**" is the **CA** VOR. Its ICAO is **V**_ _**SOCACA**_ _ _. From this ICAO one can tell that it is an ILS or LOC because Region is blank and the owning airport, **SOCA**, is listed in character positions 4 through 7. MatchedIcaosNumber is 1, meaning that there is only one VOR in the database with Ident = **'CA'**.

```
MatchedIcaosNumber: 1

          ICAO      111
Index     123456789012
0         V   SOCACA
```

'V SOCACA' is the ILS/DME 08 at Rochambeau Airport, Cayenne, French Guiana.

1.3 – IcaoSearchEnterChar. "**V**" is entered.

```
'V' (>C:fs9gps:IcaoSearchEnterChar)
```

After "**V**" was entered, the following automatically occurs:

- "**C**", previously entered, and "**V**" are concatenated to form "**CV**".

- The CursorPosition advances one place to Position 2, ready for the next character of the Ident to be entered.
- ICAOSearch returns three VORs whose Ident is **'CV':**

```
MatchedIcaosNumber: 3

          ICAO      111
Index     123456789012
0         V   EGDCCV
1         V   LFKCCV
2         VYB     CV
```

1.4 – IcaoSearchEnterChar. Lastly, "**O**" is entered.

```
'O' (>C:fs9gps:IcaoSearchEnterChar)
```

After "**O**" is entered, the following automatically occurs:

- "**CV**" and "**O**" are concatenated to form "**CVO**". Entry of the "**CVO**" Ident is now complete. Even though keyboard direct entry always is a "one letter at a time" entry process, use of a shift register in the <On> statement is not necessary. The gps module automatically concatenates, thereby enabling continuous typing.
- The CursorPosition advances one place to Position 3.
- ICAOSearch returns two VORs whose Ident is **'CVO':**

```
MatchedIcaosNumber: 2

          ICAO      111
Index     123456789012
0         VHE     CVO
1         VK1     CVO
```

VHE_ _ _ _**CVO**_ _ is the ICAO of the CAIRO VOR located in Cairo, Egypt, not CORVALLIS VOR located in Corvallis, Oregon, USA. Both share the same Ident, "**CVO**". ICAOSearch located the two VORs with that Ident, but **VHE**_ _ _ _**CVO**_ _ comes alphabetically before **VK1**_ _ _ _**CVO**_ _, the ICAO of the Corvallis VOR, so it has Index value 0. If an ICAO transfer is made at this point, the Lat and Lon of the Cairo VOR will be accessed.

ICAOs returned from ICAOSearch are Indexed, and a pointer, IcaoSearchMatchedIcao, must be defined to access the ICAOs. The default index pointer is always 0, the first item in the list. Setting IcaoSearchMatchedIcao to 1 accesses the second VOR:

```
1 (>C:fs9gps:IcaoSearchMatchedIcao)
```

```
GPS Viewer 1.2
GET  SET  SLEN  1234567890123456789012  STRING  NUMBER
 G         2    3                       IcaoSearchCursorPosition, enum
 G         3    CVO                     IcaoSearchCurrentIdent, string
 G    S    12   VK1      CVO            IcaoSearchCurrentIcao, string
 G         1    V                       IcaoSearchCurrentIcaoType, string
 G         2    K1                      IcaoSearchCurrentIcaoRegion, string
 G         2    2                       IcaoSearchMatchedIcaosNumber, enum
 G    S    2    1                       IcaoSearchMatchedIcao, enum
```

STEP 2 - Transfer the ICAO to the WaypointVor Group. After setting the proper Index pointer, ICAO Transfer into the WaypointVor Group can be performed. The appropriate xml:

```
(C:fs9gps:IcaoSearchCurrentIcao) (>C:fs9gps:WaypointVorIcao)
```

```
GPS Viewer 1.2
GET  SET  SLEN  1234567890123456789012  STRING  NUMBER
 G    S    12   VK1      CVO            WaypointVorICAO, string
 G         3    CVO                     WaypointVorIdent, string
 G         9    CORVALLIS               WaypointVorName, string
 G         4      44.49956 95           WaypointVorLatitude, degrees
 G         7    -123.29366 95           WaypointVorLongitude, degrees
 G         2    3                       IcaoSearchCursorPosition, enum
 G         3    CVO                     IcaoSearchCurrentIdent, string
 G    S    12   VK1      CVO            IcaoSearchCurrentIcao, string
 G         1    V                       IcaoSearchCurrentIcaoType, string
 G         2    K1                      IcaoSearchCurrentIcaoRegion, string
 G         2    2                       IcaoSearchMatchedIcaosNumber, enum
 G    S    2    1                       IcaoSearchMatchedIcao, enum
```

Now, WaypointVorIcao = IcaoSearchCurrentIcao = **VK1**_ _ _ _**CVO**_ _, and variables WaypointVorLatitude and WaypointVorLongitude return the desired Lat and Lon of the CORVALLIS VOR.

RESOLVING MULTIPLE ICAO MATCHES

Because Idents are not unique, ICAO Searches often return multiple ICAO matches (IcaoSearchMatchedIcaosNumber > 1), as in the case of the 'CVO' VOR search.

There are a few ways to select the index pointer of the desired facility. The gps_500 gauge typically displays ICAOSearch results (or the Idents from) on the screen and the user selects the desired facility by manipulating a scroll bar and cursor.

Alternatively, code in the users gauge can determine the correct index pointer based on selection criteria. In the VOR example, the nearest VOR may be desired. The following code snippet demonstrates one way to select the nearest VOR. It involves an ICAO Transfer into the Facility Group to retrieve the Lat and Lon of each VOR returned by the ICAO search.

```
 9 <!-- LOOP THROUGH ICAO SEARCH RESULT TO FIND NEAREST FACILITY -->
10 (@c:IcaoSearchMatchedIcaosNumber, enum) 0 &gt; (L:ResolveMultipleICAOs, bool) 1 == and
11     if{
12         (L:ResolveMultipleICAOInit, bool) 1 ==
13             if{
14                 (A:PLANE LATITUDE, degrees) (>@c:GeoCalcLatitude1, degrees)
15                 (A:PLANE LONGITUDE, degrees) (>@c:GeoCalcLongitude1, degrees)
16                 99999 (>L:FacilityNearestDistance, nmiles)
17                 0 (>L:ICAOTransferCycleCounter, enum)
18                 3 (>L:NumberOfCyclesToSkip, enum)
19                 0 (>L:NearestFacilityIndex, enum)
20                 0 (>L:NumberOfFacilitiesChecked, enum)
21                 0 (>L:ResolveMultipleICAOInit, bool)
22             }
23
24         (L:NumberOfFacilitiesChecked, enum) (>@c:IcaoSearchMatchedIcao, enum)
25         (@c:IcaoSearchCurrentIcao) (>@c:FacilityICAO)
26
27         (L:ICAOTransferCycleCounter, enum) ++ (>L:ICAOTransferCycleCounter, enum)
28         (L:ICAOTransferCycleCounter, enum) (L:NumberOfCyclesToSkip, enum) &gt;
29             if{
30                 (@c:FacilityLatitude, degrees) (>@c:GeoCalcLatitude2, degrees)
31                 (@c:FacilityLongitude, degrees) (>@c:GeoCalcLongitude2, degrees)
32                 (@c:GeoCalcDistance, nmiles) (>L:FacilityCurrentDist, nmiles)
33                 (L:FacilityNearestDistance, nmiles) (L:FacilityCurrentDist, nmiles) &gt;
34                     if{
35                         (L:FacilityCurrentDist, nmiles) (>L:FacilityNearestDistance, nmiles)
36                         (@c:IcaoSearchMatchedIcao) (>L:NearestFacilityIndex, enum)
37                     }
38                 (L:NumberOfFacilitiesChecked, enum) ++ (>L:NumberOfFacilitiesChecked, enum)
39                 0 (>L:ICAOTransferCycleCounter, enum)
40                 (@c:IcaoSearchMatchedIcao, enum) 1 + (@c:IcaoSearchMatchedIcaosNumber, enum) &gt;=
41                     if{
42                         0 (>L:ResolveMultipleICAOs, bool)
43                     }
44             }
45     }
```

- **Line 10.** The Loop is executed if there are multiple ICAOs returned by ICAOSearch and `L:ResolveMultipleICAOs` mouse Area has been clicked.
- **Lines 12 through 22.** Loop parameters are initialized.
- **Line 24.** The Index Pointer is set.
- **Line 25.** The ICAO Transfer.
- **Lines 27 and 28.** The cycle skip code used after the ICAO Transfer. This is a cycle counting technique.
- **Lines 30 through 32.** GeoCalcDistance is calculated using the aircraft position as the Lat1, Lon1 reference point and the current Facility location as Lat2, Lon2. The result is stored as an L:Var.
- **Lines 33 through 37.** The current Facility distance is checked to see if it is the shortest and, if so, then its Index Pointer is stored as `L:NearestFacilityIndex`, which is the objective.
- **Lines 38 and 39.** The cycle counter is reset to zero and the Index Pointer is incremented to prepare for the next Facility in the ICAOSearch list.

- **Lines 40 through 43.** The loop terminates after the number of Facilities checked equals MatchedIcaosNumber.

The Loop is triggered by a click area that sets both `L:ResolveMultipleICAOs` and `L:ResolveMultipleICAOInit` to 1.

ICAO SEARCH AIRPORTS – A Special Case

If interested in Airports only, then ICAO Search is not necessary in order to find the unique Airport ICAO. The reason is that the 3 to 4 character Airport Ident is unique itself, and a full Airport ICAO involves simply concatenating the Airport Start Cursor, "A", with the Ident, as follows:

```
'A______' 'KLAX' scat
```

That is the full ICAO for KLAX Airport. The first part of the statement is an 'A' followed by six spaces. The Airport Ident could be entered via Direct Keyboard Entry or Mouse or Code (discussed in later chapters).

Defining the ICAO using Direct Keyboard Entry for the Ident might look something like the following:

```
'A______'
(L:IdentChar1, enum) chr scat
(L:IdentChar2, enum) chr scat
(L:IdentChar3, enum) chr scat
(L:IdentChar4, enum) chr scat
```

Avoiding the ICAO Search using this shortcut is a special case that is safe only for Airports.

ICAO TRANSFER

The ICAO Transfer is a simple technique used to move from one gps group into another in order to access additional information (i.e., variables) regarding a specific Airport, VOR, NDB, or Intersection / Waypoint - information that is contained in the second group, but not in the first.

It is an important technique to understand as far as FS9 goes. Although useful while working with the FSX gps module, some of the need for the ICAO Transfer has been alleviated because, with FSX, the Nearest Groups (the 'first' group) are populated with many more variables from the Waypoint Groups (the 'second' group) to begin with. Makes life much easier.

An example is the best way to explain the technique.

ICAO TRANSFER EXAMPLE – NearestAirport > WaypointAirport

A good example of ICAO Transfer in fs9gps is the access of additional airport information following a NearestAirport search. Suppose the user wants a list of all frequencies from the nearest airport. The solution begins with a NearestAirport search, the results of which are shown below. In this example, all of the variables that are available in the NearestAirport Group are displayed.

```
NEAREST AIRPORT SEARCH
  41.8364 :Current Lat    10 :Max Items   10 :Items Num
 -88.2098 :Current Lon    75 :Max Distance

Current  ICAO     111 ---------------------- Current -------------------------
Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com     Freq Length
0        A      KDPA   KDPA    1   14   4.6  338   13   ILS  twr  120.90   7573
1        A      LL10   LL10    1  179   6.1  177    0                0.00   2580
2        A      1C5     1C5    1  178   9.2  157    8   GPS  CTF  122.90   3363
3        A      85LL   85LL    2  359   9.4  194    0                0.00   2300
4        A      06C     06C    1  109  10.4   28    0        CTF  123.00   3800
5        A      IS23   IS23    3  179  11.0  341    0                0.00   3000
6        A      LL22   LL22    1   89  11.3  123    0                0.00   2800
7        A      KARR   KARR    1   89  12.5  252   13   ILS  twr  120.60   6491
8        A      2IS0   2IS0    1  179  13.4  142    0                0.00   2900
9        A      LL51   LL51    2  359  14.2  192    0                0.00   2000
```

FS9 NearestAirport Group information available from a NearestAirport search is:

- 12 Character ICAO Identification
- Airport Ident
- Airport Kind (Class) Code (hard surface = 1, soft = 2, water = 3, etc)
- Longest Runway Direction
- Distance to the Airport from the reference point (aircraft)
- True Bearing to the Airport from the reference point
- Best (most precise) Approach Code

- Best (most precise) Approach Name
- Com Frequency Name (but just one, the principal airport control, usually 'twr' or 'CTF' if there is either, but never Ground or other)
- Com Frequency Value (but just one, and if there are multiple Tower frequencies, only the first)
- Longest Runway Length

However, in FS9, there is much more airport information, such as all airport frequencies, located in the WaypointAirport Group. Unfortunately, with FS9, the WaypointAirport Group variables are not accessible simply by performing a NearestAirport search.

To display the list of all frequencies of the nearest airport, the user must 'transfer' into the WaypointAirport Group where they are located. This is a simple process of sending the ICAO of the nearest airport obtained from the NearestAirport search to WaypointAirport. The xml instruction, which should be inserted in the <Update> section is:

```
0 (>C:fs9gps:NearestAirportCurrentLine) // Index pointer for the nearest

(C:fs9gps:NearestAirportCurrentICAO)
(>C:fs9gps:WaypointAirportICAO)
```

Now, WaypointAirport has a specific ICAO to work with, and all variables subsequently accessed in the WaypointAirport Group, such as the list of frequencies, return information about only that airport.

```
WAYPOINT AIRPORT FREQUENCIES
ICAO     111    12 :Frequencies Num
123456789012
A      KDPA  :Waypoint Airport ICAO

--- Waypoint Airport Frequency ----
Freq        Name  Limit  Value Type
0       Approach      0   133.50    1
1           ATIS      1   124.80    1
2      Clearance      0   119.75    1
3      Departure      0   133.50    1
4            FSS      0   122.10    1
5            FSS      0   122.30    1
6         Ground      0   121.80    1
7          Tower      0   120.90    1
8          Tower      0   124.50    1
9         Unicom      0   122.95    1
10       ILS 02L      0   111.70    2
11        ILS 10      0   109.50    2
```

A few comments:

- In this example, why doesn't one go directly to WaypointAirport for the frequency list to begin with? Because WaypointAirport does not have the ability to determine which airport is the nearest. WaypointAirport must always be told (e.g., by defining WaypointAirportICAO) which specific airport you are interested

in. This is one area in which FSX gps is easier to work with. Much airport information is available in the NearestAirport Group to begin with, unlike in FS9.

- The ICAO must be used for the transfer into another group. It is the unique database element identifier. Note that the NearestAirport to WaypointAirport transfer will not work using Ident:

```
0 (>C:fs9gps:NearestAirportCurrentLine)

(C:fs9gps:NearestAirportCurrentIdent)
(>C:fs9gps:WaypointAirportIdent)
```

- The airport frequencies accessed in WaypointAirport are an indexed list containing, in this example, 12 separate frequencies. To display or otherwise utilize any one of them requires an Index pointer. In the WAYPOINT APT FREQUENCIES DISPLAY LOOP section of the code below, line 88 is the Index pointer for the frequency list, and lines 89 through 93 are the screen display instructions. Refer to the Search> Index> Display section for more discussion.

The xml for Example 1:

```
1  <Gauge Name="NEAREST AIRPORT - AIRPORT FREQUENCIES" Version="1.0">
2  <Size X="800" Y="800" />
3
4      <Macro Name="c">C:fs9gps</Macro>
5      <Macro Name="C">C:fs9gps</Macro>
6
7  <Update Frequency="18" Hidden="No">
8  <!-- SET SEARCH VARIABLES -->
9      10 (>@c:NearestAirportMaximumItems, enum)
10     75 (>@c:NearestAirportMaximumDistance, nmiles)
11
12 <!-- SET REFERENCE POINT VARIABLES (AIRCRAFT POSITION) -->
13     (A:PLANE LATITUDE, degrees) (>@c:NearestAirportCurrentLatitude, degrees)
14     (A:PLANE LONGITUDE, degrees) (>@c:NearestAirportCurrentLongitude, degrees)
15
16 <!-- ICAO TRANSFER: NearestAirport to WaypointAirport -->
17     0 (>@c:NearestAirportCurrentLine)
18     (@c:NearestAirportCurrentICAO) (>@c:WaypointAirportICAO)
19
20 </Update>
21
22     <Element Name="BACKGROUND RECTANGLE">
23         <Position X="0" Y="0"/>
24         <Rectangle Width="800" Height="800" FillColor="white" Bright="Yes"/>
25     </Element>
```

```
<Element Name="NEAREST APT DISPLAY LOOP">
    <Position X="10" Y="5"/>
    <FormattedText X="800" Y="700" Font="Courier New" FontSize="12" LineSpacing="12"
    Color="#101010" Bright="Yes" >
        <Color Value="blue"/>
        <Color Value="darkgreen"/>
        <String>
        \{clr2}NEAREST AIRPORT SEARCH\n
        \{clr3}%((C:fs9gps:NearestAirportCurrentLatitude, degrees))%!9.4f! :Current Lat
        %((C:fs9gps:NearestAirportMaximumItems, enum))%!5d! :Max Items
        %((@c:NearestAirportItemsNumber))%!4d! :Items Num\n
        %((C:fs9gps:NearestAirportCurrentLongitude, degrees))%!9.4f! :Current Lon
        %((C:fs9gps:NearestAirportMaximumDistance, nmiles))%!5d! :Max Distance
        \n\n\{clr2}
        Current  ICAO     111 ---------------------- Current -------------------------\n
        Line     123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com    Freq Length\n
            %((@c:NearestAirportItemsNumber) s2 0 !=)
                %{if}
                    %(0 sp1)
                    %{loop}
                        %(l1 (>@c:NearestAirportCurrentLine))
                            \{clr}%((@c:NearestAirportCurrentLine))%!-5d!
                            %((@c:NearestAirportCurrentICAO))%!16s!
                            %((@c:NearestAirportCurrentIdent))%!6s!
                            %((@c:NearestAirportCurrentAirportKind))%!5d!
                            %((@c:NearestAirportCurrentLongestAirportDirection, degrees))%!5d!
                            %((@c:NearestAirportCurrentDistance, nmiles))%!6.1f!
                            %((@c:NearestAirportCurrentTrueBearing, degrees))%!5d!
                            %((@c:NearestAirportCurrentBestApproachEnum))%!4d!
                            %((@c:NearestAirportCurrentBestApproach))%!6s!
                            %((@c:NearestAirportCurrentComFrequencyName))%!5s!
                            %((@c:NearestAirportCurrentComFrequencyValue, mhz))%!8.2f!
                            %((@c:NearestAirportCurrentLongestRunwayLength, feet))%!7d!\n
                            %(l1 ++ s1 l2 &lt;)
                    %{next}
                %{end}
        </String>
    </FormattedText>
</Element>
```

```
<Element Name="WAYPOINT APT FREQUENCIES DISPLAY LOOP">
    <Position X="10" Y="190"/>
    <FormattedText X="800" Y="700" Font="Courier New" FontSize="12" LineSpacing="12"
    Color="#100000" Bright="Yes" >
        <Color Value="blue"/>
        <Color Value="darkgreen"/>
        <Color Value="gray"/>
        <String>
        \n
        \{clr2}WAYPOINT AIRPORT FREQUENCIES\n
        \{clr4}ICAO     111
        \{clr3}%((@c:WaypointAirportFrequenciesNumber))%!6d! :Frequencies Num\n
        \{clr4}123456789012\n
        \{clr3}%((@c:WaypointAirportICAO))%!12s! :Waypoint Airport ICAO
        \n\n\{clr2}
        ----- WaypointAirportFrequency -----\n
        Freq        Name  Limit  Value  Type\n
            %((@c:WaypointAirportFrequenciesNumber) s2 0 !=)
                %{if}
                    %(0 sp1)
                    %{loop}
                        %(l1 (>@c:WaypointAirportCurrentFrequency))
                            \{clr}%((@c:WaypointAirportCurrentFrequency))%!-3d!
                            %((@c:WaypointAirportFrequencyName))%!13s!
                            %((@c:WaypointAirportFrequencyLimit))%!5d!
                            %((@c:WaypointAirportFrequencyValue, mHz))%!9.2f!
                            %((@c:WaypointAirportFrequencyType))%!6d!\n
                            %(l1 ++ s1 l2 &lt;)
                    %{next}
                %{end}
        </String>
    </FormattedText>
</Element>
```

WAYPOINT AIRPORT GROUP

The WaypointAirport Group contains all variables associated with specific airports in fs9gps. The ICAO Identification must be specified before variables can be accessed, then all subsequent variables accessed in WaypointAirport return information about that airport until the ICAO is changed.

Frequencies, Transitions, Approaches and Runways are indexed variables (lists) requiring an Index Pointer to access specific items. The rest are non-indexed.

❑ WaypointAirportICAO (12 character string) [Get, Set]

The 12 character ICAO Identification for the specific airport.

❑ WaypointAirportIdent (3 to 4 character string) [Get]

The airport IDENT code. Note that this is not the same as the 12 character ICAO. Idents are three to four characters long and often begin with the first letter of the Region code.

❑ WaypointAirportKind (enum) [Get]

A number representing Airport Class.

WaypointAirportKind (Class)

#	Class (Kind)	#	Class (Kind)
0	UNKNOWN_KIND_AIRPORT = 0	3	WATER_SURFACE_AIRPORT = 3
1	HARD_SURFACE_AIRPORT = 1	4	HELIPAD_AIRPORT = 4
2	SOFT_SURFACE_AIRPORT = 2	5	PRIVATE_AIRPORT = 5

http://msdn.microsoft.com/en-us/library/cc526954.aspx#AirportClass

❑ WaypointAirportLongestRunwayDirection (degrees) [Get]

Direction (magnetic) of the longest runway.

❑ WaypointAirportType (enum) [Get]

A number representing Airport Type. Referred to as AirportPrivateType in the Microsoft online ESP SDK.

WaypointAirportType

#	Type	#	Type
0	UNKNOWN_TYPE_AIRPORT = 0	2	MILITARY_TYPE_AIRPORT = 2
1	PUBLIC_TYPE_AIRPORT = 1	3	PRIVATE_TYPE_AIRPORT = 3

http://msdn.microsoft.com/en-us/library/cc526954.aspx#AirportPrivateType

❑ WaypointAirportName (string) [Get]

Name of the airport. WaypointAirportName is the only 'name' in fs9gps that can be searched using NameSearch.

❑ WaypointAirportCity (string) [Get]

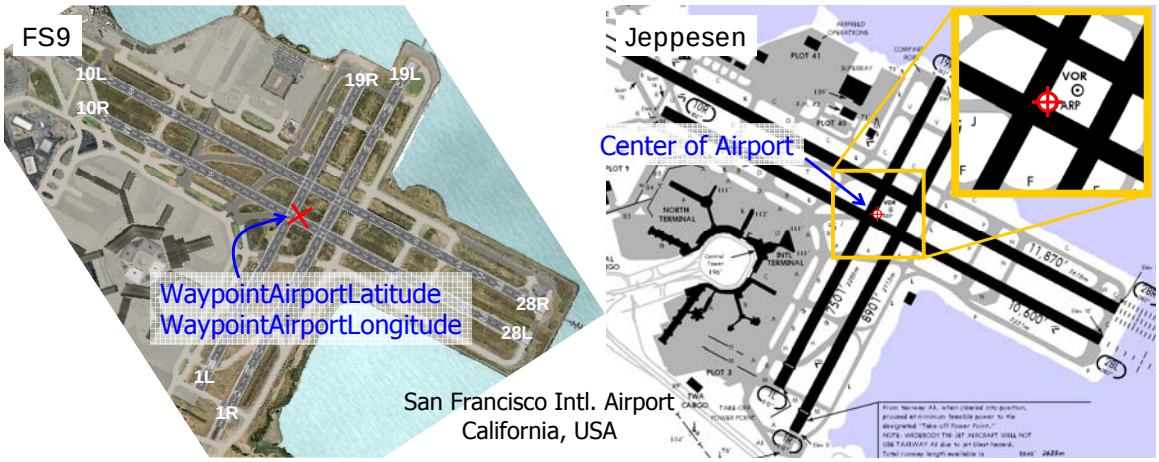
Airport city, and in the case of many airports in the North America, state or province.

❑ WaypointAirportRegion – not available

Does not exist. Although WaypointAirportRegion is listed in the SDKs as an FS9 and FSX gps variable, Region is absent in the WaypointAirportICAO, and therefore, AirportRegion is not a live variable in the WaypointAirport Group.

❑ WaypointAirportLatitude,
❑ WaypointAirportLongitude (degrees, radians) [Get]

The latitude and longitude of the airport. WaypointAirportLatitude and Longitude is the center of the runway, or in the case of large airports, the center of the airport facility. 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).



❑ WaypointAirportElevation (feet) [Get]

Airport elevation, asl.

❑ WaypointAirportFuel1 (string) [Get]

If Avgas is available at the airport, WaypointAirportFuel1 = 'Avgas'. If Avgas is not available, WaypointAirportFuel1 is blank.

❑ WaypointAirportFuel2 (string) [Get]

If Jet fuel is available at the airport, WaypointAirportFuel2 = 'Jet'. If Jet fuel is not available, WaypointAirportFuel2 is blank.

❑ WaypointAirportBestApproachEnum (enum) [Get]

A number representing the most precise approach available at the airport.

WaypointAirportBestApproach

#	Approach Type	#	Approach Type	#	Approach Type
0	UNKNOWN = 0	5	LORAN = 5	10	LDA = 10
1	VFR = 1	6	RNAV = 6	11	LOC = 11
2	HEL = 2	7	VOR = 7	12	MLS = 12
3	TACAN = 3	8	GPS = 8	13	ILS = 13
4	NDB = 4	9	SDF = 9		

http://msdn.microsoft.com/en-us/library/cc526954.aspx#AirportApproachType

❑ WaypointAirportBestApproach (string) [Get]

The name of the most precise approach available at the airport. Refer to table above.

❑ WaypointAirportRadarCoverage (enum) [Get]

This variable is not activated in FS9.

❑ WaypointAirportAirspace (string) [Get]

This variable is not activated in FS9.

❑ WaypointAirportTowered (bool) [Get]

Tower present = 1. No Tower = 0.

❑ WaypointAirportCurrentFrequency (enum) Get, Set]

Index pointer for the airport frequency list. The first frequency in the list is accessed by setting WaypointAirportCurrentFrequency=0.

❑ WaypointAirportFrequenciesNumber (enum) [Get]

Number of frequencies at the airport. Includes both Com and Nav (i.e., ILS & LOC) frequencies.

❑ WaypointAirportFrequencyName (string) [Get]

Name of the frequency. The communication frequency names are:

- ❑ Approach
- ❑ ATIS, ASOS, AWOS
- ❑ CTAF
- ❑ Unicom
- ❑ Multicom
- ❑ Clearance
- ❑ Ground
- ❑ Tower
- ❑ Departure
- ❑ FSS

Navigation frequency names are either ILS or LOC and include the runway number (e.g. ILS-24L)

❑ WaypointAirportFrequencyLimit (enum) [Get]

Frequency limited designation.

- ❑ 0 = No Limit. Transmit and Receive capability. This is the most common FrequencyLimit value in the fs9gps database.
- ❑ 1 = RX_ONLY. Receive Only. ATIS, ASOS, AWOS which transmit weather and airport information.
- ❑ 2 = TX_ONLY. Transmit only.
- ❑ 3 = PART_TIME.

❑ **WaypointAirportFrequencyValue (MHz) [Get]**

Radio frequency, usually expressed as MHz.

❑ **WaypointAirportFrequencyType (enum) [Get]**

1 = Communication frequency

2 = Navigation frequency (ILS or LOC)

❑ **WaypointAirportCurrentRunway (enum) [Get, Set]**

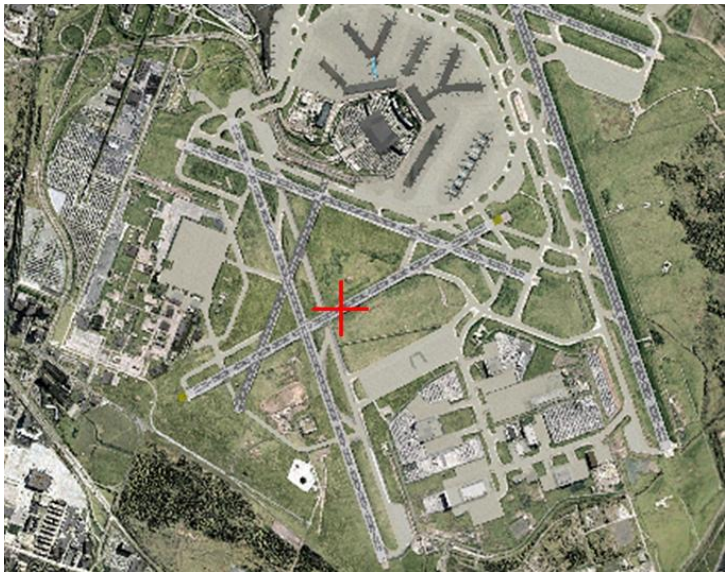
Index pointer for the airport runway list. The first runway in the list is accessed by setting WaypointAirportCurrentRunway=0.

❑ **WaypointAirportRunwaysNumber (enum) [Get]**

Number of runways at the airport. Every runway has two directions, but in the Waypoint Airport Runways list, it counts as one runway.

❑ **WaypointAirportRunwayLatitude**
❑ **WaypointAirportRunwayLongitude (degrees or radians) [Get]**

Latitude and longitude of the center point of the runway. The example below shows WaypointAirportRunwayLatitude and Longitude (41.98961 and -87.90514 degrees) for runway 04L-22R at Chicago O'Hare International Airport.



❑ WaypointAirportRunwayElevation (feet) [Get]

Elevation (asl) of the center point of the runway.

❑ WaypointAirportRunwayDirection (degrees) [Get]

Compass direction of the runway. Only the first of the pair of directions for each runway is returned. In the example above, WaypointAirportRunwayDirection of Runway 04L-22R is 39.44 degrees.

❑ WaypointAirportRunwayDesignation (string) [Get]

The designation, or name, of the runway. In the example above, "04L-22R".

❑ WaypointAirportRunwayLength (feet) [Get]

Length of the runway.

❑ WaypointAirportRunwayWidth (feet) [Get]

Width of the runway.

❑ WaypointAirportRunwaySurface (enum) [Get]

A number representing runway surface type.

RunwaySurfaceType

#	Surface Type	#	Surface Type	#	Surface Type
0	UNKNOWN = 0	105	GRAVEL = 105	112	SAND = 112
1	CONCRETE = 1	106	OIL_TREATED = 106	113	SHALE = 113
2	ASPHALT = 2	107	STEEL = 107	114	TARMAC = 114
101	GRASS = 101	108	BITUMINUS = 108	115	SNOW = 115
102	TURF = 102	109	BRICK = 109	116	ICE = 116
103	DIRT = 103	110	MACADAM = 110	201	WATER = 201
104	CORAL = 104	111	PLANKS = 111		

http://msdn.microsoft.com/en-us/library/cc526954.aspx#RunwaySurfaceType

❑ WaypointAirportRunwayLighting (enum) [Get]

A number representing airport lighting type. Note that this is not list of available lighting systems for a runway, such as VASI and REIL.

RunwayLightingType

#	Lighting Type	#	Lighting Type
0	UNKNOWN = 0	3	FULL_TIME = 3
1	NONE = 1	4	FREQUENCY = 4
2	PART_TIME = 2		

http://msdn.microsoft.com/en-us/library/cc526954.aspx#RunwayLightingType

RunwayLighting Types 2 and 4 may not exist, at least not in the fs9gps database. I have checked all runways at all airports in the fs9gps database within Europe and the USA and found no Type 2 or 4 RunwayLighting types.

❑ WaypointAirportCurrentApproach (enum) [Get, Set]

Index pointer for the airport approach list. The first approach in the list is accessed by setting WaypointAirportCurrentApproach=0.

❑ WaypointAirportApproachesNumber (enum) [Get]

The number of approach procedures for the selected airport.

❑ WaypointAirportApproachName (string) [Get]

The name of the selected approach, such as, "ILS 22R", "NDB 27R", or "RNAV 09L".

❑ WaypointAirportApproachGps (bool) [Get]

A designation indicating that the approach can be flown by the GPS receiver. ApproachGps = 1 = approach is approved for GPS use. ApproachGps = 0 = approach is not approved for GPS use. For these approaches, the GPS receiver can be used for supplemental information only.

❑ WaypointAirportApproachTransitionsNumber (enum) [Get]

The number of transitions available for the selected approach.

- **WaypointAirportApproachCurrentTransition (enum) [Get, Set]**

Index pointer for the approach transitions list. The first transition in the list is accessed by setting WaypointAirportCurrentTransition=0.

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

The name of the current transition, such as, "Vectors", or "PAPPI" (a waypoint approach fix).

- **WaypointAirportApproachTransitionLatitude**
- **WaypointAirportApproachTransitionLongitude (degrees or radians) [Get]**

The latitude and longitude of the center of the runway to which the selected approach applies.

- **WaypointAirportApproachTransitionSize (nmiles) [Get]**

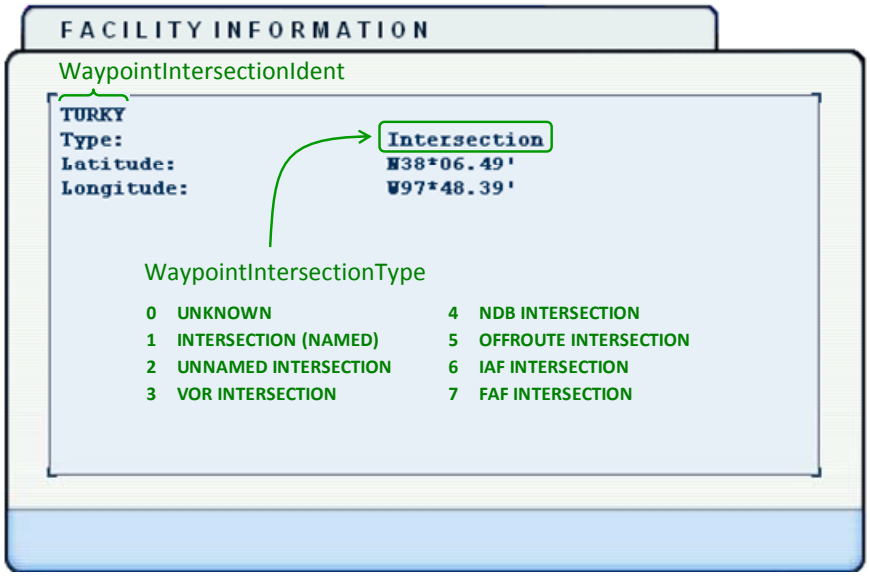
Size (radius, I assume) of the selected approach transition. It appears that all ApproachTransitionSize values in the fs9gps database are preset to 27.00 nmiles.

WAYPOINT INTERSECTION GROUP

The WaypointIntersection Group contains all variables associated with Waypoints and Intersections in the fs9gps database. The ICAO Identification must first be specified, then all variables accessed in WaypointIntersection return information about that Intersection or Waypoint until the ICAO is changed.

All variables in WaypointIntersection are non-indexed; there are no 'lists' of items associated with a specific Intersection (compared to WaypointAirport, where, for example, there are lists of different runways, approaches, transitions, and frequencies).

The screen shot below shows the Facility Information page of an example Intersection in FS9, indicating the associated gps variable names.



The following is a snapshot of all WaypointIntersection variables for the same Intersection.

```
GPS Viewer 1.2                  111111111112
GET  SET  SLEN  12345678901234567890  STRING NUMBER
 G    S    12   WK3KHUTTURKY          WaypointIntersectionICAO, string
 G         5    TURKY                 WaypointIntersectionIdent, string
 G         0                          WaypointIntersectionCity, string
 G         2    K3                    WaypointIntersectionRegion, string
 G         6      38.1081280          WaypointIntersectionLatitude, degrees
 G         7     -97.8065501          WaypointIntersectionLongitude, degrees
 G         2    1                     WaypointIntersectionType, enum
 G         3    GNP                   WaypointIntersectionNearestVorIdent, string
 G         2    2                     WaypointIntersectionNearestVorType, enum
 G         6    326.689753            WaypointIntersectionNearestVorTrueRadial, degrees
 G         6    320.689753            WaypointIntersectionNearestVorMagneticRadial, degrees
 G         10   158.043825            WaypointIntersectionNearestVorDistance, nmiles
```

❑ WaypointIntersectionICAO (string, SLEN=12) [Get, Set]

WaypointIntersectionICAO is the ICAO for the specific Intersection.

❑ WaypointIntersectionIdent (string) [Get]

The 4 to 5 character Intersection Ident.

❑ WaypointIntersectionType (enum) [Get]

An enum representing Intersection Type.

Bit	Name and Type #	Bit	Name and Type #
0	UNKNOWN = 0	4	NDB = 4
1	NAMED = 1	5	OFFROUTE = 5
2	UNNAMED = 2	6	IAF = 6
3	VOR = 3	7	FAF = 7

http://msdn.microsoft.com/en-us/library/cc526954.aspx#NearestIntersectionData

The great majority of Intersections in the fs9gps database are Type 1 = Named, followed by Type 2, 3 and 4, which is relatively unusual. It does not appear that there are any Type 0, 5, 6, or 7 Intersections in the database.

❑ WaypointIntersectionCity (string) [Get]

WaypointIntersectionCity is not populated in the fs9gps data base. It always returns a blank string.

❑ WaypointIntersectionRegion (string) [Get]

The two character Region code.

❑ WaypointIntersectionLatitude
❑ WaypointIntersectionLongitude (degrees or radians) [Get]

The latitude and longitude of the Intersection. 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).

❑ WaypointIntersectionNearestVorIdent (string) [Get]

The Ident of a VOR. It is usually NOT the nearest VOR, however.

- **WaypointIntersectionNearestVorType (enum) [Get]**

An enum representing the VOR Type of the VOR returned as "Nearest". It is the correct Type for that VOR, but the VOR is usually not the nearest to the Intersection.

VOR Type Bit Map Table

Bit	Name and Type #	Bit	Name and Type #
0	UNKNOWN = 0	4	TACAN = 4
1	VOR = 1	5	VORTAC = 5
2	VOR_DME = 2	6	ILS = 6
3	DME = 3	7	VOT = 7

http://msdn.microsoft.com/en-us/library/cc526954.aspx#VorType

- **WaypointIntersectionNearestVorTrueRadial (degrees) [Get]**
- **WaypointIntersectionNearestVorMagneticRadial (degrees) [Get]**
- **WaypointIntersectionNearestVorDistance (NMiles) [Get]**

The WaypointIntersectionNearestVor variables represent the radial bearings and distance from a VOR, but not necessarily the nearest VOR.

NOTE: WaypointIntersectionNearestVor appears to be a software bug. Exercise caution when using WaypointIntersectionNearestVor variables.

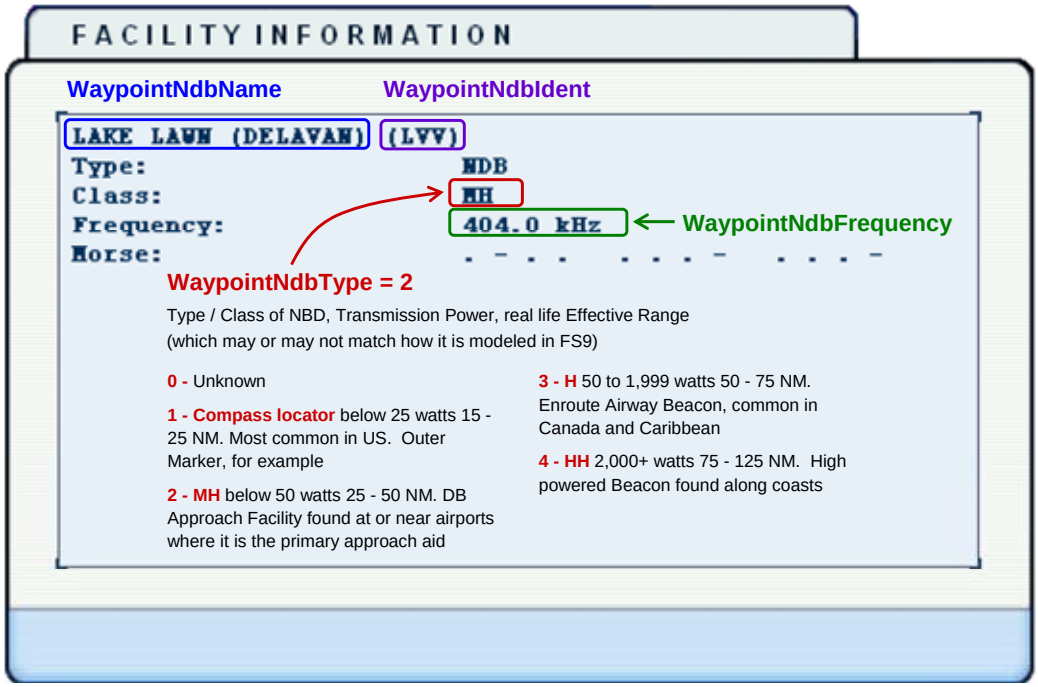
The NearestVOR search works properly, however. Use that approach to find the nearest VOR to an Intersection.

WAYPOINT NDB GROUP

The WaypointNdb Group contains all variables associated with specific Non-Directional Beacons in the fs9gps database. The ICAO Identification must be specified before variables can be accessed, then all subsequent variables accessed in WaypointNdb return information about that NDB until the ICAO is changed.

All variables in WaypointNdb are non-indexed; there are no 'lists' of items associated with a specific NDB (compared to WaypointAirport, where, for example, there are lists of different runways, approaches, transitions, and frequencies).

The figure below is a screen shot of the Facility Information page of an example NDB in FS9, indicating the associated gps variable names.



The following is a snapshot of all WaypointNdb variables for the same NDB.

```
GPS Viewer 1.2                  11111111112
GET  SET  SLEN  12345678901234567890  STRING NUMBER
 G    S    12   NK5      LVV          WaypointNdbICAO, string
 G         3    LVV                   WaypointNdbIdent, string
 G         2    2                     WaypointNdbType, enum
 G         19   LAKE LAWN (DELAVAN)   WaypointNdbName, string
 G         0                          WaypointNdbCity, string
 G         2    K5                    WaypointNdbRegion, string
 G         6      42.6988333          WaypointNdbLatitude, degrees
 G         7     -88.5931666          WaypointNdbLongitude, degrees
 G         6    947.998733            WaypointNdbElevation, feet
 G         9    404                   WaypointNdbFrequency, khz
 G         2    0                     WaypointNdbWeatherBroadcast, enum
 G         2     0.0                  WaypointNdbMagneticVariation, degrees
```

❑ WaypointNdbICAO (string, SLEN=12) [Get, Set]

The ICAO for the specific NDB. Some NDBs are nav fixes in fs9gps approach procedures. These NDBs include the "owning" airport Ident in ICAO character positions 4 through 7. See discussion in ICAO Idents.

❑ WaypointNdbIdent (string) [Get]

The 1 to 5 character NDB Ident

❑ WaypointNdbType (enum) [Get]

A number representing NDB Type (Class).

The following is a list of NDB Type and Class, real life Transmission Power, real life Effective Range (which may not match how it is modeled in FS9):

0 - Unknown. There appear to be no Type 0 NDBs in the fs9gps database.

1 - Compass Locator. Below 25 watts, 15 - 25 nmiles. Type 1 NDBs are absent within the U.S.A. in the fs9gps database, but are common in other parts of the world, especially Europe (eg, U.K.).

2 - MH. Below 50 watts, 25 - 50 nmiles. Directional Beacon Approach Facility found at or near airports where it is the primary approach aid. This is the most common type of NDB in the fs9gps database.

3 - H. 50 to 1,999 watts, 50 - 75 nmiles. Enroute Airway Beacon, common in Canada and Caribbean

4 - HH. 2,000+ watts, 75 - 125 nmiles. High powered Beacon found along coasts in the U.S.A.

❑ WaypointNdbName (string) [Get]

Name of the NDB. Interestingly, some NDBs also contain the city name in parenthesis following the NDB name – all part of the variable WaypointNdbName. I do not understand the rules/reasons that some do and some do not. In the example above, Lake Lawn is the NDB name, Delavan is the city. NDB Names are not searchable using NameSearch.

❑ WaypointNdbCity (string) [Get]

WaypointNdbCity is not populated in the fs9gps data base. It always returns a blank string.

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

The two character Region code.

- **WaypointNdbLatitude**
- **WaypointNdbLongitude (degrees or radians) [Get]**

The latitude and longitude of the NDB. 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).

- **WaypointNdbElevation (feet) [Get]**

Elevation (asl) of the NDB facility.

- **WaypointNdbFrequency (kHz) [Get]**

Radio frequency of the NDB. Commonly expressed in kHz.

- **WaypointNdbWeatherBroadcast (gps boolean) [Get]**

The ESP SDK indicates that WaypointNdbWeatherBroadcast is a gps boolean:

- 0 = Unknown
- 1 = No
- 2 = Yes

However, having scanned most NDBs in the fs9gps database, so far I have found all NDBs have WaypointNdbWeatherBroadcast = 0. Consequently, this variable may not represent an active feature in FS9.

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

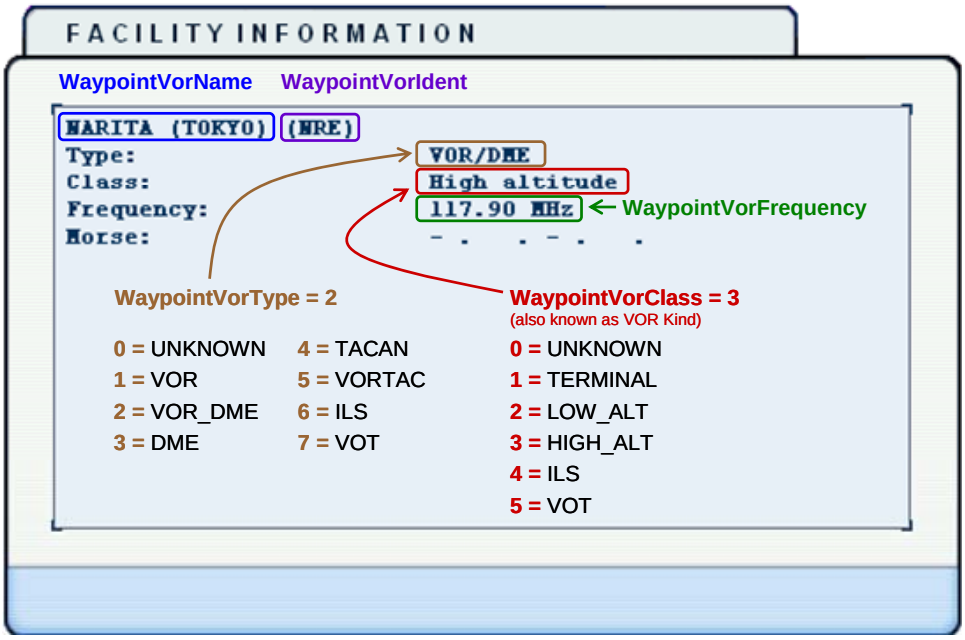
WaypointNdbMagneticVariation is the magnetic variation at the NDB location. Magnetic Variation is the difference between the compass magnetic indication and True North, measured in degrees longitude. This is an integer number in the fs9gps database.

WAYPOINT VOR GROUP

The WaypointVor Group contains all variables associated with specific VOR, VORTAC and VOR-DME beacons in the fs9gps database. The ICAO Identification must be specified before variables can be accessed, then all subsequent variables accessed in WaypointVor return information about that VOR until the ICAO is changed. ILS and LOC are not part of the WaypointVor Group even though the ICAO Type for ILS and LOC is 'V'. These two Nav facilities belong to the WaypointAirport Group.

All variables in WaypointVor are non-indexed; there are no 'lists' of items associated with a specific VOR (compared to WaypointAirport, where, for example, there are lists of different runways, approaches, transitions, and frequencies).

The figure below is a screen shot of the Facility Information page of an example VOR in FS9, indicating the associated gps variable names.



The following is a snapshot of all WaypointVor variables for the same VOR.

```
GPS Viewer 1.2                    11111111112
GET  SET  SLEN  12345678901234567890  STRING NUMBER
 G    S    12   VRJ      NRE          WaypointVorICAO, string
 G         3    NRE                   WaypointVorIdent, string
 G         2    2                     WaypointVorType, enum
 G         2    3                     WaypointVorClass, enum
 G         14   NARITA (TOKYO)        WaypointVorName, string
 G         0                          WaypointVorCity, string
 G         2    RJ                    WaypointVorRegion, string
 G         6      35.7823473          WaypointVorLatitude, degrees
 G         6     140.3625360          WaypointVorLongitude, degrees
 G         6    153.999351            WaypointVorElevation, feet
 G         10   117.900000            WaypointVorFrequency, mhz
 G         2    0                     WaypointVorWeatherBroadcast, enum
 G         2      7.00                WaypointVorMagneticVariation, degrees
```

Note that for the Narita VOR, the City is part of the WaypointVorName and is displayed in parenthesis (Tokyo).

❑ WaypointVorICAO (string, SLEN=12) [Get, Set]

WaypointVorICAO is the ICAO for the specific VOR.

❑ WaypointVorIdent (string) [Get]

The 2 to 3 character VOR Ident.

❑ WaypointVorType (enum) [Get]

A number representing VOR Type.

VOR Type

#	VOR Type	#	VOR Type
0	UNKNOWN = 0	4	TACAN = 4
1	VOR = 1	5	VORTAC = 5
2	VOR_DME = 2	6	ILS = 6
3	DME = 3	7	VOT = 7

http://msdn.microsoft.com/en-us/library/cc526954.aspx#VorType

❑ WaypointVorClass (enum) [Get]

A number representing VOR Class, also known as VOR Kind.

VOR Class (Kind)

#	VOR Class	#	VOR Class
0	UNKNOWN = 0	3	HIGH_ALT = 3
1	TERMINAL = 1	4	ILS = 4
2	LOW_ALT = 2	5	VOT = 5

http://msdn.microsoft.com/en-us/library/cc526954.aspx#VorKind

❑ WaypointVorName (string) [Get]

Name of the VOR. Interestingly, some VORs also contain the city name in parenthesis following the VOR name – all part of the variable WaypointVorName. I do not understand the rules/reasons that some do and some do not. In the example above, Narita is the VOR name, Tokyo is the city. VOR Names are not searchable using NameSearch.

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

WaypointVorCity is not populated in the fs9gps data base. It always returns a blank string.

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

The two character Region code.

- **WaypointVorLatitude**
- **WaypointVorLongitude (degrees or radians) [Get]**

The latitude and longitude of the VOR. 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).

- **WaypointVorElevation (feet) [Get]**

Elevation (asl) of the VOR facility.

- **WaypointVorFrequency (kHz) [Get]**

Radio frequency of the VOR. Commonly expressed in MHz.

- **WaypointVorWeatherBroadcast (gps boolean) [Get]**

The ESP SDK indicates that WaypointVorWeatherBroadcast is a gps boolean:

- 0 = Unknown
- 1 = No
- 2 = Yes

However, having scanned most VORs in the fs9gps database, so far I have found all VORs have WaypointVorWeatherBroadcast = 0. Consequently, this variable may not represent an active feature in FS9.

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

WaypointVorMagneticVariation is the magnetic variation at the VOR location. Magnetic Variation is the difference between the compass magnetic indication and True North, measured in degrees longitude. This is an integer number in the fs9gps database.

NEAREST AIRPORT GROUP

A NearestAirport search returns a list of airports nearest the reference point that is normally set using the current position of the aircraft. It sorts the data by ascending distance. In very large searches, ascending distance isn't strictly maintained, but in smaller searches up to 10 items or so, ascending order seems always to be maintained.

Common with all Nearest searches, not all available airport information can be obtained in a NearestAirport search, only the variables that begin with NearestAirport. If airport frequencies, runways, transitions, etc, are needed, then an ICAO transfer into WaypointAirport must be performed. Refer to the ICAO Transfer section.

- **NearestAirportCurrentLatitude**
- **NearestAirportCurrentLongitude (degrees, radians) [Get, Set]**

Latitude and Longitude of the reference point, usually the aircraft. Input is in degrees (decimal format, not deg, min, sec) or radians.

- **NearestAirportMaximumItems (enum) [Get, Set]**

The limit of numbers of items to be returned in the search.

- **NearestAirportMaximumDistance (enum) [Get, Set]**

Maximum search radius. 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.

- **NearestAirportItemsNumber (enum) [Get]**

The number of airports actually returned in the NearestAirport search.

- **NearestAirportCurrentLine (enum) [Get, Set]**

The Index pointer. Refer to the GPS Database Search section for further description.

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

The ICAO of each airport retrieved in the NearestAirport search.

❑ NearestAirportCurrentIdent (string) [Get]

The 3 to 4 character Ident of each airport retrieved in the NearestAirport search.

❑ NearestAirportCurrentAirportKind (enum) [Get]

A number representing Airport Class.

WaypointAirportKind (Class)

#	Class (Kind)	#	Class (Kind)
0	UNKNOWN_KIND_AIRPORT = 0	3	WATER_SURFACE_AIRPORT = 3
1	HARD_SURFACE_AIRPORT = 1	4	HELIPAD_AIRPORT = 4
2	SOFT_SURFACE_AIRPORT = 2	5	PRIVATE_AIRPORT = 5

http://msdn.microsoft.com/en-us/library/cc526954.aspx#AirportClass

❑ NearestAirportCurrentLongestAirportDirection (degrees) [Get]

Direction (magnetic) of the longest runway.

❑ NearestAirportCurrentDistance (nmiles) [Get]

The distance of each airport retrieved in the NearestAirport search from the reference point (NearestAirportCurrent Latitude and CurrentLongitude), which is normally set from the aircraft's current position.

❑ NearestAirportCurrentTrueBearing (degrees) [Get]

The bearing (true) from the reference point to each VOR retrieved in the NearestVor search.

❑ NearestAirportCurrentBestApproachEnum (enum) [Get]

A number representing the most precise approach available at the airport.

WaypointAirportBestApproach

#	Approach Type	#	Approach Type	#	Approach Type
0	UNKNOWN = 0	5	LORAN = 5	10	LDA = 10
1	VFR = 1	6	RNAV = 6	11	LOC = 11
2	HEL = 2	7	VOR = 7	12	MLS = 12
3	TACAN = 3	8	GPS = 8	13	ILS = 13
4	NDB = 4	9	SDF = 9		

http://msdn.microsoft.com/en-us/library/cc526954.aspx#AirportApproachType

❑ NearestAirportCurrentBestApproach (string) [Get]

The name of the most precise approach available at the airport. Refer to table above.

❑ NearestAirportCurrentComFrequencyName (string) [Get]

NearestAirportCurrentComFrequencyName is the abbreviation for the airport traffic control name if one is present at the airport. These include:

- ❑ twr = Tower
- ❑ CTF = Common Traffic Advisory Frequency (CTAF)
- ❑ uni = Unicomm
- ❑ mul = Multicomm

Ground, Clearance, Approach, Departure, ATIS, ASOS, AWOS, and FSS are not named by CurrentComFrequencyName. Those frequencies are available only from the WaypointAirport Group.

❑ NearestAirportCurrentComFrequencyValue (MHz) [Get]

The frequency value of CurrentComFrequencyName. If more than one tower is available at an airport, only the first frequency will be returned by CurrentComFrequencyValue. If no traffic control frequencies are available for the airport, CurrentComFrequencyValue returns 0.00.

```
NEAREST AIRPORT SEARCH

  49.3668 :Current Lat    15 :Max Items   15 :Items Num
 -97.1143 :Current Lon   100 :Max Distance

Current   ICAO      lll ---------------------- Current -------------------------
Line      123456789012 Ident Kind Rwy*  Dist  Brg Best Appr  Com     Freq Length
0         A       CKK7   CKK7    1  176  18.0   64    0        uni  122.80   3000
1         A       CJB3   CJB3    1  149  20.2   57    0        mul  122.70   3000
2         A       CJL6   CJL6    1  180  22.9  226    0        mul  123.20   3248
3         A       KA4     KA4    2  126  24.5  102    0             0.00   2875
4         A       KPMB   KPMB    1  160  25.9  191    7  VOR   CTF  122.80   3797
5         A       NA67   NA67    2  177  26.8  211    0             0.00   2000
6         A       CJL5   CJL5    2   96  29.7   11    0        mul  123.20   3000
7         A       8NA6   8NA6    2   87  30.5  214    0             0.00   3800
8         A       CKJ7   CKJ7    2  137  30.5  314    0        mul  123.40   3000
9         A       5ND3   5ND3    2  156  31.7  195    0             0.00   1600
10        A       CYWG   CYWG    1  188  32.9  352   13  ILS   twr  118.30  10988
11        A       CKZ7   CKZ7    1   90  33.7  250    0        uni  122.80   2900
12        A       MN57   MN57    2   96  35.3  168    0             0.00   2500
13        A       KHC0   KHC0    1  140  37.5  170    8  GPS   CTF  122.80   4002
14        A       CKA8   CKA8    2   17  37.5  333    0             0.00   3000
```

❑ NearestAirportCurrentLongestRunwayLength (feet) [Get]

Length of the longest runway at the airport.

NEAREST INTERSECTION GROUP

A NearestIntersection search returns a list of Intersections nearest the reference point that is normally set from the current aircraft position. It sorts the data by ascending distance. In very large searches, ascending distance isn't strictly maintained, but in smaller searches up to 10 items or so, ascending order seems always to be maintained.

Common with all Nearest searches, not all available Intersection information can be obtained in a NearestIntersection search, only the variables that begin with NearestIntersection. If Region, Nearest VOR Ident, Nearest VOR Type, Nearest VOR True Radial, Nearest VOR Magnetic Radial, or Nearest VOR Distance from an intersection following a NearestIntersection search is needed, then an ICAO transfer into WaypointIntersection must be performed. Refer to the ICAO Transfer section.

❑ NearestIntersectionCurrentLatitude (degrees, radians) [Get, Set]

Latitude of the reference point, usually the aircraft. Input is in degrees (decimal format, not deg, min, sec) or radians.

❑ NearestIntersectionCurrentLongitude (degrees, radians) [Get, Set]

Longitude of the reference point, usually the aircraft. Input is in degrees (decimal format, not deg, min, sec) or radians.

❑ NearestIntersectionMaximumItems (enum) [Get, Set]

The limit of numbers of items to be returned in the search. In practice, this should be kept realistically small so the NeasrestIntersection search returns data quickly. The gps_500 gauge, for example, sets this value to 9.

❑ NearestIntersectionMaximumDistance (enum) [Get, Set]

Maximum search radius. 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.

❑ NearestIntersectionCurrentFilter (enum) [Get, Set]

NearestIntersectionCurrentFilter is a number between 0 and 255 that is the decimal equivalent of the 8 bit binary number that indicates which of the 8 Intersection types are to be included in the NearestIntersection search.

The Intersection types for FSX, and presumably also for FS9 are:

Bit	Name and Type #	Bit	Name and Type #
0	UNKNOWN = 0	4	NDB = 4
1	NAMED = 1	5	OFFROUTE = 5
2	UNNAMED = 2	6	IAF = 6
3	VOR = 3	7	FAF = 7

http://msdn.microsoft.com/en-us/library/cc526954.aspx#NearestIntersectionData

The default fs9gps NearestIntersectionCurrentFilter value is 230. The binary equivalent is*:

```
7 . . 4  3 . . 0 – Bit number (Bit 0 thru Bit 7 = 8 Bits)
1 1 1 0  0 1 1 0
```

which includes Intersection types 1, 2, 5, 6, and 7. These are **NAMED**, **UNNAMED**, **OFFROUTE**, **IAF**, and **FAF** intersections. That is, all Intersection types except **VOR** and **NDB**.

❑ NearestIntersectionAddIntersectionType (enum) [Set]

NearestIntersectionAddIntersectionType is an enum (not a binary) that adds an Intersection type to the NearestIntersection search.

❑ NearestIntersectionRemoveIntersectionType (enum) [Set]

NearestIntersectionRemoveIntersectionType is an enum (not a binary) that removes an Intersection type from the NearestIntersection search.

❑ NearestIntersectionSetDefaultFilter (enum) [Set]

NearestIntersectionSetDefaultFilter returns the NearestIntersection search to the default value CurrentFilter = 230. The proper syntax is:

```
(>C:fs9gps:NearestIntersectionSetDefaultFilter)
```

Note that an argument is not required, however you can include anything you want as log as the '>' is included.

EXAMPLE: NearestIntersection Add, Remove, and SetDefault

To begin with, an example search result using the default filter setting:

```
NEAREST INTERSECTION SEARCH

  37.6625 :Curr Lat  10 :Max Items  10 :Items Num
 -95.4875 :Curr Lon  50 :Max Dist  230 :Filter

---------- NearestIntersectionCurrent ----------
          ICAO     111
Line      123456789012    Ident  Type   Dist   Brg
0         WK3KCNUFLOUR    FLOUR     1    0.0   187
1         WK3KCNUMACEZ    MACEZ     1    0.3   331
2         WK3    BABGE    BABGE     1    3.4   162
3         WK3KCNUMYYER    MYYER     1    5.0   187
4         WK3K88 ABUYA    ABUYA     1    8.1    26
5         WK3KPPFAGNEW    AGNEW     1    9.5   186
6         WK3KPPFFF17      FF17     2   14.5   184
7         WK3KPPFDENUM    DENUM     1   16.0   182
8         WK3K88 SAVOF    SAVOF     1   17.7    19
9         WK3KPPFBATTR    BATTR     1   17.9   194
```

A few points of interest in the 12 character ICAO identifier include:

- The first character of the 12 character ICAO identifier, ICAO Type, is "W" for Intersection types 1 and 2 = NAMED and UNNAMED (Waypoint) Intersections, "V" for Intersection type 3 = VOR Intersection, and "N" for Intersection type 4 = NDB Intersection.
- For Type 1 NAMED Intersections, the Intersection Idents are 5 letters long (character positions 8 through 12) and are often geographically recognizable words.
- Some Intersection ICAOs contain an Airport Ident listed in character positions 4 through 7. These are Terminal Waypoints and are displayed with blue intersection symbols △ on the FS9 map. Terminal Waypoints are part of fs9gps terminal procedures or part of approaches or departures towards and away from a runway. The ident of the airport that "owns" Terminal Waypoint is included in the ICAO. The ICAOs without an Airport Ident are enroute intersections generally used to create Victor Airway and Jet Airway routes. These are the magenta colored intersections △ on the FS9 map.

The fs9gps database appears to be populated with only 4 types of Intersections: Type 1 = NAMED, Type 2 = UNNAMED, Type 3 = VOR, and Type 4 = NDB. If correct, then including types 5, 6, and 7 is irrelevant.

(*) Binary numbers are read from right to left, the right-most digit is always Bit 0, the digit to its left is Bit 1, and so forth.

EXAMPLE: NearestIntersectionAddIntersectionType

To add Intersection Type 3 (VOR Intersection) to the search, the following xml is used:

```
<Update>
3 (>C:fs9gps:NearestIntersectionAddIntersectionType)
</Update>
```

which yields these search results:

```
NEAREST INTERSECTION SEARCH

  37.6625 :Curr Lat  10 :Max Items  10 :Items Num
 -95.4875 :Curr Lon  50 :Max Dist 238 :Filter

---------- NearestIntersectionCurrent ----------
          ICAO     111
Line      123456789012     Ident  Type   Dist   Brg
0         WK3KCNUFLOUR     FLOUR     1    0.0   187
1         WK3KCNUMACEZ     MACEZ     1    0.3   331
2         WK3     BABGE    BABGE     1    3.4   162
3         WK3KCNUMYYER     MYYER     1    5.0   187
4         VK3     CNU        CNU     3    5.5   247
5         WK3K88 ABUYA     ABUYA     1    8.1    26
6         WK3KPPFAGNEW     AGNEW     1    9.5   186
7         WK3KPPFFF17       FF17     2   14.5   184
8         WK3KPPFDENUM     DENUM     1   16.0   182
9         WK3K88 SAVOF     SAVOF     1   17.7    19
```

The VOR Intersection 'CNU' (Line 4) is in the search results and the filter value is 238.

EXAMPLE: NearestIntersectionRemoveIntersectionType

To subsequently remove Intersection Type 1 (Named Intersection) from the search, the xml would be:

```
<Update>
3 (>C:fs9gps:NearestIntersectionAddIntersectionType)
1 (>C:fs9gps:NearestIntersectionRemoveIntersectionType)
</Update>
```

which yields these search results:

```
NEAREST INTERSECTION SEARCH

  37.6625 :Curr Lat   10 :Max Items   10 :Items Num
 -95.4875 :Curr Lon   50 :Max Dist  236 :Filter

----------- NearestIntersectionCurrent ----------
          ICAO     111
Line      123456789012     Ident  Type   Dist   Brg
0         VK3     CNU        CNU     3    5.5   247
1         WK3KPPFFF17       FF17     2   14.5   184
2         WK3KPPFMA118     MA118     2   19.7   183
3         WK3KPPFFF35       FF35     2   25.5   182
4         WK3KJLNJLN31     JLN31     2   27.6   106
5         VK3     OSW        OSW     3   33.2   156
6         WK3KIDPMA023     MA023     2   33.7   205
7         WK3KUKLFF36       FF36     2   36.3   341
8         WK3KPTSFF166     FF166     2   36.7   104
9         WK3KCFVFF35       FF35     2   39.8   186
```

Intersection Type 1 has been removed and the filter value is 236.

EXAMPLE: NearestIntersectionSetDefaultFilter

If you wish to now reset the filter to the default setting, the xml would be:

```
<Update>
3 (>C:fs9gps:NearestIntersectionAddIntersectionType)
1 (>C:fs9gps:NearestIntersectionRemoveIntersectionType)
(>C:fs9gps:NearestIntersectionSetDefaultFilter)
</Update>
```

and the search results return to the default filter value 230 setting:

```
NEAREST INTERSECTION SEARCH

  37.6625 :Curr Lat   10 :Max Items   10 :Items Num
 -95.4875 :Curr Lon   50 :Max Dist  230 :Filter

----------- NearestIntersectionCurrent ----------
          ICAO     111
Line      123456789012     Ident  Type   Dist   Brg
0         WK3KCNUFLOUR     FLOUR     1    0.0   187
1         WK3KCNUMACEZ     MACEZ     1    0.3   331
2         WK3     BABGE    BABGE     1    3.4   162
3         WK3KCNUMYYER     MYYER     1    5.0   187
4         WK3K88 ABUYA     ABUYA     1    8.1    26
5         WK3KPPFAGNEW     AGNEW     1    9.5   186
6         WK3KPPFFF17       FF17     2   14.5   184
7         WK3KPPFDENUM     DENUM     1   16.0   182
8         WK3K88 SAVOF     SAVOF     1   17.7    19
9         WK3KPPFBATTR     BATTR     1   17.9   194
```

❑ NearestIntersectionItemsNumber (enum) [Get]

The number of Intersections actually returned in the NearestIntersection search.

❑ NearestIntersectionCurrentLine (enum) [Get, Set]

The Index pointer. Refer to the GPS Database Search section for further description.

❑ NearestIntersectionCurrentICAO (string) [Get]

The ICAO of each Intersection retrieved in the NearestIntersection search.

❑ NearestIntersectionCurrentIdent (string) [Get]

The 1 to 5 character Ident of each Intersection retrieved in the NearestIntersection search.

❑ NearestIntersectionCurrentType (enum) [Get]

The type of each Intersection retrieved in the NearestIntersection search. See NearestIntersectionCurrentFilter for additional discussion.

❑ NearestIntersectionCurrentDistance (nmiles or meters) [Get]

The distance of each Intersection retrieved in the NearestIntersection search from the reference point (NearestIntersectionCurrent Latitude and CurrentLongitude), which is normally set from the aircraft's current position.

❑ NearestIntersectionCurrentTrueBearing (degrees or radians) [Get]

The bearing (true) from the reference point to each Intersection retrieved in the NearestIntersection search.

NEAREST VOR GROUP

A NearestVor search returns a list of VORs nearest the reference point that is normally set using the current position of the aircraft. It sorts the data by ascending distance. In very large searches, ascending distance isn't strictly maintained, but in smaller searches up to 10 items or so, ascending order seems always to be maintained.

Common with all FS9 Nearest searches, not all available VOR information can be obtained in a NearestVor search, only the variables that begin with NearestVor. If Class, Name, Region, Elevation, Weather Broadcast, or Magnetic Variation of a VOR from a NearestVor search is needed, then an ICAO transfer into WaypointVor must be performed. Refer to the ICAO Transfer section. This is one area in which the gps module in FSX is much easier to work with.

- **NearestVorCurrentLatitude**
- **NearestVorCurrentLongitude (degrees, radians) [Get, Set]**

Latitude and Longitude of the reference point, usually the aircraft. Input is in degrees (decimal format, not deg, min, sec) or radians.

- **NearestVorMaximumItems (enum) [Get, Set]**

The limit of numbers of items to be returned in the search.

- **NearestVorMaximumDistance (enum) [Get, Set]**

Maximum search radius. 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.

- **NearestVorCurrentFilter (enum) [Get, Set]**

For the fs9gps database, NearestVorCurrentFilter is a number between 0 and 63 that is the decimal equivalent of the 6 bit binary number that indicates which of the 6 FS9 VOR types are included in the NearestVor search.

I cannot locate documentation that lists the 6 VOR types, however, Microsoft's ESP SDK lists 8 types of VORs (FSX):

Bit	Name and Type #	Bit	Name and Type #
0	UNKNOWN = 0	4	TACAN = 4
1	VOR = 1	5	VORTAC = 5
2	VOR_DME = 2	6	ILS = 6
3	DME = 3	7	VOT = 7

http://msdn.microsoft.com/en-us/library/cc526954.aspx#VorType

The assumption is that the VOR types for FS9 are:

Bit	Name and Type #	Bit	Name and Type #
0	UNKNOWN = 0	3	DME = 3
1	VOR = 1	4	TACAN = 4
2	VOR_DME = 2	5	VORTAC = 5

The default fs9gps NearestVorCurrentFilter value is 62. The binary equivalent is*:

```
5 . . . . 0 – Bit number (Bit 0 thru Bit 5 = 6 Bits)
1 1 1 1 1 0
```

which includes all VOR types except UNKNOWN.

To include only VOR types 1, 2, and 3 in the NearestVor search, the binary would be*:

```
5 . . . . 0 – Bit number (Bit 0 thru Bit 5 = 6 Bits)
0 0 1 1 1 0
```

the decimal equivalent of which is 14. The xml instruction to set the filter would be:

```
14 (>C:fs9gps:NearestVorCurrentFilter)
```

However, the fs9gps database appears to be populated with only 3 types of VORs: Type 1 = VOR, Type 2 = VOR_DME, and Type 3 = DME, so the identities of types 4 and 5 are irrelevant.

* Binary numbers are read from right to left, the right-most digit is always Bit 0, the digit to its left is Bit 1, and so forth.

❑ NearestVorAddVorType (enum) [Set]

NearestVorAddVorType is an enum (not a binary) adds a VOR type to the NearestVor search. If the <Update> section contains the following statement:

```
<Update>
10 (>C:fs9gps:NearestVorCurrentFilter)
</Update>
```

then only VOR types 1 and 3 (decimal 10 = binary **001010**) will be included in the NearestVor search, and a typical search result would look like:

```
NEAREST VOR SEARCH

  42.9638 :Current Lat   20 :Max Items  11 :Items Num
 -88.9091 :Current Lon  300 :Max Dist   10 :Filter

----------------- NearestVorCurrent -----------------
      ICAO     111
Line  123456789012  Ident Type     Freq    Dist   Brg
0     VK5     BJB      BJB    1   109.80    43.9    51
1     VK5     VOK      VOK    3   110.40    83.1   315
2     VK5     OLK      OLK    1   110.40   183.5   123
3     VK5     CGG      CGG    1   109.80   208.2    59
4     VK5     TOL      TOL    3   112.50   241.2   108
5     VK5     AOH      AOH    1   108.40   259.0   120
6     VK3     EST      EST    1   110.40   256.7   278
7     VK5     MAH      MAH    1   114.90   261.4   115
8     VK3     ULM      ULM    3   111.30   255.9   290
9     VK5     SKE      SKE    3   112.20   268.6   189
10    VK5     BUD      BUD    1   109.80   296.9   116
```

But, if the <Update> section is further edited to include a NearestVorAddVorType instruction, the NearestVor search changes:

```
<Update>
10 (>C:fs9gps:NearestVorCurrentFilter)
2 (>C:fs9gps:NearestVorAddVorType)
</Update>
```

```
NEAREST VOR SEARCH

  42.9638 :Current Lat   20 :Max Items  20 :Items Num
 -88.9091 :Current Lon  300 :Max Dist   14 :Filter

----------------- NearestVorCurrent -----------------
      ICAO     111
Line  123456789012  Ident Type     Freq     Dist   Brg
0     VK5     MSN      MSN    2   108.60     21.8   300
1     VK5     JVL      JVL    2   114.30     25.8   200
2     VK5     BAE      BAE    2   116.40     28.9    71
3     VK5     BUU      BUU    2   114.50     31.4   121
4     VK5     LJT      LJT    2   112.50     39.2    77
5     VK5     BJB      BJB    1   109.80     43.9    51
6     VK5     RFD      RFD    2   110.80     46.1   196
7     VK5     ENW      ENW    2   109.20     48.3   117
8     VK5     HRK      HRK    2   117.70     49.6   104
9     VK5     DLL      DLL    2   117.00     51.3   314
10    VK5     LNR      LNR    2   112.80     57.2   291
11    VK5     OBK      OBK    2   113.00     61.4   136
12    VK5     OSH      OSH    2   111.80     63.5    14
13    VK5     PLL      PLL    2   111.20     65.8   205
14    VK5     FAH      FAH    2   110.00     66.9    43
15    VK5     DPA      DPA    2   108.40     69.0   159
16    VK5     ORD      ORD    2   113.90     73.5   142
17    VK5     VOK      VOK    3   110.40     83.1   315
18    VK3     DBQ      DBQ    2   115.80     86.3   248
19    VK5     MTW      MTW    2   111.00     88.0    37
```

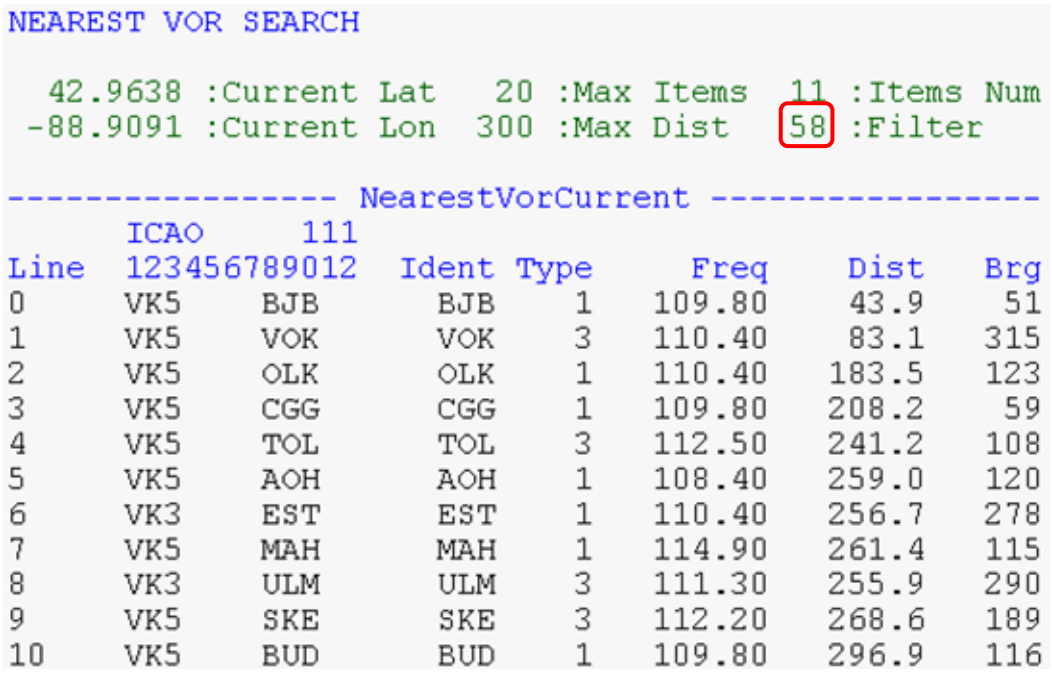
Now, VOR Type 2 has been added, the Filter value becomes 14 (binary **001110**, VOR types 1, 2, and 3), and additional VORs have been found within the 300 mile search radius.

❑ NearestVorRemoveVorType (enum) [Set]

NearestVorRemoveVorType functions in the opposite manner from AddVorType. If, for example, no NearestVorCurrentFilter instruction is given, the default CurrentFilter = 62 is assumed and all VOR types except UNKNOWN are included in the NearestVor search. The following will remove VOR Type =2 from the NearestVor search:

```
<Update>
2 (>C:fs9gps:NearestVorRemoveVorType)
</Update>
```

and the search result becomes:



```
NEAREST VOR SEARCH

  42.9638 :Current Lat   20 :Max Items  11 :Items Num
 -88.9091 :Current Lon  300 :Max Dist   58 :Filter

----------------- NearestVorCurrent -----------------
      ICAO     111
Line  123456789012  Ident Type     Freq    Dist   Brg
0     VK5    BJB      BJB    1   109.80    43.9    51
1     VK5    VOK      VOK    3   110.40    83.1   315
2     VK5    OLK      OLK    1   110.40   183.5   123
3     VK5    CGG      CGG    1   109.80   208.2    59
4     VK5    TOL      TOL    3   112.50   241.2   108
5     VK5    AOH      AOH    1   108.40   259.0   120
6     VK3    EST      EST    1   110.40   256.7   278
7     VK5    MAH      MAH    1   114.90   261.4   115
8     VK3    ULM      ULM    3   111.30   255.9   290
9     VK5    SKE      SKE    3   112.20   268.6   189
10    VK5    BUD      BUD    1   109.80   296.9   116
```

Only VOR types 1, 3, 4 and 5 are included in the search, and the Filter, which had been the default value 62, is now 58, binary **`111010`** (the fs9gps database appears to be populated with no Type 4 or 5 VORs, so none can be found in a search).

❑ NearestVorSetDefaultFilter (enum) [Set]

NearestVorSetDefaultFilter returns the NearestVor search to the default value CurrentFilter = 62. The proper syntax is:

```
(>C:fs9gps:NearestVorSetDefaultFilter)
```

Note that a value apparently is not required, however you can include anything you want. Whatever is already in the stack will be fine; a negative number, zero, positive number, decimal…

To recap:

```
<Update>
14 (>C:fs9gps:NearestVorCurrentVorType)
</Update>
```

sets the CurrentFilter to 14 (binary **`001110`**) and VOR types 1, 2, and 3 are included in the NearestVor search.

```
<Update>
14 (>C:fs9gps:NearestVorCurrentVorType)
2 (>C:fs9gps:NearestVorRemoveVorType)
</Update>
```

removes VOR Type 2, CurrentFilter becomes 10 (binary **001010**), and VOR types 1 and 3 are included in the NearestVor search.

```
<Update>
14 (>C:fs9gps:NearestVorCurrentVorType)
2 (>C:fs9gps:NearestVorRemoveVorType)
(>C:fs9gps:NearestVorSetDefaultFilter)
</Update>
```

resets CurrentFilter to 62 (binary **111110**) and all VOR types except UNKNOWN are included in the NearestVor search. The following SetDefaultFilter statements would all have done the same thing:

```
1 (>C:fs9gps:NearestVorSetDefaultFilter)
0 (>C:fs9gps:NearestVorSetDefaultFilter)
-5 (>C:fs9gps:NearestVorSetDefaultFilter)
12.378 (>C:fs9gps:NearestVorSetDefaultFilter)
```

but,

```
(C:fs9gps:NearestVorSetDefaultFilter)
```

will not work.

❑ NearestVorItemsNumber (enum) [Get]

The number of VORs actually returned in the NearestVor search.

❑ NearestVorCurrentLine (enum) [Get, Set]

The Index pointer. Refer to the GPS Database Search section for further description.

❑ NearestVorCurrentICAO (string) [Get]

The ICAO of each VOR retrieved in the NearestVor search.

❑ NearestVorCurrentIdent (string) [Get]

The 1 to 3 character Ident of each VOR retrieved in the NearestVor search.

❑ NearestVorCurrentType (enum) [Get]

The type of each VOR retrieved in the NearestVor search. It appears that only Type 1 = VOR, Type 2 = VOR DME, and Type 3 = DME exist in the fs9gps database. See **NearestVorCurrentFilter** for additional discussion.

❑ NearestVorCurrentFrequency (MHz) [Get]

The frequency of each VOR retrieved in the NearestVor search.

❑ NearestVorCurrentDistance (nmiles or meters) [Get]

The distance of each VOR retrieved in the NearestVor search from the reference point (NearestVorCurrent Latitude and CurrentLongitude), which is normally set from the aircraft's current position.

❑ NearestVorCurrentTrueBearing (degrees) [Get]

The bearing (true) from the reference point to each VOR retrieved in the NearestVor search.

NEAREST NDB GROUP

A NearestNdb search returns a list of NDBs nearest the reference point that is normally set from the current aircraft position. It sorts the data by ascending distance. In very large searches, ascending distance isn't strictly maintained, but in smaller searches up to 10 items or so, ascending order seems always to be maintained.

Common with all Nearest searches, not all available NDB information can be obtained in a NearestNdb search, only the variables that begin with NearestNdb. If Name, Region, Elevation, Weather Broadcast, or Magnetic Variation of a specific NDB found in a NearestNdb search is needed, then an ICAO transfer into WaypointNdb must be performed. Refer to the ICAO Transfer section.

❑ NearestNdbCurrentLatitude (degrees, radians) [Get, Set]

Latitude of the reference point, usually the aircraft. Input is in degrees (decimal format, not deg, min, sec) or radians.

❑ NearestNdbCurrentLongitude (degrees, radians) [Get, Set]

Longitude of the reference point, usually the aircraft. Input is in degrees (decimal format, not deg, min, sec) or radians.

❑ NearestNdbMaximumItems (enum) [Get, Set]

The limit of numbers of items to be returned in the search. In practice, this should be kept realistically small so the NeasrestNdb search returns data quickly. The gps_500 gauge, for example, sets this value to 9.

❑ NearestNdbMaximumDistance (enum) [Get, Set]

Maximum search radius. 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.

❑ NearestNdbItemsNumber (enum) [Get]

The number of NDBs actually returned in the NearestNdb search.

❑ NearestNdbCurrentLine (enum) [Get, Set]

The Index pointer. Refer to the GPS Database Search section for further description.

❑ NearestNdbCurrentICAO (string) [Get]

The ICAO of each NDB retrieved in the NearestNdb search.

❑ NearestNdbCurrentIdent (string) [Get]

The 1 to 5 character Ident of each NDB retrieved in the NearestNdb search.

❑ NearestNdbCurrentType (enum) [Get]

The type of each NDB retrieved in the NearestNdb search.

The following is a list of NDB Type and Class, real life Transmission Power, real life Effective Range (which may not match how it is modeled in FS9):

0: Unknown. There appear to be no Type 0 NDBs in the fs9gps database.

1: Compass Locator. Below 25 watts, 15 - 25 nmiles. Type 1 NDBs are absent within the U.S.A. in the fs9gps database, but are common in other parts of the world, especially Europe (eg, U.K.).

2: MH. Below 50 watts, 25 - 50 nmiles. Directional Beacon Approach Facility found at or near airports where it is the primary approach aid. This is the most common type of NDB in the fs9gps database.

3: H. 50 to 1,999 watts, 50 - 75 nmiles. Enroute Airway Beacon, common in Canada and Caribbean

4: HH. 2,000+ watts, 75 - 125 nmiles. High powered Beacon found along coasts in the U.S.A.

An example of a NearestNdb search:

```
NEAREST NDB SEARCH

  52.7392 :Curr Lat    50 :Max Items    12 :Items Num
  -0.0070 :Curr Lon    50 :Max Distance

---------------- NearestVorCurrent ----------------
         ICAO     111
Line     123456789012  Ident  Type  Freq   Dist  Brg
0        NEG     FNL      FNL      4   401    0.8  281
1        NEG     CWL      CWL      2   423   24.5  315
2        NEG     BOU      BOU      2   392   31.7  183
3        NEG     CAM      CAM      2   333   32.5  168
4        NEG     LE        LE      2   384   38.2  258
5        NEG     NN        NN      2   379   39.1  228
6        NEG     NOT      NOT      2   430   40.4  286
7        NEGEGTCCIT       CIT      2   850   41.7  209
8        NEG     CIT      CIT      2   850   41.7  209
9        NEG     EME      EME      2   354   43.5  278
10       NEG     NWI      NWI      1   343   47.4   94
11       NEG     CM        CM      3   314   49.3   76
```

The "CIT" NDB (CurrentLine 7) contains the "EGTC" airport ident in character positions 4 through 7. This NDB serves as a Terminal Waypoint in a fs9gps approach procedure. The airport EGTC is the "owner" of the waypoint, consequently, its ident is included in the NDB ICAO.

❑ NearestVorCurrentFrequency (kHz) [Get]

The frequency of each NDB retrieved in the NearestNdb search, usually expressed in KhZ units.

❑ NearestNdbCurrentDistance (nmiles) [Get]

The distance of each NDB retrieved in the NearestNdb search from the reference point (NearestNdbCurrent Latitude and CurrentLongitude). The reference point is normally set from the aircraft's current position.

❑ NearestNdbCurrentTrueBearing (degrees or radians) [Get]

The bearing (true) from the reference point to each NDB retrieved in the NearestNdb search.

NEAREST AIRSPACE GROUP

All Nearest searches require the latitude and longitude of the reference point, also known as the Current point, which is normally the aircraft location, plus specified limitations to the amount of data you want to be returned in the search: maximum search distance (radius) and maximum number of returned items.

Nearest Airspace searches, however, require more information to define the search than Nearest Airport, Intersection, VOR, and NDB searches because an airspace is a three dimensional shape rather than a one dimensional point. Another feature of Nearest Airspace searches is that some gauges that make use of NearestAirspace, like the stock MSFS gps_500 gauge, issue messages to the pilot when the aircraft is simply *near* an airspace. Consequently, "closeness" to an airspace must also be defined.

Required information for a NearestAirspace search includes CurrentLatitude, CurrentLongitude, CurrentAltitude, TrueGroundTrack, GroundSpeed, NearDistance, NearAltitude, AheadTime, Query, MaximumItems and MaximumDistance. The first 5 are constantly changing in flight and are usually input via an A: variable. The last 6 would usually not be changed during flight, and are input using standard value declarations in your code.

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

The location of the reference point, expressed as latitude and longitude. The units of Lat/Lon can be degrees (formatted +/-ddd.dddd where S16 degrees 30 minutes would be formatted as -16.5000) or radians (d.dddd). In most applications, the reference point for Nearest searches is the current aircraft location.

(A:PLANE LATITUDE, Radians) (>@c:NearestAirspaceCurrentLatitude, Radians)
(A:PLANE LONGITUDE, Radians) (>@c:NearestAirspaceCurrentLongitude, Radians)

Some people prefer use of A:PLANE LATITUDE / LONGITUDE rather than A:GPS POSITION LAT / LON because A:PLANE is updated every gauge update cycle whereas A:GPS is updated every one second (referring to time, as in 1/60th of a minute).

- **NearestAirspaceCurrentAltitude (feet) [Get, Set]**

Altitude (ASL) of the reference point, which is normally the aircraft. Common units are feet or meters.

```
(A:GPS POSITION ALT, feet)
(>@c:NearestAirspaceCurrentAltitude, feet)
```

Of course, `(A:PLANE ALTITUDE, feet)` works also.

❑ NearestAirspaceTrueGroundTrack (degrees) [Get, Set]

Ground track of the aircraft relative to true north.

```
(A:GPS GROUND TRUE TRACK, degrees)
(>@c:NearestAirspaceTrueGroundTrack, degrees)
```

❑ NearestAirspaceGroundSpeed (knots) [Get, Set]

Ground speed of the aircraft.

```
(A:GPS GROUND SPEED, knots)
(>@c:NearestAirspaceGroundSpeed, knots)
```

❑ NearestAirspaceNearDistance (nmiles) [Get, Set]

NearestAirspaceNearDistance is the horizontal distance between the aircraft and an airspace boundary at which point NearestAirspaceCurrentStatus changes and airspace encroachment messages can be issued. It is discussed more completely in the NearestAirspaceCurrentStatus section later in this chapter.

The default is 2 nmiles.

❑ NearestAirspaceNearAltitude (feet) [Get, Set]

NearestAirspaceNearAltitude is a vertical distance buffer applied to the current aircraft altitude such that, if the aircraft is within + / - NearAltitude of an airspace floor or ceiling, the NearestAirspaceCurrentStatus changes and airspace encroachment messages can be issued. It is discussed more completely in the NearestAirspaceCurrentStatus section later in this chapter.

The default is 200 feet.

❑ NearestAirspaceAheadTime (minutes) [Get, Set]

NearestAirspaceAheadTime is the time separation between the aircraft and an airspace boundary at which point NearestAirspaceCurrentStatus changes and airspace encroachment messages can be issued. It is computed using NearestAirspaceGroundSpeed and is discussed more completely in the NearestAirspaceCurrentStatus section later in this chapter.

The default is 10 minutes.

❑ NearestAirspaceQuery (6 digit Hexadecimal 'enum') [Get, Set]

NearestAirspaceQuery tells the gps.dll which type(s) of airspaces to include in the NearestAirspace search. This is somewhat analogous to IcaoSearchStartCursor which tells the gps.dll which types of facilities to include in an IcaoSearch.

NearestAirspaceQuery is expressed in Hexadecimal format to more easily define the types of airspaces to include. It is a six digit hex number which represents 24 bits of information (6 hex digits x 4 = 24 bits). Since each bit is a 1 or 0, it functions as an individual 'include' / 'do not include' switch. Each of the bits is mapped to a specific airspace type. If the bit corresponding to Class C airspace (Bit 4) is set to 1, then the NearestAirspace search will include Class C airspaces, otherwise it will not. Any combination of airspace types can be searched by setting the right bits.

On line 83 of the gps_500 xml gauge there is a parameter declaration named kDisplayedAirspaces that is assigned the hex value **0xEFC038**. Converting this hex number to binary yields:

```
2..2 ...1 ...1
3..0 ...6 ...2 ...8 ...4 ...0 – Bit number (Bit 0 thru Bit 23 = 24 Bits)
1110 1111 1100 0000 0011 1000
```

which contains 12 "1s", meaning that there are 12 airspace types included in kDisplayedAirspaces.

The complete list of airspace types (24 in total) is:

Airspace Bit Map Table

Bit	Name and Type #	Bit	Name and Type #	Bit	Name and Type #
0	NONE = 0	8	CLASS_G = 8	16	PROHIBITED = 16
1	CENTER = 1	9	TOWER = 9	17	WARNING = 17
2	CLASS_A = 2	10	CLEARANCE = 10	18	ALERT = 18
3	CLASS_B = 3	11	GROUND = 11	19	DANGER = 19
4	CLASS_C = 4	12	DEPARTURE = 12	20	NATIONAL_PARK = 20
5	CLASS_D = 5	13	APPROACH = 13	21	MODE_C = 21
6	CLASS_E = 6	14	MOA = 14	22	RADAR = 22
7	CLASS_F = 7	15	RESTRICTED = 15	23	TRAINING = 23

http://msdn.microsoft.com/en-us/library/cc526954.aspx#FAC_BV_TYPE

Therefore, reading **1110 1111 1100 0000 0011 1000** from *right to left,* (the right-most digit is always Bit 0, the digit to its left is Bit 1, and so forth) and comparing each Bit to the Airspace Bit Table, above, the NearestAirspace search of kDisplayedAirspaces will include **CLASS_B, CLASS_C, CLASS_D, MOA, RESTRICTED, PROHIBITED, WARNING, ALERT, DANGER, MODE_C, RADAR,** and **TRAINING** airspace types. In fact, note that these are listed in the comment line (line 82) directly above the kDisplayedAirspaces declaration.

The other NearestAirspaceQuery listed in the gps_500 gauge is kAlwaysDisplayedAirspaces = **0x0FC000** (line 85). Its binary equivalent is:

```
2..2 ...1 ...1
3..0 ...6 ...2 ...8 ...4 ...0 – Bit number
0000 1111 1100 0000 0000 0000
```

which means that kAlwaysDisplayedAirspaces includes **MOA**, **RESTRICTED**, **PROHIBITED**, **WARNING**, **ALERT**, and **DANGER** airspace types.

To include any combination of airspaces, create a binary number by indicating 1 or 0 (include or don't include) for each of the 24 airspace types. Start with Airspace Type 0 (Bit 0) and work up the list to Airspace Type 23 (Bit 23), building the number from right to left. Once a 24 digit number is assembled from this selection process, convert it to hexadecimal format and enter that number into NearestAirspaceQuery.

If you wanted to include **TRAINING**, **RADAR**, **DANGER**, **ALERT**, **WARNING**, **MOA**, and **CLASS_C** airspaces in a NearestAirspace search, the binary would be:

```
2..2 ...1 ...1
3..0 ...6 ...2 ...8 ...4 ...0 – Bit number
1100 1110 0100 0000 0001 0000
```

the hexadecimal equivalent of which is **0xCE4010**. The proper xml instruction is:

```
0xCE4010 (>C:fs9gps:NearestAirspaceQuery)
```

It is acceptable to enter the decimal equivalent instead, but that approach may not make as much sense given the binary origin of the number:

```
13516816 (>C:fs9gps:NearestAirspaceQuery)
```

❑ NearestAirspaceMaximumItems (enum) [Get, Set]

Maximum number of airspace sectors that will be included in the search results. After this number of items is reached, the search process terminates.

```
9 (>@c:NearestAirspaceMaximumItems)
```

In the gps_500 gauge, "9" is used for maximum search items, matching the capability of the Garmin GNS 500 / 530 / 530A system after which it is modeled. When a Nearest search concludes, the list of 9 items (Airports, Intersections, VORs, NDBs, or Airspaces) is displayed using a {loop} within a <String> expression, with the Nearest being at the top.

❑ NearestAirspaceMaximumDistance (nmiles, meters) [Get, Set]

The maximum distance (radius) the search will extend from the reference point. If the search reaches this limit, it will terminate and Airspaces beyond MaximumDistance will not be included (by definition).

The NearestAirspace search will terminate when the earlier of NearestAirspaceMaximumItems or NearestAirspaceMaximumDistance is reached.

```
100 (>@c:NearestAirspaceMaximumDistance, nmiles)
```

❑ NearestAirspaceItemsNumber (enum) [Get]

The number of items actually retrieved during the NearestAirspace search, within the limitation specified by MaximumItems.

❑ NearestAirspaceCurrentLine (enum) [Get, Set]

The Index pointer. NearestAirspace search returns a list of airspace sectors. Setting NearestAirspaceCurrentLine allows you to select a specific sector in the list. Index numbers always start at 0.

❑ NearestAirspaceCurrentName (string) [Get]

The name of the airspace. For many airspace types, this is a descriptive, often recognizable name associated with the particular airspace location. In NearestAirspace searches, different sectors of the same airspace will sometimes have the same CurrentName, but are differentiated by different CurrentMinAltitude and CurrentMaxAltitudes and different CurrentNearDistance and CurrentAheadTime.

Center (type 1) airspaces have 8 character names constructed from the Region ID, the Center ID and Airspace names. As an example, the Center airspace name that Rotterdam, the Netherlands is within begins with the Netherlands Region ID, "EH", then the Center ID, "AA", then the Airspace name, 2302 which forms CurrentName EHAA2302. In the United States, the single character Region ID "K" is used, followed by the Air Route Traffic Control Center ID, then the Airspace name. For example, the Center airspace that Chicago O'Hare International airport is within is named KZAU4988. "ZAU" is the ARTCC ID, and 4988 is the Airspace name.

- **NearestAirspaceCurrentType (enum) [Get]**

A number representing airspace type. Refer to the Airspace Bit Table in NearestAirspaceQuery section. In MSFS, it is possible to be inside more than one airspace type or sector at the same time.

- **NearestAirspaceCurrentFrequency (MHz) [Get]**
- **NearestAirspaceCurrentFrequencyName (string = "Center") [Get]**

NearestAirspaceCurrentFrequency is the radio frequency (MHz) of Center airspace types found in a NearestAirspace search. Center airspaces types (type=1) must be included in NearestAirspaceQuery in order for CurrentFrequency to return a value other than 0.0. Apparently, only Center airspaces have an associated CurrentFrequency and CurrentFrequencyName. Airspace types 9, 10, 11, 12, and 13 (Tower, Clearance, Ground, Departure, and Approach) do not appear to have an associated CurrentFrequency and CurrentFrequencyName.

A NearestAirspace search that includes Centers may return several Center airspaces, all with different CurrentFrequency but the same CurrentFrequencyName, "Center". Centers have CurrentMinAltitude=0 (ground surface) and CurrentMaxAltitude=100000 meters or 328084 feet (edge of space).

As with all NearestAirspace search results, the airspace with the highest NearestAirspaceCurrentStatus will be listed first, then airspaces will be sorted by increasing distance from the reference point, which is usually the aircraft.

- **NearestAirspaceCurrentMinAltitude (feet) [Get]**
- **NearestAirspaceCurrentMaxAltitude (feet) [Get]**

The floor and ceiling of the airspace sector, measured above mean sea level. In fs9gps nomenclature, a value of 0 for MinAltitude refers to the ground surface. On aeronautical maps, it would be abbreviated "SFC"; in fs9gps terms, "0".

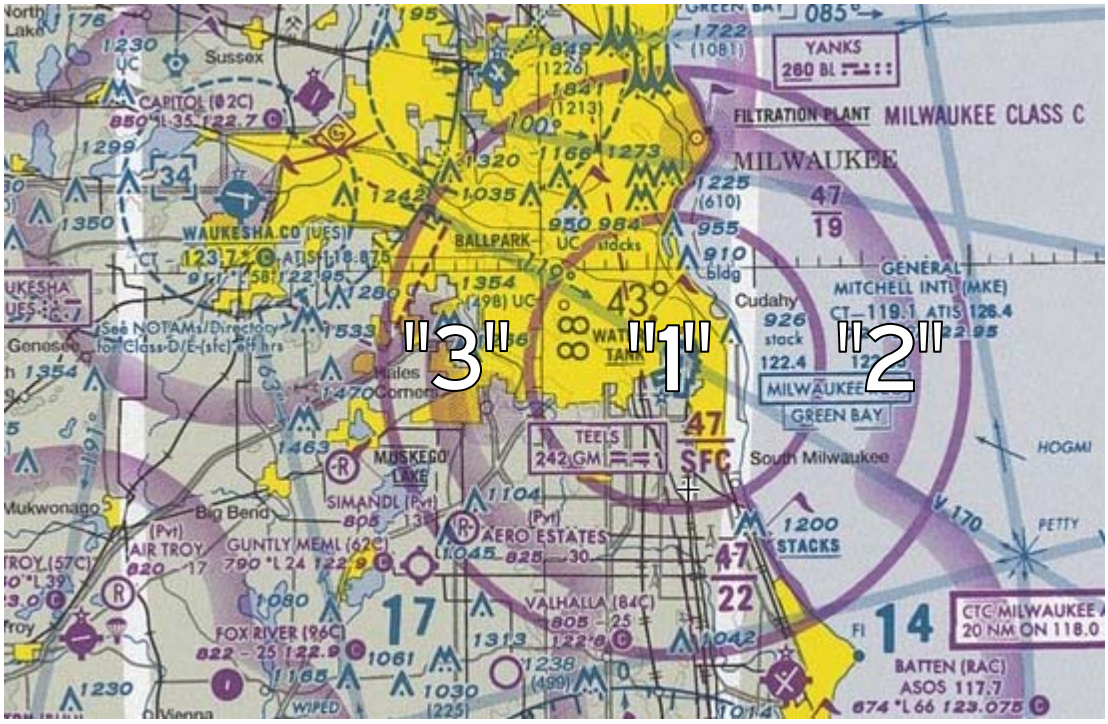
- **NearestAirspaceCurrentStatus (enum: 0, 1, 2, 3, or 4) [Get]**

NearestAirspaceCurrentStatus is a number that reflects an aircraft's positional status relative to nearby airspaces. It is determined from the "closeness" of the aircraft to airspace sectors in either x, y, z space or time. CurrentStatus, in turn, controls MessageItemsNumber and MessageCurrentType that can be used to display warning messages to the pilot. How close the aircraft needs to be to trigger a CurrentStatus change is determined by variables NearDistance, NearAltitude, and NearTime.

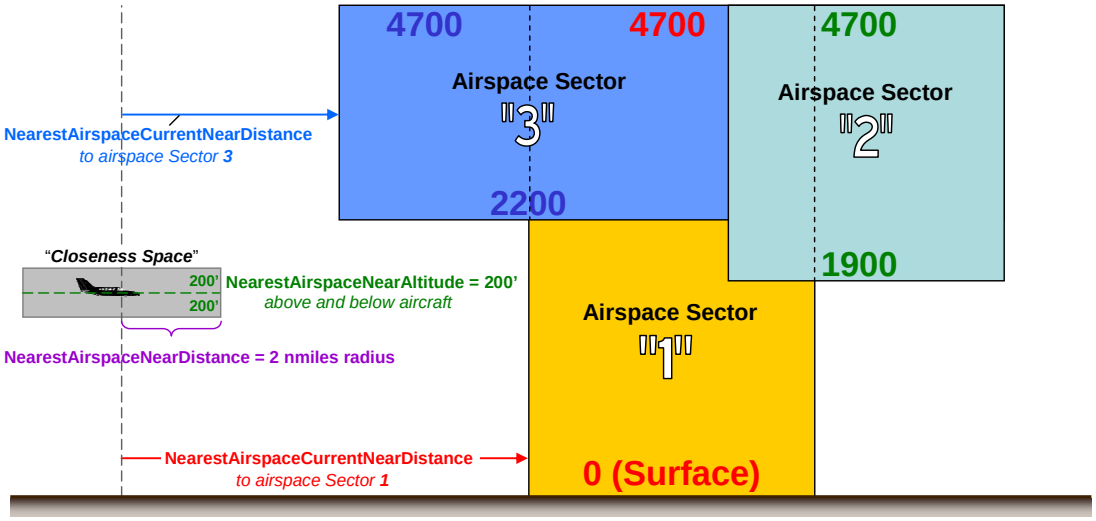
In x, y, z space, one way to think about "closeness" is to visualize as a disk extending outward from the aircraft NearDistance nmiles, whose height is two times NearAltitude feet, with the aircraft in the center. In the two dimensional diagram below, it is represented by the gray rectangle. When this "Closeness Space" (admittedly, not a proper name) touches/enters or exits an airspace sector, CurrentStatus changes. If the aircraft in the figure below is at an altitude of 1800', then as it flies through the Milwaukee airspace, CurrentStatus changes will be triggered when the "Closeness Space" touches Sector 1 and Sector 2. Regarding Sector 3, the aircraft is too low, or NearAltitude is too small, to trigger CurrentStatus values above 0.

In addition to x,y,z position, enroute time "closeness" also controls, or triggers, CurrentStatus changes.

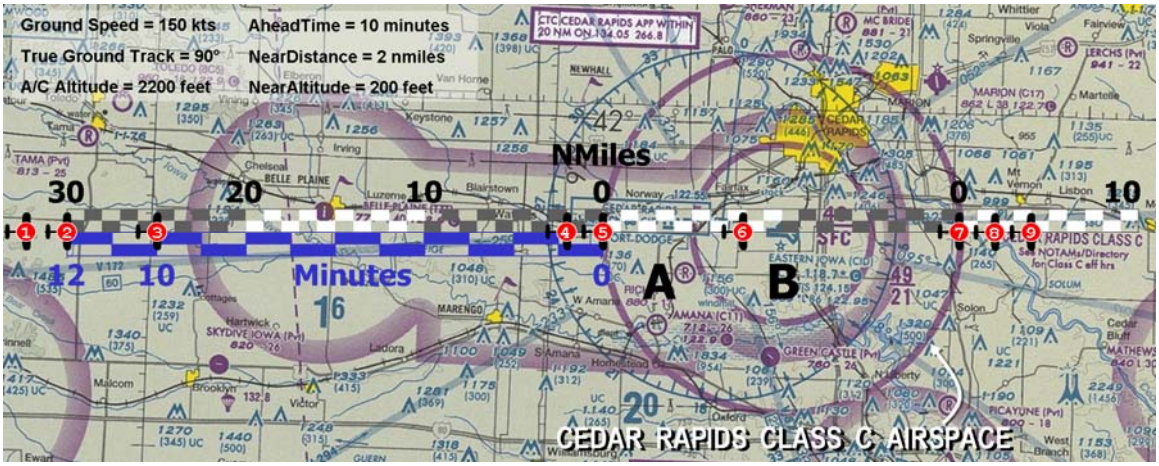
MILWAUKEE CLASS C AIRSPACE



MILWAUKEE CLASS C AIRSPACE



Enroute Time and Distance triggers can be demonstrated by tracking CurrentStatus changes in an example flight. In the figure below, an aircraft flies from Point 1 eastward through the Cedar Rapids Class C Airspace, continuing past Point 9.



CurrentStatus changes occur at various times and positions as the aircraft flies from west to east, as summarized in the table below. For the sake of simplicity, only the search results associated with Airspace Sector A (4900'/2100') are discussed:

Point	Airspace Current Status	Toward / Away from Airspace	Trigger Type	Trigger Condition for Status Change	Message Current Type	gps_500 Message
1	none	N/A	Time	CurrentAheadTime > 1.20 x AheadTime Aircraft too far from Airspace Sector A to display in Search	0 = NONE	none
2	0	Toward	Time	CurrentAheadTime ≤ 1.20 x AheadTime Aircraft appears in NearestAirspace search display	0 = NONE	none
3	2	Toward	Time	CurrentAheadTime ≤ 1.00 x AheadTime	2 = AHEAD	Airspace ahead -- less than 10 minutes
4	3	Toward	Distance	CurrrentNearDistance ≤ 1.00 x NearDistance	3 = NEAR_AHEAD	Airspace near and ahead
5	4	Inside	Distance	Entering Airspace Sector A	4 = INSIDE	Inside Airspace
6	4	Inside	Distance	Inside Airspace Sector A	4 = INSIDE	Inside Airspace
7	1	Away	Distance	Exiting Airspace Sector A CurrrentNearDistance counts upward from 0.00	1 = NEAR	none
8	0	Away	Distance	CurrrentNearDistance ≥ 1.00 x NearDistance	0 = NONE	none
9	none	N/A	Distance	CurrrentNearDistance ≥ 2.00 x NearDistance Aircraft too far from Airspace Sector A, dropped from Search	0 = NONE	none

Point 1 – The aircraft is too distant (in time) from Airspace Sector A to be displayed in the NearestAirspace search.

Point 2 – Airspace Sector A (4900'/2100') first appears in the NearestAirspace search display when CurrentAheadTime = 1.20 x AheadTime = 12 minutes. The 1.20 factor appears to be hard coded into the gps module. The initial CurrentStatus is 0.

```
NEAREST AIRSPACE SEARCH
  41.8837 :Current Lat   2198 :Cur Alt       153 :Gnd Speed     90 :Tru Gnd Trk
 -92.6176 :Current Lon      2 :Near Dist     200 :Near Alt      10 :Ahead Time
                           16 :Query (Dec)   100 :Max Dist      50 :Max Items

------------------------------------ Current ------------------------------------
Line             Name Type Status MaxAlt MinAlt NearDist AheadTime     Freq  FreqName
0         CEDAR RAPIDS    4      0   4900   2100     30.6     12.00    0.000
```

Point 3 - CurrentAheadTime = AheadTime. CurrentStatus = 2. MessageCurrentType = 2. gps_500 message = "Airspace ahead -- less than 10 minutes". CurrentStatus changes to 2 when CurrentAheadTime = AheadTime.

```
NEAREST AIRSPACE SEARCH
  41.8836 :Current Lat   2199 :Cur Alt       153 :Gnd Speed     90 :Tru Gnd Trk
 -92.5046 :Current Lon      2 :Near Dist     200 :Near Alt      10 :Ahead Time
                           16 :Query (Dec)   100 :Max Dist      50 :Max Items

------------------------------------ Current ------------------------------------
Line             Name Type Status MaxAlt MinAlt NearDist AheadTime     Freq  FreqName
0         CEDAR RAPIDS    4      0   4900   2100     25.5     10.00    0.000
1         CEDAR RAPIDS    4      0   4900      0     30.5     11.97    0.000
```

Point 4 – CurrentNearDistance = NearDistance. CurrentStatus = 3. MessageCurrentType = 3. gps_500 message = "Airspace near and ahead".

```
NEAREST AIRSPACE SEARCH
  41.8847 :Current Lat   2199 :Cur Alt       153 :Gnd Speed     90 :Tru Gnd Trk
 -91.9776 :Current Lon      2 :Near Dist     200 :Near Alt      10 :Ahead Time
                           16 :Query (Dec)   100 :Max Dist      50 :Max Items

------------------------------------ Current ------------------------------------
Line             Name Type Status MaxAlt MinAlt NearDist AheadTime     Freq  FreqName
0         CEDAR RAPIDS    4      3   4900   2100      2.0      0.77    0.000
1         CEDAR RAPIDS    4      2   4900      0      7.0      2.74    0.000
```

Point 5 – Entering Airspace Sector A. CurrentNearDistance = 0. CurrentStatus = 4. MessageCurrentType = 4. gps_500 message = "Inside Airspace".

```
NEAREST AIRSPACE SEARCH
  41.8850 :Current Lat   2199 :Cur Alt       153 :Gnd Speed     90 :Tru Gnd Trk
 -91.9302 :Current Lon      2 :Near Dist     200 :Near Alt      10 :Ahead Time
                           16 :Query (Dec)   100 :Max Dist      50 :Max Items

------------------------------------ Current ------------------------------------
Line             Name Type Status MaxAlt MinAlt NearDist AheadTime     Freq  FreqName
0         CEDAR RAPIDS    4      4   4900   2100      0.0      0.00    0.000
1         CEDAR RAPIDS    4      2   4900      0      4.8      1.90    0.000
```

Point 6 – Aircraft is inside both Sector A and B. For both sectors, CurrentNearDistance = 0. CurrentStatus = 4. MessageCurrentType = 4. gps_500 message = "Inside Airspace".

```
NEAREST AIRSPACE SEARCH
  41.8857 :Current Lat   2198 :Cur Alt       153 :Gnd Speed    89 :Tru Gnd Trk
 -91.8204 :Current Lon      2 :Near Dist     200 :Near Alt     10 :Ahead Time
                           16 :Query (Dec)   100 :Max Dist     50 :Max Items

------------------------------------ Current ------------------------------------
Line              Name Type Status MaxAlt MinAlt NearDist AheadTime     Freq  FreqName
0          CEDAR RAPIDS    4      4   4900      0      0.0      0.00   0.000
1          CEDAR RAPIDS    4      4   4900   2100      0.0      0.00   0.000
```

Point 7 – Exiting Airspace Sector A. CurrentNearDistance = 0, and begins counting upwards. CurrentStatus = 1. MessageCurrentType = 1. gps_500 message = none. Note that AheadTime equals 8464.92 minutes. At 153 knots ground speed, this equates to 21,586 nmiles, which is the circumference of the earth at 2200' asl. This is an indication the airspace sector is *behind* the aircraft.

```
NEAREST AIRSPACE SEARCH
  41.8862 :Current Lat   2199 :Cur Alt       153 :Gnd Speed    90 :Tru Gnd Trk
 -91.4829 :Current Lon      2 :Near Dist     200 :Near Alt     10 :Ahead Time
                           16 :Query (Dec)   100 :Max Dist     50 :Max Items

------------------------------------ Current ------------------------------------
Line              Name Type Status MaxAlt MinAlt NearDist AheadTime     Freq  FreqName
0          CEDAR RAPIDS    4      1   4900   2100      0.1   8464.92   0.000
```

Point 8 - CurrentNearDistance = 2.0 and still counting upwards. CurrentStatus = 0. MessageCurrentType = 0. gps_500 message = none.

```
NEAREST AIRSPACE SEARCH
  41.8861 :Current Lat   2199 :Cur Alt       153 :Gnd Speed    90 :Tru Gnd Trk
 -91.4390 :Current Lon      2 :Near Dist     200 :Near Alt     10 :Ahead Time
                           16 :Query (Dec)   100 :Max Dist     50 :Max Items

------------------------------------ Current ------------------------------------
Line              Name Type Status MaxAlt MinAlt NearDist AheadTime     Freq  FreqName
0          CEDAR RAPIDS    4      0   4900   2100      2.1   8463.80   0.000
```

Point 9 - CurrentNearDistance = 2.0 = 2.00 x NearDistance. At 2 times NearDistance, the airspace sector is dropped out of the NearestAirspace search display. The 2.00 factor appears to be hard coded into the gps module.

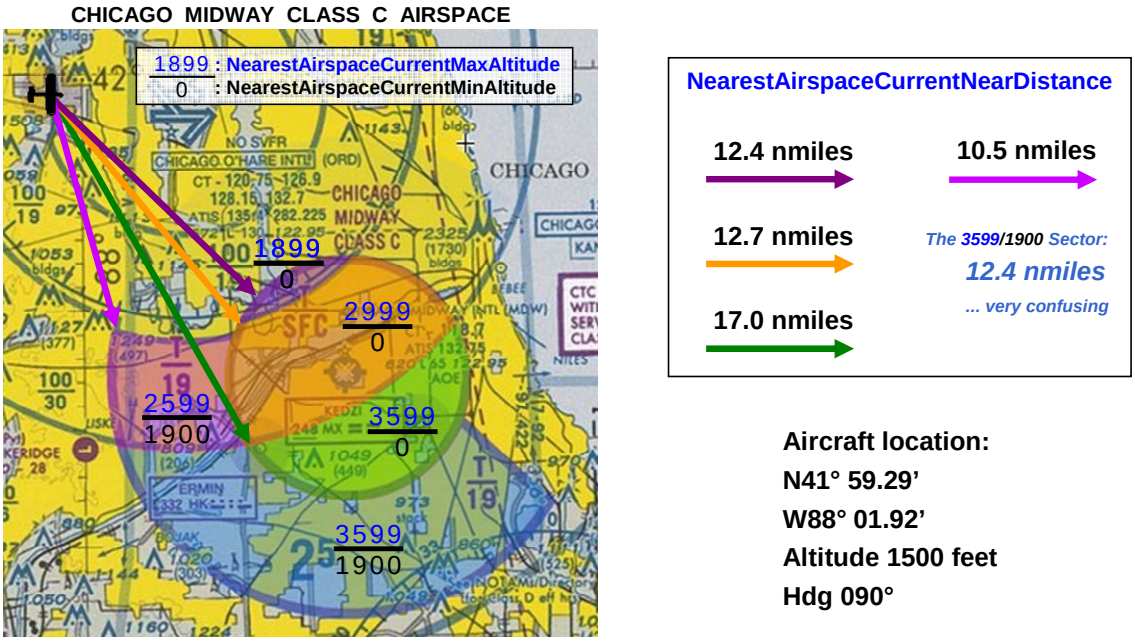
The results of a NearestAirspace search are always sorted first by CurrentStatus, then by increasing distance. All the Status = 4 airspaces are first listed in ascending distance, then the Status = 3 airspaces are listed in ascending order, and so forth.

❑ NearestAirspaceCurrentNearDistance (nmiles) [Get]

NearestAirspaceCurrentNearDistance is the ground distance between the current position of the aircraft and the closest point of each airspace sector to the aircraft (or other reference point).

The figure below demonstrates NearestAirspaceCurrentNearDistance for an aircraft near Chicago Midway International airport after a NearestAirspace search of Class "C" airspaces.

NearestAirspaceCurrentNearDistance



The NearestAirspace search returns a separate CurrentNearDistance for each airspace sector. Note the confusing (to me) result of the far airspace, 3599 / 1900. CurrentNearDistance of this sector is the *same* as the closest airspace that extends to the surface. CurrentNearDistance rules pertaining to 'hidden' or 'farthest' sectors such as this are not clear to me.

NearestAirspaceCurrentAheadTime (minutes) [Get]

NearestAirspaceCurrentAheadTime is the enroute (elapsed) time until the aircraft physically enters the airspace sector. When flying toward an airspace sector, AheadTime counts down, starting at 12 minutes when the sector first appears in NearestAirspace search. CurrentAheadTime = 0 while the aircraft is inside the sector. Leaving the airspace sector is more interesting. Upon exit, CurrentAheadTime resumes a countdown until the aircraft again enters the sector *after flying around the globe*. However, the NearestAirspace distance trigger, CurrentNearDistance, drops the sector from NearestAirspace search when CurrentNearDstance is 2X NearDistance.

MESSAGE GROUP

The Message Group variables control the content of the Airspace messages displayed by the gps_500 gauge. They do not control *when* a message is displayed – variables in the NearestAirspace Group do that. Instead, the Message Group variables determine *which* message will be displayed.

There are four different airspace messages in the gps_500 gauge, and under the right circumstances, up to three can be displayed at the same time. Consequently, the messages are indexed, and a typical display loop is used for the display.

❑ MessageItemsNumber (enum) [Get]

MessageItemsNumber is the number of messages that are active and can be seen by pressing the MSG button on the gps_500 gauge.

❑ MessageCurrentLine (enum) [Get, Set]

MessageCurrentLine is the Index pointer used in the display.

❑ MessageCurrentType (enum) [Get]

MessageCurrentType is the id number of the specific message.

1. Near Airspace – less than 2 nm
2. Airspace ahead – less than 10 minutes
3. Airspace near and ahead
4. Inside Airspace

❑ NewMessagesNumber (enum) [Get]

NewMessagesNumber indicates the number of new messages waiting to be read. It is reset to 0 when the pilot presses the MSG Button to view the messages, but can increase from 1 to 2 or 3 if the messages are ignored.

❑ NewMessagesConfirm (enum) [Set]

NewMessagesConfirm is a write-only variable that is set to 1 when the pilot presses the MSG Button after viewing the message(s).

ICAO SEARCH GROUP

ICAOSearch is a procedure that attempts to find ICAOs that contain the Ident the user enters. The ICAO is important because it is required before access to variables in the Waypoint Groups is possible. ICAOSearch data entry can be accomplished via direct keyboard entry, mouse click, or from code.

While the 12 character ICAO is unique, Idents are sometimes not, and consequently ICAOSearch may find multiple ICAOs that can be matched to facility type and Ident. Because of this, ICAOSearch results are indexed and require an Index Pointer in order to access the desired ICAO in the list returned by ICAOSearch. Often, however, only one ICAO is found that matches the input parameters and the default Index Pointer value 0 (zero) automatically takes care of selection of the proper ICAO.

			NameSearch			ICAOSearch
Facility	**Waypoint Group**	**Name Exists?**	**NAME Searchable?**	**IDENT Exists?**	**ICAO Type**	**IDENT Searchable?**
AIRPORT	AIRPORT	YES	YES	YES	A	YES
RUNWAY WAYPOINT	None	NO	NO	YES	R	NO
VOR	VOR	YES	NO	YES	V	YES
ILS / LOC	AIRPORT	YES	NO	YES	V	YES
NDB	NDB	YES	NO	YES	N or X	YES
fs9gps WAYPOINT	INTERSECTION	NO	NO	YES	W	YES
USER WAYPOINT	None	NO	NO	NO	None	NO

❑ IcaoSearchInitialIcao (string) [Get, Set]

IcaoSearchInitialIcao is the ICAO of the first Ident displayed as certain gps_500 pages containing ICAOSearch code open. It is set by the current FacilityICAO (gps_500 lines 3808 and 3813). In the event that user input updates ICAOSearch and the Ident displayed on the screen changes, but the user ultimately cancels the selections, then the current ICAO will revert back to IcaoSearchInitialIcao.

❑ IcaoSearchStartCursor (1 to 5 character string) [Set]

IcaoSearchStartCursor is used to filter, or restrict, the gps search of ICAOs to those of a certain type:

StartCursor	FACILITY
A	AIRPORT
V	VOR
N or X	NDB
W	WAYPOINT / INTERSECTION
M	Does Not Exist

Only these letters may be used in IcaoSearchStartCursor.

Any combination of the letters can also be entered. 'AVNW' will enable a search of all types of ICAOs. Searching 'W' will yield results for VORs and NDBs in addition to Waypoint/Intersections.

The gps_500 gauge suggests a StartCursor of 'M' , Marker (gps_500 lines 3813, 3814), but no searchable Facility with StartCursor 'M' exists in the fs9gps database.

❑ IcaoSearchStopCursor (enum) [Set]

I am not completely sure what this variable accomplishes. IcaoSearchStopCursor is an enum that appears only in the Enter and Clear macros in the gps_500 (`<Macro Name="ENTButton">` and `<Macro Name="CancelInput">`). The value assigned in the macro is always zero. Consequently, it appears related to the cancelation of user input ("Cursor").

Having said that, I've not yet needed StopCursor in any of the scripts written in preparation of this guidebook. As well, I've assigned different values to StopCursor and also removed it from the gps_500 gauge, all with no effect that I have been able to see.

MSFT put it there for a reason, but so far, it escapes me.

DATA ENTRY METHODS FOR ICAOSearch

There are three methods of data entry of the search filter and Ident:

1. **Keyboard Direct Entry**. The user types input information on the keyboard. IcaoSearchEnterChar will automatically invoke IcaoSearchAdvanceCursor and automatically concatenate keystroke entries, allowing for continuous typing.

```
<On Key="AlphaNumeric">
      <Visible> (L:ICAOSearchEntry, enum) 101 == </Visible>
      (M:Key) chr (>@c:IcaoSearchEnterChar)
</On>
```

2. **Mouse**. Click spots are used to mimic the use of knobs to enter the Ident, as in a Garmin GNS 500 or the FS9 gps_500 gauge. Note that in the example below, IcaoSearchAdvanceCursor and IcaoSearchAdvanceCharacter are used, but IcaoSearchEnterChar is not needed. When entering an Ident with the mouse, as the user advances the cursor, `1 (>C:fs9gps:IcaoSearchAdvanceCursor)`, the character currently selected by AdvanceCharacter is automatically entered into EnterChar. Additionally, as the cursor continues to advance, the character selected using AdvanceCharacter is concatenated with the previous characters, thus building the Ident string. The string is passed to EnterChar each time a character is selected (each time AdvanceCharacter is 'clicked').

   Garmin-type GNS knob, Upper Left click spot:

```
<Area Left="470" Top="307" Width="25" Height="12">
      <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="No">
      -1 (>C:fs9gps:IcaoSearchAdvanceCursor)
      </Click>
</Area>
```

   Upper Right click spot:

```
<Area Left="500" Top="307" Width="25" Height="12">
      <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="No">
      1 (>C:fs9gps:IcaoSearchAdvanceCursor)
      </Click>
</Area>
```

   Lower Left click spot:

```
<Area Left="480" Top="336" Width="15" Height="12">
      <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="Yes">
      -1 (>C:fs9gps:IcaoSearchAdvanceCharacter)
      </Click>
</Area>
```

Lower Right click spot:

```
<Area Left="500" Top="336" Width="15" Height="12">
      <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="Yes">
      1 (>C:fs9gps:IcaoSearchAdvanceCharacter)
      </Click>
</Area>
```

3. **Code Entry**. The users gauge code may enter data into the IcaoSearchEnterChar variable:

   ```
   'V' (>C:fs9gps:IcaoSearchStartCursor)
   ```

   Code entry of StartCursor, rather than keyboard or mouse, is always used.

   ```
   (A:NAV1 Ident, string) (>C:fs9gps:IcaoSearchEnterChar)
   ```

   or

   ```
   'SEA' (>C:fs9gps:IcaoSearchEnterChar)
   ```

   Up to 5 characters can be entered at once using Code Entry.

❑ IcaoSearchAdvanceCursor (enum) [Set]

IcaoSearchAdvanceCursor is cursor position 'incrementer' necessary when mouse entry of the Ident string is used, as if manipulating the large right knob on a Garmin GNS 500 or the FS9 stock gps-500 gauge.

AdvanceCursor value of 1 or -1, will advance the cursor one position, or back up one position. Not only does -1 cause the cursor to back up one position, so does -2, -3, etc -- they all cause the cursor to back up only one position. A zero value causes the cursor to not move, although zero is not a logical choice for any application. A value of 1 or greater causes the cursor to advance, but only one position at a time regardless of AdvanceCursor value.

❑ IcaoSearchAdvanceCharacter (enum) [Set]

An alphabet advance or backup 'incrementer'. Either 1 or -1 is used for input. The value -1 (or any other negative value) causes the letter at the current cursor position to back up one letter of the alphabet at a time. An AdvanceCharacter value of zero causes no progress or back up in the alphabet, but, of course, would be an illogical choice for applications. A value of 1 (or any positive integer) causes an advance of one letter in the alphabet.

For Airports, the alphabet values are alphanumeric characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
```

The 'space' character is between Z and 0 (zero). Advancing one character from '9' yields 'A'. Backing up one character from '0' (zero) yields the space character. Backup one more yields 'Z'.

For other Facilities, the alphabet values are ascii characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&'()*+,./:;<=>?@[]^_`{|}
~space
```

The last character is the space character. These are shown in the order they are processed by AdvanceCharacter.

Summarizing,

```
1 (>c:fs9gps:IcaoSearchAdvanceCharacter)
```

causes the next letter to be selected, and

```
-1 (>c:fs9gps:IcaoSearchAdvanceCharacter)
```

causes the previous letter to be selected.

❑ IcaoSearchEnterChar (string) [Set]

Similar with AdvanceCharacter and AdvanceCursor, IcaoSearchEnterChar is used to enter the Ident string that the gps module will search for in its database. EnterChar automatically concatenates the Ident character entry to form CurrentIdent. As an example using keyboard direct entry, the user types A, B, C, D, E, F, and then G, and gets the following results:

Keyboard Entry Sequence	EnterChar	CurrentIdent	String Length
1 - 'A'	A	A	1
2 - 'B'	B	AB	2
3 - 'C'	C	ABC	3
4 - 'D	D	ABCD	4
5 - 'E'	E	ABCDE	5
6 - 'F'	F	ABCDF	5
7 - 'G'	G	ABCDG	5

	Cursor Advance	Concatenation
Keyboard Direct Entry	Automatic	Automatic
Mouse Entry	by Mouse	Automatic
Code Entry	Not Necessary	Not Necessary

There is a 5 character limit to Ident entry because Idents have a maximum string length of 5 characters. If more than 5 characters are entered, then the letter in cursor position 4 (the 5th character - the first cursor position is numbered zero) is replaced with the excess character, as above.

If EnterChar is used in association with direct keyboard entry, then the <On Key> parameters **AlphaNumeric** or **Ascii** determine which characters may be entered.

Using **AlphaNumeric**, only characters

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890
```

may be entered. **AlphaNumeric** is a good choice for Idents of an ICAOSearch.

Ascii is similar to **AlphaNumeric** except that characters "**+**", "**-**", "**,**", and "**.**" (plus, minus, comma and period/decimal point) are also accepted. **Ascii** is the choice when entering numbers such as Latitude and Longitude because of the minus sign and decimal point. Note that the gps_500 gauge uses **Ascii** for NameSearch entry (see line 3947).

❑ IcaoSearchBackupChar (enum) [Set]

IcaoSearchBackupChar is used for backspace when entering Ident via keyboard direct entry.

For example, from gps_500 lines 3951 – 3955:

```
<On Key="Backspace">
    <Visible>(C:fs9gps:enteringInput)</Visible>
      10 19 (C:fs9gps:enteringInput) rng 31
      (C:fs9gps:enteringInput) == ||
            if{
                  -1 (>C:fs9gps:IcaoSearchBackupChar) @InitBlinker quit
            }
</On>
```

Another, simpler example:

```
<On Key="Backspace">
    <Visible>(L:AsciiEntryEnable, bool)</Visible>
    (M:Key) (>C:fs9gps:IcaoSearchBackupChar)
</On>
```

Neither (M:Key) nor a number actually need to be entered in order to create a backspace. All of the following create a single backspace at a time as the keyboard backspace key is entered:

```
<On Key="Backspace">
    <Visible>(L:AsciiEntryEnable, bool)</Visible>
    (M:Key) -1 (>C:fs9gps:IcaoSearchBackupChar)
</On>

<On Key="Backspace">
    <Visible>(L:AsciiEntryEnable, bool)</Visible>
    (M:Key) 1 (>C:fs9gps:IcaoSearchBackupChar)
</On>

<On Key="Backspace">
    <Visible>(L:AsciiEntryEnable, bool)</Visible>
    (M:Key) 0 (>C:fs9gps:IcaoSearchBackupChar)
</On>

<On Key="Backspace">
    <Visible>(L:AsciiEntryEnable, bool)</Visible>
    (M:Key) -200 (>C:fs9gps:IcaoSearchBackupChar)
</On>


<On Key="Backspace">
    <Visible>(L:AsciiEntryEnable, bool)</Visible>
    (>C:fs9gps:IcaoSearchBackupChar)
</On>
```

Note the use of <Visible> tags. In the examples above, only when `L:AsciiEntryEnable = 1` will keyboard direct entry for the Ident work. The ability to "turn off" keyboard direct entry is necessary in order to retain normal FS9/X keyboard entry instructions such as "G" = Landing Gear Toggle when Ident entry is not needed.

❑ IcaoSearchCursorPosition (enum) [Get]

IcaoSearchCursorPosition is the current cursor position of the Ident entry. The first character entered is CursorPosition 0. The second is CursorPosition 1, and so forth.

CursorPosition is active for all types of Ident entry – keyboard, mouse, code.

❑ IcaoSearchCurrentIdent (string) [Get]

IcaoSearchCurrentIdent is the facility Ident constructed (concatenated) as the user enters Ident characters. It grows (SLEN increases by one) with each keystroke or mouse AdvanceCursor entry.

❑ IcaoSearchCurrentIcao (string) [Get, Set]

IcaoSearchCurrentIcao is the ICAO formed using StartCursor and CurrentIdent. Because Idents are not unique, there may be multiple ICAOs that can be matched to StartCursor and CurrentIdent. Consequently, CurrentIcao is an indexed variable (a list) that requires an Index Pointer (IcaoSearchMatchedIcao) to access.

With each Ident character entry, the gps module searches the database for ICAOs containing StartCursor and CurrentIdent and if there are any, the number of matches is stored into IcaoSearchMatchedIcaosNumber.

❑ IcaoSearchCurrentIcaoType (string) [Get]

IcaoSearchCurrentIcaoType is the facility type and is the same as StartCursor:

IcaoType	Facility
A	Airport
V	VOR, ILS, LOC
N	NDB
W	Waypoint, Intersection

❑ IcaoSearchCurrentIcaoRegion (string, SLEN=2) [Get]

The two character Region code.

❑ IcaoSearchMatchedIcaosNumber (enum) [Get]

IcaoSearchMatchedIcaosNumber is the number of ICAOs that were matched to StartCursor and CurrentIdent during ICAOSearch.

❑ IcaoSearchMatchedIcao (enum) [Get, Set]

The Index Pointer used to access specific ICAOs returned by ICAOSearch. The default is zero.

RESOLVING MULTIPLE ICAO MATCHES

An example of one method that can be used to resolve multiple ICAOSearch matches is discussed in the **ICAO SEARCH EXAMPLE** chapter (starting on page 34).

NAME SEARCH GROUP

NameSearch is a procedure that allows the user to retrieve the ICAO by entering facility Name. It is limited to Airport Name search only. The ICAO is important because it is required before access to variables in the Waypoint Airport Group is possible. NameSearch data entry can be accomplished via direct keyboard entry, mouse click, or from code.

		NameSearch			ICAOSearch
Facility	Waypoint Group	Name Exists?	NAME Searchable?	IDENT Exists?	ICAO Type	IDENT Searchable?
AIRPORT	AIRPORT	YES	YES	YES	A	YES
RUNWAY WAYPOINT	None	NO	NO	YES	R	NO
VOR	VOR	YES	NO	YES	V	YES
ILS / LOC	AIRPORT	YES	NO	YES	V	YES
NDB	NDB	YES	NO	YES	N or X	YES
fs9gps WAYPOINT	INTERSECTION	NO	NO	YES	W	YES
USER WAYPOINT	None	NO	NO	NO	None	NO

❑ NameSearchInitialIcao (string) [Get, Set]

NameSearchInitialIcao is the ICAO of the first Name displayed as certain gps_500 pages containing NameSearch code open. It is set by the current FacilityICAO (gps_500 lines 3809 and 3814). In the event that user input updates NameSearch and the Name displayed on the screen changes, but the user ultimately cancels the selections, then the current ICAO will revert back to NameSearchInitialIcao.

❑ NameSearchInitialName (string) [Get, Set]

NameSearchInitialName is the Airport Name associated with NameSearchInitialIcao.

❑ NameSearchStartCursor (string) [Set]

NameSearchStartCursor is included in the gps_500 gauge (line 3814) in parallel with NameSearch input. Unlike NameSearchStartCursor, however, it appears to be non-functional and not needed.

❑ NameSearchStopCursor (enum) [Set]

I am not sure what this variable does. NameSearchStopCursor is an enum that is used only in the Enter and Clear macros in the gps_500 (<Macro Name="ENTButton"> and <Macro Name="CancelInput">). In the macros, the value assigned to each is

always zero. The value assigned in the macro is always zero. Consequently, it appears related to the cancelation of user input ("Cursor").

Having said that, I've not needed StopCursor, not yet anyway, in any of the scripts written in preparation of this guidebook. As well, I've assigned different values to StopCursor and also removed it from the gps_500 gauge, all with no effect that I have have been able to see.

MSFT put it there for a reason, but so far, it escapes me.

❑ NameSearchAdvanceCursor (enum) [Set]

NameSearchAdvanceCursor is cursor position 'incrementer' necessary when mouse entry of the Ident string is used, as if manipulating the large right knob on a Garmin GNS 500 or the FS9 stock gps-500 gauge.

AdvanceCursor value of 1 or -1, will advance the cursor one position, or back up one position. Not only does -1 cause the cursor to back up one position, so does -2, -3, etc -- they all cause the cursor to back up only one position. A zero value causes the cursor to not move, although zero is not a logical choice for any application. A value of 1 or greater causes the cursor to advance, but only one position at a time regardless of AdvanceCursor value.

❑ NameSearchAdvanceCharacter (enum) [Set]

An alphabet advance or backup 'incrementer'. Either 1 or -1 is used for input. The value -1 (or any other negative value) causes the letter at the current cursor position to back up one letter of the alphabet at a time. An AdvanceCharacter value of zero causes no progress or back up in the alphabet, but, of course, would be an illogical choice for applications. A value of 1 (or any positive integer) causes an advance of one letter in the alphabet.

For Airports, the alphabet values are alphanumeric characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
```

The 'space' character is between Z and 0 (zero). Advancing one character from '9' yields 'A'. Backing up one character from '0' (zero) yields the space character. Backup one more yields 'Z'.

Summarizing,

```
1 (>c:fs9gps:NameSearchAdvanceCharacter)
```

causes the next letter to be selected, and

```
-1 (>c:fs9gps:NameSearchAdvanceCharacter)
```

causes the previous letter to be selected.

❑ NameSearchEnterChar (string) [Set]

NameSearchEnterChar is used to enter the Name string that the gps module will search for in its database.

There are three common methods of data entry/input of the search filter and Name:

❑ **Keyboard Direct Entry**. The user types input information on the keyboard. EnterChar will automatically invoke NameSearchAdvanceCursor and automatically concatenate keystroke entries, allowing for continuous typing.

```
<On Key="AlphaNumeric">
<Visible> (L:NameSearchEntry, enum) 101 == </Visible>
(M:Key) chr (>@c:NameSearchEnterChar)
</On>
```

❑ **Mouse.** Click spots are used to mimic the use of knobs to enter the Name, as in a Garmin GNS 500 or the FS9 gps_500 gauge. Note that in the example below, NameSearchAdvanceCursor and NameSearchAdvanceCharacter are used, but NameSearchEnterChar is not needed. When entering a Name with the mouse, as the user advances the cursor,

```
1 (>C:fs9gps:NameSearchAdvanceCursor),
```

the character currently selected by AdvanceCharacter is automatically entered into EnterChar. Additionally, as the cursor continues to advance, the character selected using AdvanceCharacter is concatenated with the previous characters, thus building the Name string. The string is passed to EnterChar each time a character is selected (each time AdvanceCharacter is 'clicked').

Garmin-type GNS knob, Upper Left click spot:

```
<Area Left="470" Top="307" Width="25" Height="12">
   <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="No">
         -1 (>C:fs9gps:NameSearchAdvanceCursor)
      </Click>
</Area>
```

Upper Right click spot:

```
<Area Left="500" Top="307" Width="25" Height="12">
   <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="No">
         1 (>C:fs9gps:NameSearchAdvanceCursor)
      </Click>
</Area>
```

Lower Left click spot:

```
<Area Left="480" Top="336" Width="15" Height="12">
   <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="Yes">
         -1 (>C:fs9gps:NameSearchAdvanceCharacter)
      </Click>
</Area>
```

Lower Right click spot:

```
<Area Left="500" Top="336" Width="15" Height="12">
   <Cursor Type="Hand" />
      <Click Kind="LeftSingle" Repeat="Yes">
         1 (>C:fs9gps:NameSearchAdvanceCharacter)
      </Click>
</Area>
```

- **Code Entry**. Code may be used to enter the Name:

`'Garde' (>C:fs9gps:NameSearchEnterChar)`

returns the ICAO A _ _ _ _ _ _ EDOC' (Gardelegen Airport, Gardelegen, Germany).

While,
`'Garder' (>C:fs9gps:NameSearchEnterChar)`, or
`'Garderm' (>C:fs9gps:NameSearchEnterChar)`, or
`'Gardermo' (>C:fs9gps:NameSearchEnterChar)`, or
`'Gardermoe' (>C:fs9gps:NameSearchEnterChar)`, or
`'Gardermoen' (>C:fs9gps:NameSearchEnterChar)`

all return the ICAO A _ _ _ _ _ _ ENGM' (Gardermoen Airport, Oslo, Norway)

EnterChar and AdvanceCursor/AdvanceCharacter automatically concatenates the character entry to form CurrentName. Up to 80 characters can be entered into CurrentName even though Airport Names are not nearly that long.

	Cursor Advance	Concatenation
Keyboard Direct Entry	Automatic	Automatic
Mouse Entry	by Mouse	Automatic
Code Entry	Not Necessary	Not Necessary

❑ NameSearchBackupChar (enum) [Set]

NameSearchBackupChar is used for backspace when entering a Name via keyboard direct entry.

For example, gps_500 lines 3951 – 3955:

```
<On Key="Backspace">
    <Visible>(C:fs9gps:enteringInput)</Visible>
      110 119 (C:fs9gps:enteringInput) rng 131
      (C:fs9gps:enteringInput) == ||
            if{
                  -1 (>C:fs9gps:NameSearchBackupChar) @InitBlinker quit
            }
</On>
```

Another, simpler example:

```
<On Key="Backspace">
    <Visible>(L:AlphanumericEntryEnable, bool)</Visible>
    (M:Key) (>C:fs9gps:NameSearchBackupChar)
</On>
```

Neither (M:Key) nor a number actually need to be entered in order to create a backspace. All of the following create a single backspace at a time as the keyboard backspace key is entered:

```
<On Key="Backspace">
    <Visible>(L:AlphanumericEntryEnable, bool)</Visible>
    (M:Key) -1 (>C:fs9gps:NameSearchBackupChar)
</On>
```

```
<On Key="Backspace">
    <Visible>(L:AlphanumericEntryEnable, bool)</Visible>
    (M:Key) 1 (>C:fs9gps:NameSearchBackupChar)
</On>

<On Key="Backspace">
    <Visible>(L:AlphanumericEntryEnable, bool)</Visible>
    (M:Key) 0 (>C:fs9gps:NameSearchBackupChar)
</On>

<On Key="Backspace">
    <Visible>(L:AlphanumericEntryEnable, bool)</Visible>
    (M:Key) -200 (>C:fs9gps:NameSearchBackupChar)
</On>

<On Key="Backspace">
    <Visible>(L:AlphanumericEntryEnable, bool)</Visible>
    (>C:fs9gps:NameSearchBackupChar)
</On>
```

Note the use of `<Visible>` tags. In the examples above, only when `L:AlphanumericEntryEnable` = 1 will keyboard direct entry for the Name work. The ability to "turn off" keyboard direct entry is necessary in order to retain normal FS9/X keyboard entry instructions such as "G" = Landing Gear Toggle when Name entry is not needed.

❑ NameSearchCursorPosition (enum) [Get]

NameSearchCursorPosition is the current cursor position of the Name entry. The first character entered is CursorPosition 0. The second is CursorPosition 1, and so forth.

CursorPosition is active for all types of Name entry – keyboard, mouse, code.

❑ NameSearchCurrentName (string) [Get, Set]

NameSearchCurrentIdent is the airport Name constructed (concatenated) as the user enters Name characters. It grows (SLEN increases by one) with each keystroke or mouse AdvanceCursor entry.

❑ NameSearchCurrentMatch (enum) [Get]

NameSearchCurrentMatch is the number of ICAOs that were matched to the Airport Name during the NameSearch. It is always either 0 or 1.

❑ NameSearchCurrentIcao (string) [Get]

NameSearchCurrentIcao is the ICAO associated with the current Airport Name. As an Airport Name is being entered via direct keyboard entry or mouse one character at a time, an Airport Name is progressively 'concatenated' and a NameSearch performed as each new character is entered.

❑ NameSearchCurrentIcaoType (string) [Get]

NameSearchCurrentIcaoType is always 'A', for Airport.

❑ NameSearchCurrentIcaoRegion (string) [Get]

NameSearchCurrentIcaoRegion is always a blank string because Airport ICAOs do not contain a Region Code.

FLIGHT PLAN GROUP

The Flight Plan Data Group variables control the navigation engine of the gps. They cover Flight Planning to En Route Navigation to Instrument Approaches.

In my opinion, the flight navigation capability provided by the Flight Plan Data Group variables is thorough and pretty impressive, especially considering the inexpensive price of the software. The Flight Plan Group is by far the largest Group within fs9gps, containing 99 variables.

I find that understanding and predicting the response of the Flight Plan variables requires documentation of significant detail, at least for my purposes, which explains the length of this chapter. Still, there are things not covered and still not understood.

Flight Planning

COMPONENTS OF THE FLIGHT PLAN

The Flight Plan components are a group of read-only variables that are set through use of FS9's Flight Planner, mid-flight Flight Plan filing using FS9's ATC capability, user editing of the Flight Plan.PLN file, or through third-party Flight Planners (I've never investigated any, but several are out there that look pretty good).

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

FlightPlanTitle is created from Departure Airport Ident and Destination airport Ident. "Departure Airport Ident to Destination Airport Ident"

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

FlightPlanDescription is created from Departure Airport Ident and Destination airport Ident. "Departure Airport Ident, Destination Airport Ident"

- **FlightPlanFlightPlanType (enum) [Get]**

FlightPlanFlightPlanType is a number defining the type of Flight Plan.

- 0 = NONE
- 1 = VFR
- 2 = IFR

❑ FlightPlanRouteType (enum) [Get]

FlightPlanRouteType is an enum representing basic routing type. See table below.

ATC ROUTE TYPE

Bit	Name and Type #	Bit	Name and Type #
0	DIRECT = 0	2	LOWALT = 2
1	VOR = 1	3	HIGHALT = 3

http://msdn.microsoft.com/en-us/library/cc526954.aspx#ATC_ROUTETYPE

❑ FlightPlanCruisingAltitude (feet) [Get]

Discussed toward at the end of this section.

❑ FlightPlanDepartureAirportIdent (string) [Get]

The 3 to 4 character Ident of the Departure Airport.

❑ FlightPlanDepartureLatitude
❑ FlightPlanDepartureLongitude (degrees, radians) [Get]

Latitude and Longitude of the starting position of the aircraft at the departure airport. This could be parked at a gate or on a runway, and these will represent different coordinates. It is fixed at the starting point and does not change as the aircraft taxis for takeoff. If FS9 Flight Planner is used to position the aircraft on the Active Runway, then as shown in the figure below, the aircraft will be placed approximately 175 feet beyond the physical end of the runway. Units are degrees (decimal format, not deg, min, sec) or radians.

❑ FlightPlanDepartureAltitude (feet) [Get]

Altitude (asl) of the departure location, or, starting waypoint of the flight plan.

❑ FlightPlanDepartureName (string) [Get]

Name of the Departure Airport.

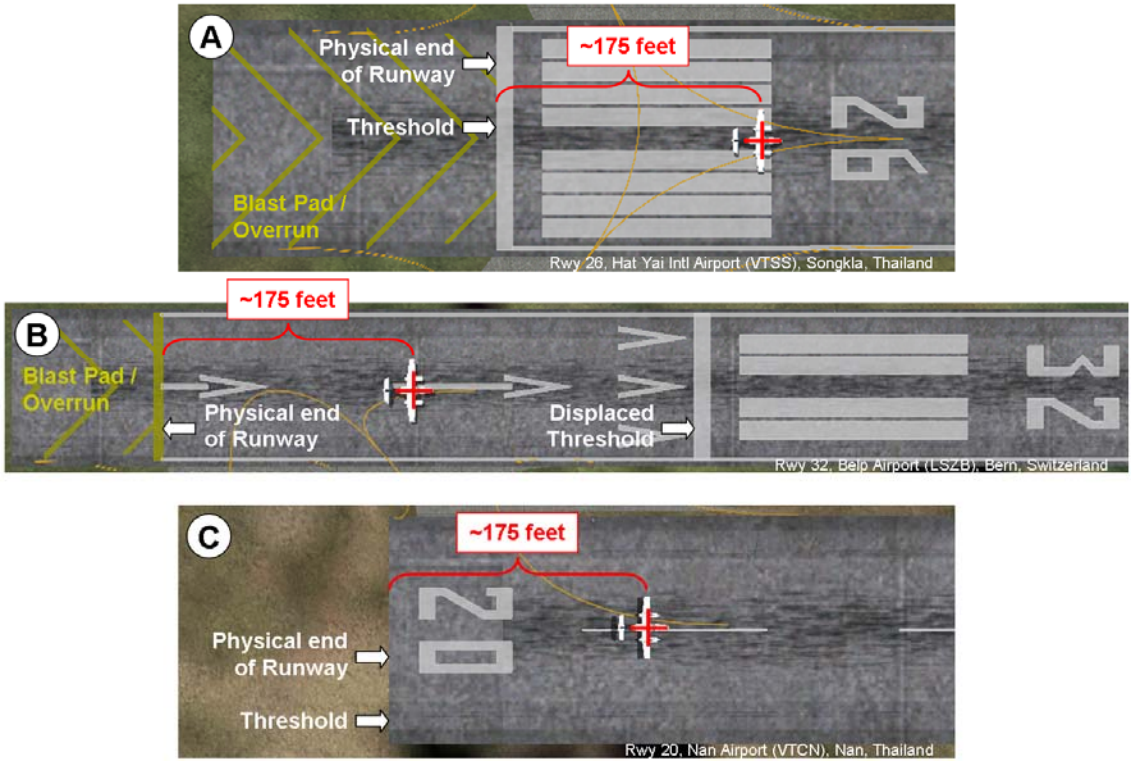
❑ FlightPlanDestinationAirportIdent (string) [Get]

The 3 to 4 character Destination airport Ident.

❑ **FlightPlanDestinationLatitude**
❑ **FlightPlanDestinationLongitude (degrees, radians) [Get]**

Latitude and Longitude of a "destination point" at the destination airport approximately 175 feet beyond the physical end of the approach end of the active runway, as shown below. Although landing is not permitted before a displaced threshold, it appears that for the sake of simplicity in the fs9gps world, the departure and destination points of the active runway are coincident, even in the case of displaced thresholds. Units are degrees (decimal format, not deg, min, sec) or radians.

FlightPlanDestinationLatitude, FlightPlanDestinationLongitude



Departure/Destination and Runway Approach Waypoints (△) are co-located with FlightPlanDestinationLatitude and Longitude at airports having an Instrument Approach Procedure.

❑ **FlightPlanDestinationAltitude (feet) [Get]**

Altitude (asl) of the destination point.

❑ **FlightPlanDestinationName (string) [Get]**

Destination airport Name.

The following table compares fs9gps variables to the equivalent entries of the Flight Plan .PLN file for a flight from Pochentong Airport (Phnom Penh International), Phnom Penh, Cambodia direct Changi Airport, Singapore.

fs9gps	FS9 Flight Plan IFR Pochentong Intl to Changi.PLN
	1. [flightplan]
	2. AppVersion=9.1.40901
FlightPlanTitle	3. title=VDPP to WSSS
FlightPlanDescription	4. description=VDPP, WSSS
FlightPlanFlightPlanType	5. type=IFR
FlightPlanRouteType	6. routetype=0
FlightPlanCruisingAltitude	7. cruising_altitude=34000
See id detail below	8. departure_id=VDPP, N11* 32.52', E104* 51.16', +000040.00
	9. departure_position=PARKING 1
See id detail below	10. destination_id=WSSS, N1* 19.75', E103* 59.11', +000021.99
FlightPlanDepartureName	11. departure_name=Pochentong Intl
FlightPlanDestinationName	12. destination_name=Changi
Flight Play Waypoint Info	13. waypoint.0=, VDPP, , VDPP, A, N11* 32.52', E104* 51.16', +000040.00,
	14. waypoint.1=, WSSS, , WSSS, A, N1* 19.75', E103* 59.11', +000021.99,

departure_id=VDPP, N11* 32.52', E104* 51.16', +000040.00

FlightPlanDepartureAirportIdent	VDPP
FlightPlanDepartureLatitude	N11* 32.52'
FlightPlanDepartureLongitude	E104* 51.16'
FlightPlanDepartureAltitude	+000040.00

And the associated fs9gps values:

```
GPS Viewer 1.2                  11111111112
GET SET SLEN 12345678901234567890 STRING NUMBER
 G       12  VDPP to WSSS         FlightPlanTitle, string
 G       10  VDPP, WSSS           FlightPlanDescription, string
 G        4  VDPP                 FlightPlanDepartureAirportIdent, string
 G       15  Pochentong Intl      FlightPlanDepartureName, string
 G        6     11.5420785        FlightPlanDepartureLatitude, degrees
 G        6    104.8526934        FlightPlanDepartureLongitude, degrees
 G        6   40.0000004          FlightPlanDepartureAltitude, feet
 G        4   WSSS                FlightPlanDestinationAirportIdent, string
 G        6   Changi              FlightPlanDestinationName, string
 G        6      1.3292220        FlightPlanDestinationLatitude, degrees
 G        6    103.9851643        FlightPlanDestinationLongitude, degrees
 G        5   21.998038           FlightPlanDestinationAltitude, feet
 G        0                       FlightPlanAlternateAirportIdent, string
 G        0                       FlightPlanAlternateName, string
 G        2      0.0000000        FlightPlanAlternateLatitude, degrees
 G        2      0.0000000        FlightPlanAlternateLongitude, degrees
 G        2   0.000000            FlightPlanAlternateAltitude, feet
```

- **FlightPlanAlternateAirportIdent (string) [Get]**
- **FlightPlanAlternateLatitude (degrees) [Get]**
- **FlightPlanAlternateLongitude (degrees) [Get]**
- **FlightPlanAlternateAltitude (feet) [Get]**
- **FlightPlanAlternateName (string) [Get]**

With the FlightPlanAlternate variables, an alternate airport or waypoint destination can be identified in the Flight Plan file. FS9 Flight Planner lacks the capability to create the Alternate, and the fs9gps FlightPlanAlternate variables are Get only, so the user must hand edit the .PLN file or use a third-party Flight Planner (if one exists that does this, I don't know) to define the Alternate.

I've added lines 13 and 14 to the .PLN file to define the Alternate Airport. For Changi, it is Hang Nadim Airport (Ident = WIKB), Batam, Indonesia, as shown below. I must obtain the airport Ident, Latitude, Longitude, Altitude and Name independently beforehand, and add it to the .PLN file.

fs9gps		**FS9 Flight Plan**
		IFR Pochentong Intl to Changi.PLN
	1.	[flightplan]
	2.	AppVersion=9.1.40901
FlightPlanTitle	3.	title=VDPP to WSSS
FlightPlanDescription	4.	description=VDPP, WSSS
FlightPlanFlightPlanType	5.	type=IFR
FlightPlanRouteType	6.	routetype=0
FlightPlanCruising Altitude	7.	cruising_altitude=34000
See id detail below	8.	departure_id=VDPP, N11* 32.52', E104* 51.16', +000040.00
	9.	departure_position=PARKING 1
See id detail below	10.	destination_id=WSSS, N1* 19.75', E103* 59.11', +000021.99
FlightPlanDepartureName	11.	departure_name=Pochentong Intl
FlightPlanDestinationName	12.	destination_name=Changi
See id detail below	13.	**alternate_id=WIKB, N1* 07.13', E104* 06.85', +000126.99**
FlightPlanAlternateName	14.	**alternate_name=Hang Nadim**
Flight Play Waypoint Info	15.	waypoint.0=, VDPP, , VDPP, A, N11* 32.52', E104* 51.16', +000040.00,
	16.	waypoint.1=, WSSS, , WSSS, A, N1* 19.75', E103* 59.11', +000021.99,

alternate_id=WIKB, N1* 07.13', E104* 06.85', +000126.99

FlightPlanAlternateAirportIdent	WIKB
FlightPlanAlternateLatitude	N1* 07.13'
FlightPlanAlternateLongitude	E104* 06.85'
FlightPlanAlternateAltitude	+000126.99

The fs9gps values showing the Alternate Airport that has been added:

```
GPS Viewer 1.2                  11111111112
GET SET SLEN 12345678901234567890 STRING NUMBER
 G        12  VDPP to WSSS          FlightPlanTitle, string
 G        10  VDPP, WSSS            FlightPlanDescription, string
 G        4   VDPP                  FlightPlanDepartureAirportIdent, string
 G        15  Pochentong Intl       FlightPlanDepartureName, string
 G        6     11.5419998          FlightPlanDepartureLatitude, degrees
 G        6    104.8526665          FlightPlanDepartureLongitude, degrees
 G        6   40.000004             FlightPlanDepartureAltitude, feet
 G        4   WSSS                  FlightPlanDestinationAirportIdent, string
 G        6   Changi                FlightPlanDestinationName, string
 G        6     1.3291667           FlightPlanDestinationLatitude, degrees
 G        6    103.9851664          FlightPlanDestinationLongitude, degrees
 G        6   21.000009             FlightPlanDestinationAltitude, feet
 G        0     WIKB                FlightPlanAlternateAirportIdent, string
 G        0   Hang Nadim            FlightPlanAlternateName, string
 G        6     1.1188332           FlightPlanAlternateLatitude, degrees
 G        6    104.1141665          FlightPlanAlternateLongitude, degrees
 G        6   126.000001            FlightPlanAlternateAltitude, feet
```

The FlightPlanAlternate variables are not truly *functional*, however. They don't "do" anything. If desired, an Airport ICAO could be easily constructed from the AlternateAirportIdent as follows:

```
'A_ _ _ _ _ _'
(C:fs9gps:FlightPlanAlternateAirportIdent) scat
```

The first line is an A followed by 6 spaces. The xml above yields 'A WIKB', which is the full ICAO for Hang Nadim Airport. I suppose one could then use it to specify a new approach Airport. In a similar manner, FlightPlanAlternateLatitude and Longitude could be used to specify a new Waypoint that could be added to the Flight Plan.

However, when coding a gps or flight management computer from scratch, I guess it would be easier and more realistic to enter the alternate destination (doesn't have to be an Airport – it could be a Transition Waypoint) Ident while Flight Sim is running, storing the Ident as chr >L:Vars for later use as appropriate.

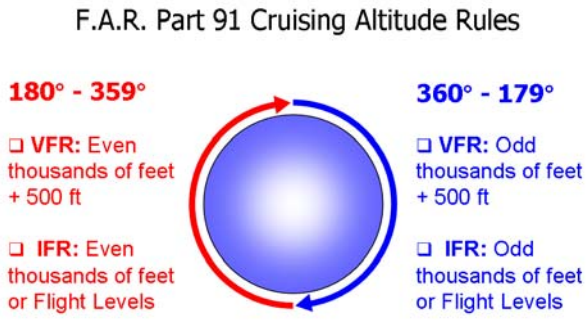
FlightPlanCruisingAltitude DISCUSSION

FlightPlanCruisingAltitude is read only and a separate flight planning application such as the stock FS9 Flight Planner is needed if the user wants a cruising altitude to be computed automatically. Charts and manual entry within FS9 Flight Planner works as well, of course.

Given user inputs of 1) departure airport, 2) destination airport, 3) route type (Direct-GPS, Low Altitude Airways, High Altitude Airways, or VOR to VOR), and 4) flight plan type (VFR or IFR), FS9's Flight Planner determines a flight plan route and then computes a single cruising altitude. The associated gps variables, DepartureAirportIdent, DestinationAirportIdent, RouteType, and FlightPlanType are also read-only.

A detailed discussion of FS9 Flight Planner is beyond the scope of this GPS Guidebook. I'm not certain of the rules that determine exactly how Flight Planner computes Cruising Altitude, however, here are a few Flight Planner notes based on limited observations:

- U.S. F.A.R. Part 91 Cruising Altitude rules are applied world-wide. Above 18,000 feet altitude where, in U.S.A. airspace, flight is governed by Instrument Flight Rules, FS9 Flight Planner still adds 500 feet to the cruising altitude of a flight that is set up as a VFR flight.



- FS9 Flight Planner 'scans' at least a coarse terrain grid along its computed flight route and then provides an amount of clearance above the maximum ground elevation in the calculation of Cruising Altitude.

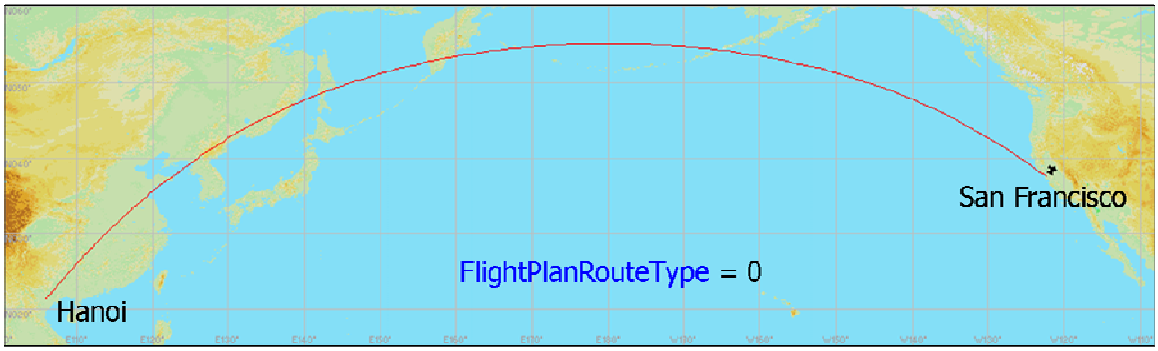
- Ground clearance rules appear to vary geographically. Ground clearance in the USA seems to be 3000 to 4000 feet above highest ground elevation. Flying from France to Italy over the Alps, 7000 feet. Italy to Austria, 4000 feet. Thailand, 3000 feet. And so on. I have not figured out how to predict the value.

- Cruising Altitude often exceeds even the maximum airway segment MEA for flight routes along low altitude Airways.

- FS9 Flight Planner calculates Cruising Altitude once the "Find Route" button is clicked. Subsequent changes to the flight route by adding or deleting waypoints will not automatically re-compute the Cruising Altitude, even if it should.

- Restricted Airspace sometimes influences FS9 Flight Planner's Cruising Altitude. The effect seems to be inconsistent, however, so again, I am not sure.

- As everyone knows, Flight Planner calculates GPS-Direct routes along Great Circle paths.



In summary, when it comes to FS9 Flight Planner's rules for computing CruisingAltitude, I am guessing. At any rate, after you consult your charts, FlightPlanCruisingAltitude is manually settable from within FS9 Flight Planner and can be changed during flight by request to ATC.

FLIGHT PLAN STATUS VARIABLES

❑ FlightPlanIsActiveFlightPlan (bool) [Get]

FlightPlanIsActiveFlightPlan =1 means a flight plan is loaded.

❑ FlightPlanIsLoadedApproach (bool) [Get]

FlightPlanIsLoadedApproach - an approach is loaded (flight plan may/may not be loaded)

❑ FlightPlanIsActiveApproach (bool) [Get, Set]

FlightPlanIsActiveApproach is used to activate a loaded approach. The xml:

```
1 (>C:fs9gps:FlightPlanIsActiveApproach)
```

It requires an argument. 1 for activate, as above. 0 means approach is loaded, but not activated.

❑ FlightPlanIsActiveWaypoint (bool) [Get, ~~Set~~]

FlightPlanIsActiveWaypoint is a bool representing whether or not any Waypoint in the Flight Plan is active. At the departure airport and en route, one waypoint is always active. The status changes upon reaching the destination waypoint, however, at which point FlightPlanIsActiveWaypoint becomes 0.

FlightPlanIsActiveWaypoint is 1 as long as there is a Flight Plan or Approach waypoint that is active.

FlightPlanIsActiveWaypoint appears to be read-only, contrary to the SDK description.

❑ FlightPlanIsDirectTo (bool) [Get, ~~Set~~]

FlightPlanIsDirectTo is a bool representing Direct To status. 1 = Flight Plan is Direct To. 0 = Not Direct To.

FlightPlanIsDirectTo appears to be read-only, contrary to the SDK description.

❑ FlightPlanDirectToWaypoint (enum) [Get]

DirectToWaypoint is the Flight Plan Waypoint Index that the Direct To points to. Because a DirectTo Flight Plan is (apparently) always a two-waypoint Flight Plan, then the only logical DirectToWaypoint value is 1. Indeed, DirectToWaypoint is always 1 when the Flight Plan is DirectTo, or -1 when it is not.

❑ FlightPlanActiveWaypoint (enum) [Get, Set]

FlightPlanActiveWaypoint is the Index number of the waypoint that the aircraft is currently flying directly toward or on an intercept course toward; it's the next waypoint. The active Flight Plan *leg* is the flight path from the previous waypoint to the active Waypoint.

In the stock gps_500 gauge, the Ident of the ActiveWaypoint, if an Ident exists, is displayed in magenta text and the line color of the active Flight Plan leg is also magenta. In the FS9 Map, the line color of the active Flight Plan leg is magenta.

FlightPlanActiveWaypoint is read *and write* capable, contrary to the SDK description.

❑ FlightPlanActiveApproachWaypoint (enum) [Get]

FlightPlanActiveApproachWaypoint is the index number of the active (current) approach segment. ApproachWaypoint does not change when a sub-segment changes – only when the segment changes.

❑ FlightPlanIsActiveWaypointLocked (bool) [Get, Set]

When set to 1, FlightPlanIsActiveWaypointLocked locks the ActiveWaypoint's Index number so that it cannot advance by 1 when the aircraft reaches the Active Waypoint. This has significant consequences – it effectively terminates the Flight Plan at the ActiveWaypoint until and unless ActiveWaypointLocked is set to zero.

If the aircraft is being controlled by a typical autopilot, it will turn 360° upon reaching the locked Waypoint, circling around to repeatedly cross it. The same occurs when an aircraft reaches the destination point of a Flight Plan but does not land (I am referring to *Flight Plan* – not an Approach with a Missed Approach Procedure).

FlightPlanIsActiveWaypointLocked can be set to one or zero at any point in time, locking the ActiveWaypoint or unlocking it and allowing the Flight Plan to continue as normal. When a Flight Plan is opened, FlightPlanIsActiveWaypointLocked is set to zero. Perhaps it is possible outside of xml to create a flight pan with a locked ActiveWaypoint, but I don't know what purpose that would serve.

Note: When adding or deleting Waypoints *beyond* the ActiveWaypoint, fs9gps sets ActiveWaypointLocked to 1, so you may want to subsequently reset it to zero unless you want the Flight Plan to terminate at the ActiveWaypoint.

An example of the use and effects of FlightPlanIsActiveWaypointLocked is given in Example 3 of this section.

❑ FlightPlanWaypointsNumber (enum) [Get]

FlightPlanWaypointsNumber is the total number of waypoints in the flight plan. The Departure airport is always the first waypoint and has Index = 0. The total number of flight plan legs is FlightPlanWaypointsNumber minus 1.

NEWWAYPOINT GROUP: CREATING AND EDITING A FLIGHT PLAN

CREATING A FLIGHT PLAN WITH XML

Flight Plans can be easily created using xml, but unfortunately, not saved owing to a current lack of file I/O capability in the xml environment. Even saving the Flight after editing the flight plan using the NewWaypoint variables will not save the Flight Plan; the gps engine will reject the edited Flight Plan when the Flight is loaded.

To create a new Flight Plan when one is not currently loaded, FlightPlanDirectToDestination is used. Use of the FlightPlanDirectToDestination variable is reviewed later in this section.

Of course, an easy way of editing *and saving* a flight plan by adding or deleting existing navaid or published intersection waypoints is with FS9's built-in Flight Planner. With the Find Route map open, just click on the flight path between any two waypoints and drag it to the new waypoint facility. To delete a waypoint, select the waypoint from the waypoint list to the right of the map, then click "Delete". Then "Save". Very easy.

EDITING A FLIGHT PLAN

The NewWaypoint variables are a small group of Set-only variables that can be used to edit flight plans by adding or deleting en route or alternate destination waypoints, one waypoint at a time.

Adding a new waypoint to an active flight plan is a two step process that involves defining the latitude and longitude of the new waypoint to be added, followed by assigning the waypoint index of the new waypoint (where the new en route waypoint will be inserted in the Flight Plan).

Restating this, the required information for a new en route waypoint is:

1. Latitude and Longitude
2. New Waypoint Index position

A valid en route waypoint can be just a point on the map, not associated with an existing navaid or fs9gps waypoint. Fs9gps will assign WaypointType = 5 (User) to any waypoint added using AddWaypoint that is not an existing navaid or published waypoint.

ENTERING NEW WAYPOINT LATITUDE AND LONGITUDE

There are a few approaches to this:

1. Enter the new waypoint lat and lon directly (necessary for user-defined waypoints)
2. Enter the new waypoint ICAO, from which lat and lon will automatically be accessed by fs9gps

    a. Enter the 12 character ICAO directly (usually not very realistic).

    b. Enter the new waypoint facility (airport, navaid or intersection) Ident followed by an ICAO search that determines the unique ICAO, from which lat and lon will be automatically accessed.

    c. For airport waypoints, enter the airport Name followed by Name Search which can be used to find the airport ICAO.

❑ FlightPlanNewWaypointLatitude
❑ FlightPlanNewWaypointLongitude (degrees) [Set]

Latitude and Longitude of the waypoint to be added. Either NewWaypointLatitude and Longitude, or NewWaypointICAO must be entered before AddWaypoint is executed.

```
51.3278 (>C:fs9gps:FlightPlanNewWaypointLatitude, degrees)
 7.1770 (>C:fs9gps:FlightPlanNewWaypointLongitude, degrees)
```

or

```
'VED    BAM' (>C:fs9gps:FlightPlanNewWaypointICAO)
```

Both will define a new en route waypoint at the same location.

❑ FlightPlanNewWaypointICAO (string) [Set]

FlightPlanNewWaypointICAO can be entered instead of Latitude and Longitude. From the ICAO, fs9gps will automatically access the Ident, Waypoint Type, Latitude and Longitude of the new waypoint. Waypoint Altitude will not be returned because, at least in FS9, you must enter the related facility Group to access altitude.

❑ FlightPlanNewWaypointIdent (string) [Set]

Asking the user to enter the 12 character ICAO isn't a completely realistic sim experience. Instead, entering the Ident of a navaid, intersection, or airport is a more real-world procedure. The ultimate objective is still to define the Lat and Lon of the new waypoint.

Because Idents other than airports are not always unique, using NewWaypointIdent will require an ICAO Search to isolate the correct ICAO from which Lat and Lon will be accessed.

The ICAO Search process is more straight-forward than it may sound and is discussed in the ICAO Search chapter. If the search goal is to display the list of ICAOs containing the specified Ident that the user can select from, then the ICAOSearch code will be quite simple.

Once ICAO Search is complete and the desired ICAO selected, then all that remains is to transfer the ICAOSearchCurrentIcao to FlightPlanNewWaypointIcao:

```
(@c:IcaoSearchCurrentIcao) (>@c:FlightPlanNewWaypointIcao)
```

A more complete xml example is included in the ICAO Search Data Group chapter.

ASSIGNING A WAYPOINT INDEX AND ADDING THE NEW WAYPOINT

❑ FlightPlanAddWaypoint (enum) [Set]

FlightPlanAddWaypoint is used to add a single, additional waypoint to a loaded flight plan, one added waypoint at a time. It requires an argument (index pointer) to indicate where in the Flight Plan the new waypoint will be inserted.

```
2 (>C:fs9gps:FlightPlanAddWaypoint)
```

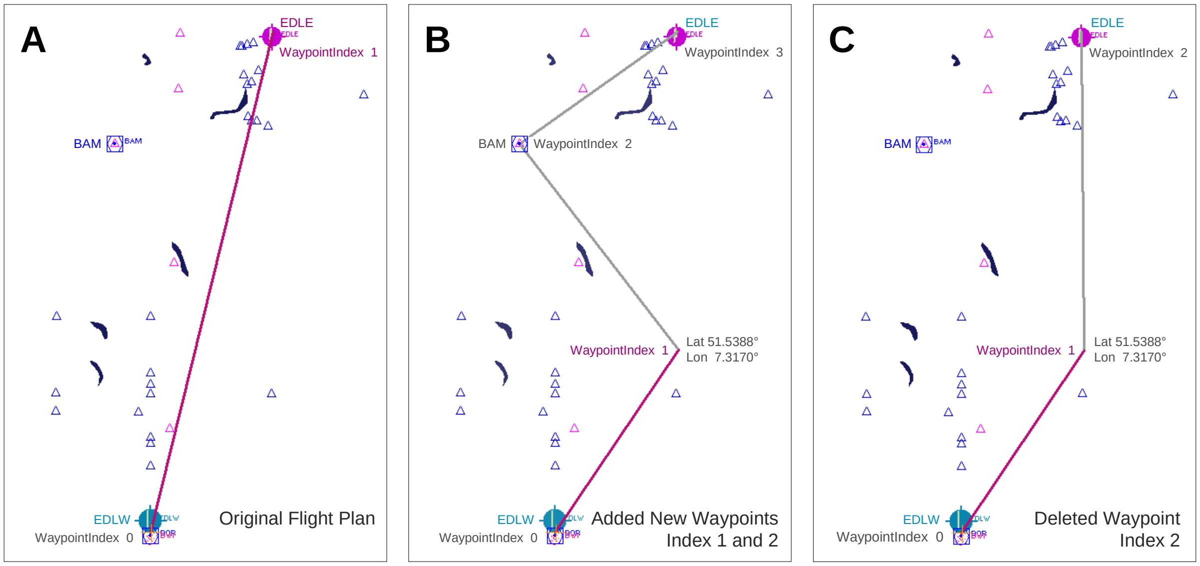
will add a new waypoint at WaypointIndex 2 with whatever Lat and Lon had previously been specified using NewWaypointLatitude and Longitude, or NewWaypointICAO. The waypoint previously at WaypointIndex 2 is advanced to become WaypointIndex 3. 3 becomes 4, and so forth. New waypoints can only be added/deleted to Flight Plans, not to Approaches.

❑ FlightPlanDeleteWaypoint (enum) [Set]

FlightPlanDeleteWaypoint is used to delete a single waypoint from a loaded flight plan. One waypoint delete at a time. It requires an argument (index pointer) to indicate which waypoint is to be deleted.

Example 1: FlightPlanAddWaypoint and FlightPlanDeleteWaypoint

The following example demonstrates FlightPlanAddWaypoint and FlightPlanDeleteWaypoint:



Map **A** shows a Direct To routing from Dortmund Airport, Dortmund Germany to Essen-Mülheim Airport, Essen/Mülheim Germany. The table below lists the FlightPlanWaypoint variables:

EDIT FLIGHT PLAN

FlightPlanTitle: EDLW to EDLE
2 :FlightPlanWaypointsNumber   1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType   2 :FlightPlanFlightPlanType

-- FlightPlanWaypoint --

Idx	ICAO 111 123456789012		Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Dist Ttl	Dist Rem	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A	EDLW	EDLW	424	1	0	51.52330	7.62637	0.00	0.0	27.1	0.0	0.00	0.00	0.00	12.61	240.3	0.0	0.0
1	A	EDLE	EDLE	424	1	256	51.40018	6.92987	27.07	27.1	0.0	26.7	8.12	0.00	14.63	12.86	0.0	5.9	0.0

Next, two new waypoints are added using NewWaypoint variables. The xml:

```
<!-- The first new Waypoint -->
   'VED    BAM' (>C:fs9gps:FlightPlanNewWaypointICAO)
   1 (>C:fs9gps:FlightPlanAddWaypoint)

<!-- The second new Waypoint -->
   51.5388 (>C:fs9gps:FlightPlanNewWaypointLatitude, degrees)
   7.3170 (>C:fs9gps:FlightPlanNewWaypointLongitude, degrees)
   1 (>C:fs9gps:FlightPlanAddWaypoint)
```

The new routing is shown in Map **B**, and the new FlightPlanWaypoint variable list is shown below. The red outline highlights the new waypoints. Note that the User-defined waypoint, WaypointIndex 1 (just a lat, lon position and not an existing fs9gps facility), does not have an ICAO. Note also that both new waypoints are added using the same argument (i.e., 1) for FlightPlanAddWaypoint. When the second waypoint is added also as Waypoint 1, the previously existing Waypoint 1 is automatically advanced to become Waypoint 2. 2 becomes 3, and so forth.

EDIT FLIGHT PLAN

FlightPlanTitle: EDLW to EDLE
4 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType 2 :FlightPlanFlightPlanType

FlightPlanWaypoint

Idx	ICAO	111 123456789012	Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Dist Ttl	Dist Rem	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A	EDLW	EDLW	424	1	0	51.52330	7.62637	0.00	0.0	35.5	0.0	0.00	0.00	0.00	0.00	240.2	0.0	0.0
1				0	5	276	51.53880	7.31700	11.59	11.6	23.9	11.6	3.47	0.00	0.00	0.00	0.0	2.5	0.0
2	VED	BAM	BAM	0	3	204	51.32776	7.17699	13.70	25.3	10.2	13.7	4.10	0.00	0.00	0.00	0.0	3.0	0.0
3	A	EDLE	EDLE	424	1	297	51.40018	6.92987	10.23	35.5	0.0	10.2	3.07	0.00	0.00	0.00	0.0	2.2	0.0

Upon executing AddWaypoint, fs9gps automatically updates waypoint index, magnetic heading, distances, ETEs, ETAs and fuel variables for all affected waypoints.

Finally, WaypointIndex 2 (BAM VOR-DME) is deleted using FlightPlanDeleteWaypoint. The xml:

```
2 (>C:fs9gps:FlightPlanDeleteWaypoint)
```

The new routing is shown in Map **C**, and the final FlightPlanWaypoint variable list is shown below:

EDIT FLIGHT PLAN

FlightPlanTitle: EDLW to EDLE
3 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType 2 :FlightPlanFlightPlanType

FlightPlanWaypoint

Idx	ICAO	111 123456789012	Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Dist Ttl	Dist Rem	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A	EDLW	EDLW	424	1	0	51.52333	7.62633	0.00	0.0	28.3	0.0	0.00	0.00	0.00	0.00	241.9	0.0	0.0
1				0	5	276	51.53880	7.31700	11.58	11.6	16.7	11.6	3.47	0.00	0.00	0.00	0.0	2.5	0.0
2	A	EDLE	EDLE	424	1	242	51.40017	6.92983	16.69	28.3	0.0	16.7	5.00	0.00	0.00	0.00	0.0	3.6	0.0

❑ FlightPlanDirectToDestination (bool) [Set]

FlightPlanDirectToDestination will create a new, two-waypoint Flight Plan originating at the aircraft's current x,y,z position and culminating at the latitude and longitude defined by FlightPlanNewWaypointLatitude and Longitude, or FlightPlanNewWaypointICAO.

If no Flight Plan is currently loaded, FlightPlanDirectToDestination will create a new two-waypoint Flight Plan. If a Flight Plan is currently active, FlightPlanDirectToDestination will replace the entire Flight Plan with the new two-waypoint one.

The current aircraft location will become WaypointIndex 0. FlightPlanWaypointAltitude will be set to the current aircraft altitude and FlightPlanWaypointType will be 5 (User). No ICAO will be associated with this waypoint.

The Direct To location can be any latitude and longitude; it does not have to be an fs9gps navaid facility or intersection, nor a waypoint currently in the Flight Plan. However, FlightPlanNewWaypointLatitude and Longitude, or FlightPlanNewWaypointICAO must be defined immediately preceding the FlightPlanDirectToDestination statement as follows.

Direct To a custom, user-defined lat and lon:

```
37.8487 (>C:fs9gps:FlightPlanNewWaypointLatitude, degrees)
-97.8157 (>C:fs9gps:FlightPlanNewWaypointLongitude, degrees)
(>C:fs9gps:FlightPlanDirectToDestination)
```

L:Vars could be substituted for the numbers, of course:

```
(L:DTO_Lat, degrees) (>@c:FlightPlanNewWaypointLatitude, degrees)
```

FlightPlanDirectToDestination does not require an argument.

Direct To a Waypoint in the Flight Plan:

```
(L:DTOWaypointIndex, enum) (>@c:FlightPlanWaypointIndex)
(@c:FlightPlanWaypointICAO) (>@c:FlightPlanNewWaypointICAO)
(>@c:FlightPlanDirectToDestination)
```

In the example above, `L:DTOWaypointIndex` is a user-specified Flight Plan waypoint index number. It is entered into WaypointIndex from which WaypointICAO is determined. From there, the ICAO transfer into NewWaypointICAO will provide the latitude and longitude the DirectToDestination statement needs.

Note that this approach requires that the waypoint have an ICAO. All fs9gps facilities (airports, navaids, waypoints/intersections) have a unique ICAO, but if the flight plan contains a custom, user-defined waypoint, that waypoint will not have an ICAO.

Direct To any fs9gps Facility:

This approach requires that the facility ICAO be determined as the first step. Any source of the ICAO will do:

- ICAOSearchCurrentICAO
- NameSearchCurrentICAO
- Waypoint or Facility Group ICAOs such as WaypointAirportICAO
- Nearest Group ICAOs such as NearestVorCurrentICAO

This is followed by the ICAO transfer into NewWaypointICAO and, finally, the DirectToDestination statement:

```
(@c:NearestVorCurrentICAO) (>@c:FlightPlanNewWaypointICAO)
(>@c:FlightPlanDirectToDestination)
```

Note: Following execution of FlightPlanDirectToDestination, it appears that use of gps variables that attempt to access the ICAO or Ident of either of the DirectTo Waypoints, or properties of those Waypoints, may cause the simulation to crash.

As an example, the following operations will cause the sim to crash if they are preceeded by execution of FlightPlanDirectToDestination:

- `((C:fs9gps:FlightPlanDescription) slen)`
- `((C:fs9gps:FlightPlanDepartureAirportIdent) slen)`
- `((C:fs9gps:FlightPlanDepartureName) slen)`
- `((C:fs9gps:FlightPlanDestinationAirportIdent) slen)`
- `((C:fs9gps:FlightPlanDestinationName) slen)`

However, use of those variables without slen will just return an empty string without causing a crash.

Admittedly, I do not yet understand the issue very well and there obviously is more going on than I realize. I suspect that other operations will also cause a sim crash. ***Suffice it to say that I have found that a simulation crash can easily occur following FlightPlanDirectToDestination***, so be on the alert.

❑ FlightPlanCancelDirectTo (bool) [Set]

FlightPlanCancelDirectTo restores the Flight Plan to the state prior to execution of FlightPlanDirectToDestination. However, if the Flight Plan is changed (AddWaypoint or DeleteWaypoint) after DirectToDestination is executed, then FlightPlanCancelDirectTo will no longer be able to restore the Flight Plan to the prior state.

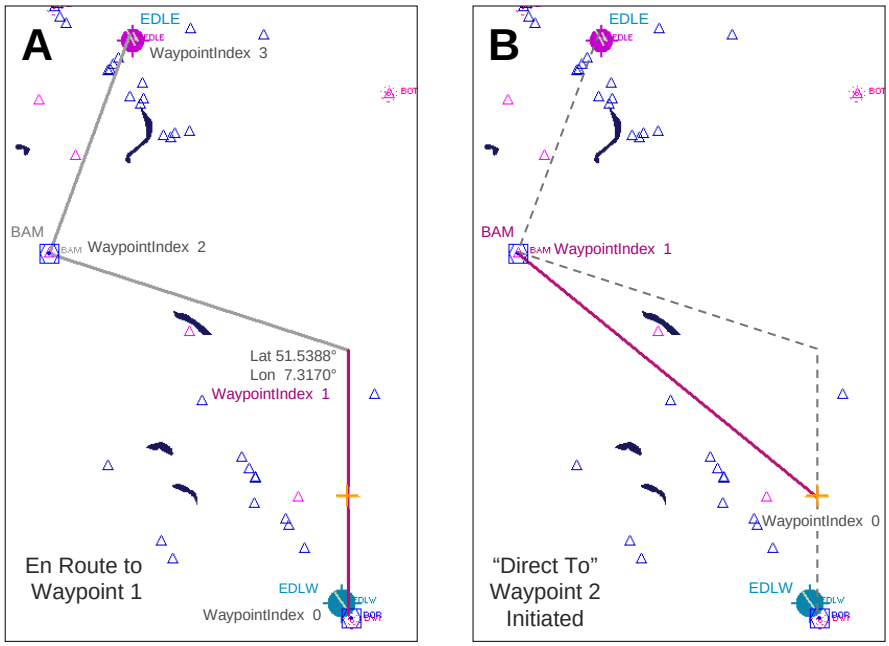
The xml:

```
(>C:fs9gps:FlightPlanCancelDirectTo)
```

FlightPlanCancelDirectTo does not require an argument.

Example 2: FlightPlanDirectToDestination and CancelDirectTo

The following example demonstrates FlightPlanDirectToDestination and CancelDirectTo:



Map **A** shows flight progress as the aircraft is en route to waypoint 1. The aircraft position is indicated with the orange colored + symbol. The table below lists the FlightPlanWaypoint variables.

```
EDIT FLIGHT PLAN

FlightPlanTitle: EDLW to EDLE
   4 :FlightPlanWaypointsNumber   1 :FlightPlanActiveWaypoint
   0 :FlightPlanRouteType         2 :FlightPlanFlightPlanType
```

FlightPlanWaypoint

Idx	ICAO 111	ICAO 123456789012	Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Dist Ttl	Dist Rem	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A	EDLW	EDLW	424	1	0	51.52333	7.62633	0.00	0.0	35.5	0.0	0.00	0.00	0.00	12.80	240.9	0.0	0.0
1				0	5	276	51.53880	7.31700	11.58	11.6	23.9	6.4	3.47	0.00	2.47	12.87	0.0	2.5	0.0
2	VFD	BAM	BAM	0	3	204	51.32776	7.17699	13.70	25.3	10.2	13.7	4.10	0.00	5.30	12.96	0.0	3.0	0.0
3	A	EDLE	EDLE	424	1	297	51.40017	6.92983	10.23	35.5	0.0	10.2	3.07	0.00	3.95	13.03	0.0	2.2	0.0

Map **B** shows the Flight Plan immediately after a DirectToDestination Waypoint 2 is initiated.

The xml:

```
(L:DTOWaypointIndex, enum) (>@c:FlightPlanWaypointIndex)
(@c:FlightPlanWaypointICAO) (>@c:FlightPlanNewWaypointICAO)
(>@c:FlightPlanDirectToDestination)
```

where the user has entered 2 for L:DTOWaypointIndex. The table below lists the FlightPlanWaypoint variables. Note that the present aircraft position becomes the new Waypoint 0, the aircraft's current altitude (2964') becomes WaypointAltitude for Index 0,

and the WaypointType is 5 (User). The DirectTo Waypoint (original Waypoint 2, BAM VOR-DME) is now WaypointIndex 1, and WaypointMagneticHeading and Distance variables are adjusted. FlightPlanTitle is also updated to reflect the DTO.

```
EDIT FLIGHT PLAN

FlightPlanTitle: Direct to BAM
   2 :FlightPlanWaypointsNumber   1 :FlightPlanActiveWaypoint
   0 :FlightPlanRouteType         0 :FlightPlanFlightPlanType
```

FlightPlanWaypoint

Idx	ICAO 111 123456789012	Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Dist Ttl	Dist Rem	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0			2964	5	0	51.53029	7.48828	0.00	0.0	16.8	0.0	0.00	0.00	0.00	12.83	239.7	0.0	0.0
1	VED	BAM	0	3	226	51.32776	7.17699	16.83	16.8	0.0	16.8	5.03	0.00	6.50	12.94	0.0	3.7	0.0

Map **C** shows progress of the flight en route to the Direct To waypoint. At this point, a CancelDirectTo is intitated. The xml:

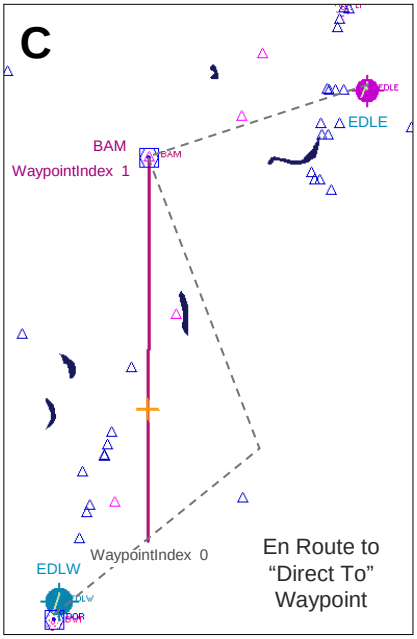
```
(>@c:FlightPlanCancelDirectTo)
```

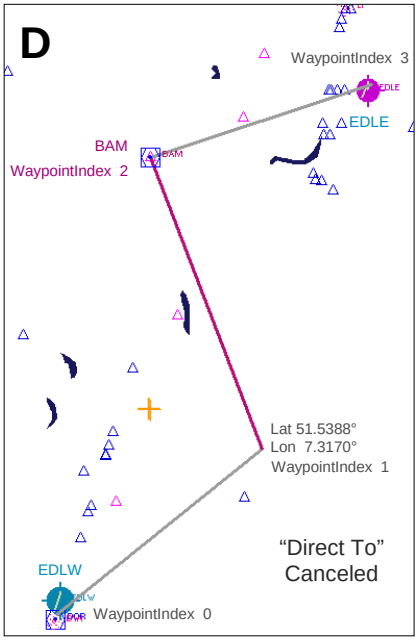
Map **D** reflects the Flight Plan immediately after CancelDirectTo, and the table below lists the FlightPlanWaypoint variables. Note that the original Flight Plan is restored and that distance and fuel variables associated with ActiveWaypoint 2 are updated. The ActiveWaypoint is now 2, and, if on autopilot, the aircraft will begin a right turn to intercept the Waypoint 1 to Waypoint 2 leg.

```
EDIT FLIGHT PLAN

FlightPlanTitle: EDLW to EDLE
   4 :FlightPlanWaypointsNumber   2 :FlightPlanActiveWaypoint
   0 :FlightPlanRouteType         2 :FlightPlanFlightPlanType
```

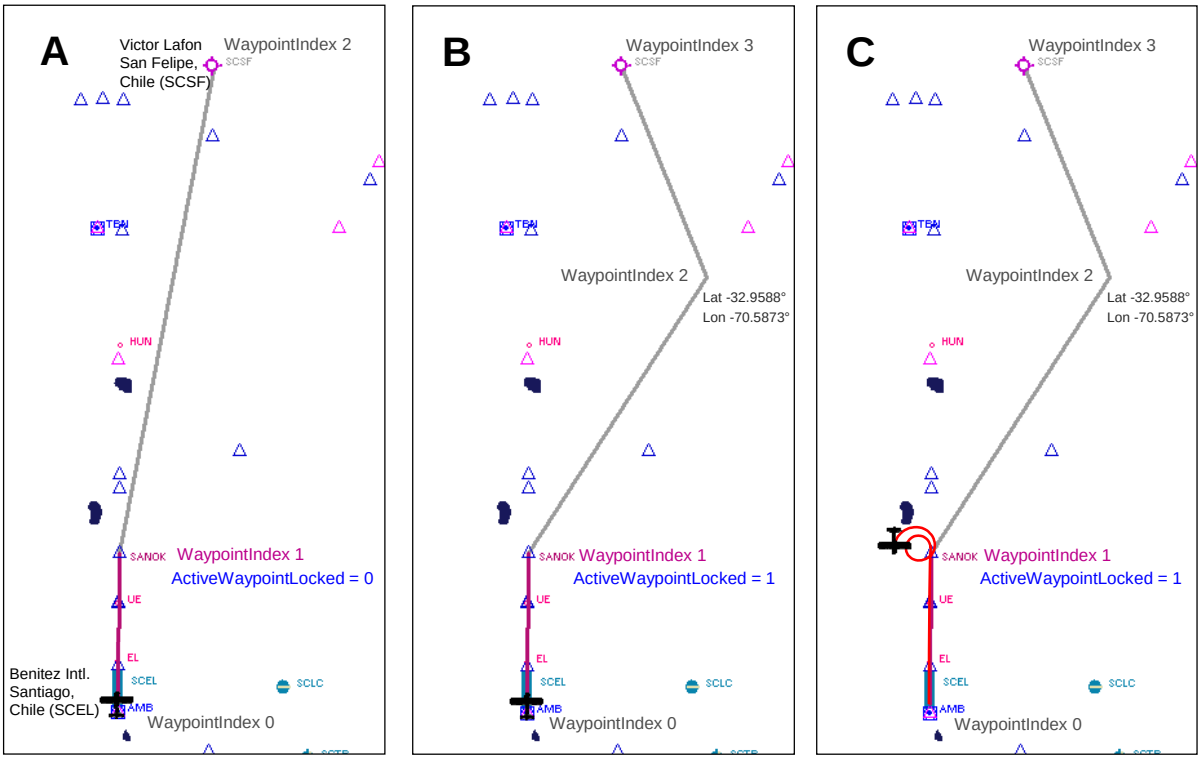
FlightPlanWaypoint

Idx	ICAO 111 123456789012	Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Dist Ttl	Dist Rem	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A	EDLW	EDLW	424	1	0	51.52333	7.62633	0.00	0.0	35.5	0.0	0.00	0.00	12.87	238.0	0.0	0.0
1				0	5	276	51.53880	7.31700	11.58	11.6	23.9	0.0	3.47	0.00	0.00	12.87	238.0	2.5	0.0
2	VED	BAM	BAM	0	3	204	51.32776	7.17699	13.70	25.3	10.2	5.3	4.10	0.00	3.40	12.92	0.0	3.0	0.0
3	A	EDLE	EDLE	424	1	297	51.40017	6.92983	10.23	35.5	0.0	10.2	3.07	0.00	3.15	12.98	0.0	2.2	0.0





Example 3: ActiveWaypointLocked, AddWaypoint and ActiveWaypoint



This example begins with a Flight Plan from Arturo Merino Benitez Intl. Airport (SCEL), Santiago, Chile, to Victor Lafon Airport (SCSF), San Felipe, Chile. It includes WaypointIndex 1, SANOK Intersection. The Flight Plan map is shown in Figure **A**, above, and the table below lists some of the Flight Plan variables. Note that ActiveWaypoint is 1 and ActiveWaypointLocked is 0. The Flight Plan was created in FS9's Flight Planner.

```
FLIGHT PLAN

FlightPlanTitle: SCEL to SCSF
       3 :FlightPlanWaypointsNumber          1 :FlightPlanActiveWaypoint     0 :FlightPlanIsActiveWaypointLocked
       1 :FlightPlanRouteType                2 :FlightPlanFlightPlanType     1 :FlightPlanIsActiveWaypoint

------------------------------------------- FlightPlanWaypoint -------------------------------------------
     ICAO      111                          Mag                                    Dist      Dist      Rem
Idx  123456789012  Ident   Alt Type  Hdg       Lat         Lon       Dist       Ttl       Rem      Dist      ETE
0    A       SCEL    SCEL  1554    1    0  -33.40935   -70.78492     0.00       0.0      39.9       0.0     0.00
1    WSCSCELSANOK  SANOK     0    2  352  -33.25284   -70.79138     9.40       9.4      30.5       9.4     2.82
2    A       SCSF    SCSF  2160    1    3  -32.74934   -70.70198    30.55      39.9       0.0      30.5     9.15
```

Figure **B** shows the results of adding a user-defined waypoint as new WaypointIndex 2 at Latitude S32° 57.53', Longitude W70° 35.238'. The xml:

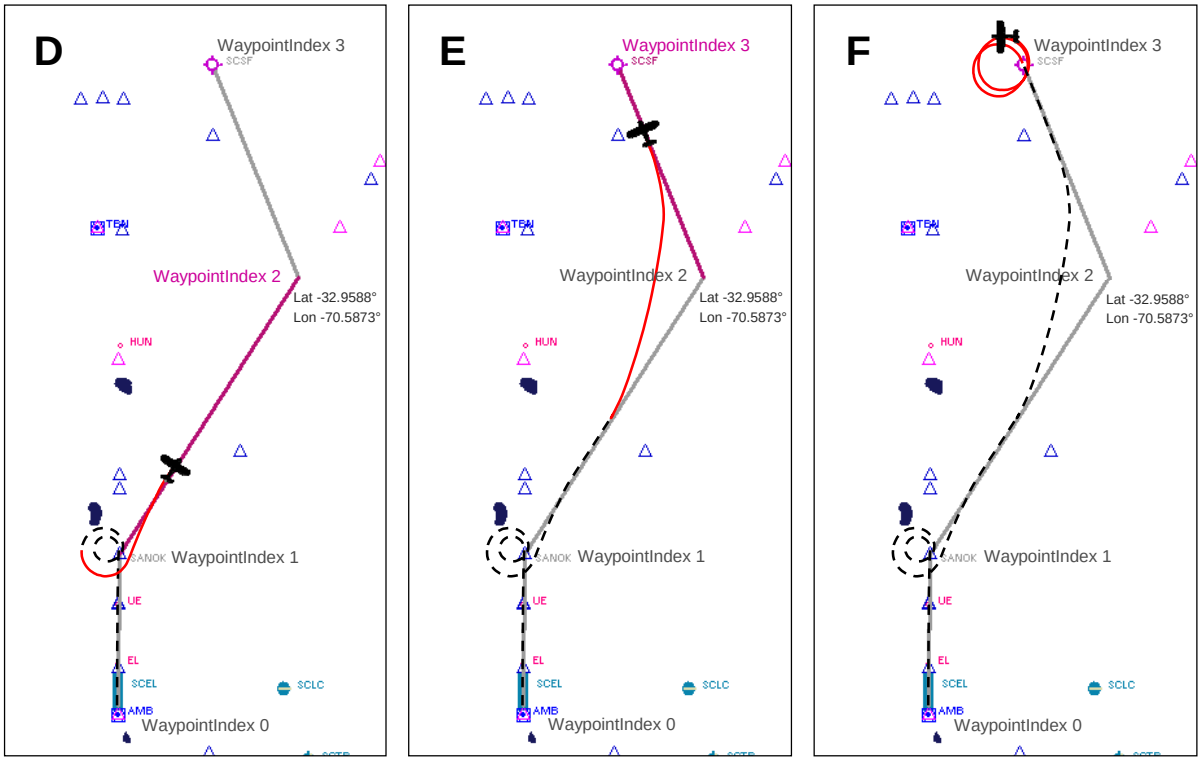
```
-32.95883 (>C:fs9gps:FlightPlanNewWaypointLatitude, degrees)
-70.58733 (>C:fs9gps:FlightPlanNewWaypointLongitude, degrees)
2 (>C:fs9gps:FlightPlanAddWaypoint)
```

Note from the table below that ActiveWaypointLocked is now 1 as a result of the AddWaypoint statement. The destination waypoint, SCSF, which had been WaypointIndex 2

has been automatically advanced to become WaypointIndex 3 and the associated distances and time have been adjusted to accommodate the new waypoint.

```
FLIGHT PLAN

FlightPlanTitle: SCEL to SCSF
       4 :FlightPlanWaypointsNumber          1 :FlightPlanActiveWaypoint     1 :FlightPlanIsActiveWaypointLocked
       1 :FlightPlanRouteType                2 :FlightPlanFlightPlanType     1 :FlightPlanIsActiveWaypoint

------------------------------------------- FlightPlanWaypoint -------------------------------------------
     ICAO     111                          Mag                                  Dist      Dist      Rem
Idx  123456789012  Ident   Alt Type  Hdg       Lat         Lon      Dist        Ttl       Rem       Dist      ETE
0    A      SCEL    SCEL  1554    1    0  -33.40935  -70.78492      0.00        0.0      43.6       0.0     0.00
1    WSCSCELSANOK  SANOK     0    2  352  -33.25284  -70.79138      9.40        9.4      34.2       9.4     2.82
2                             0    5   24  -32.95880  -70.58730     20.41       29.8      13.8      20.4     6.12
3    A      SCSF    SCSF  2160    1  330  -32.74934  -70.70198     13.83       43.6       0.0      13.8     4.13
```

Figure **C** shows the flight at Waypoint 1. Because ActiveWaypointLocked is 1, the Active Waypoint is locked and does not advance to the next Index number. The result is that the aircraft (controlled by an autopilot) keeps turning 360° to repeatedly intercept Waypoint 1.



Mid-way through the second 360° turn, ActiveWaypointLocked is reset to zero:

```
0 (>C:fs9gps:FlightPlanIsActiveWaypointLocked)
```

and as shown in Figure **D**, ActiveWaypoint advances to WaypointIndex 2, and the aircraft turns to intercept the flight leg to Waypoint 2.

At about the half way point to Waypoint 2 (Figure **E**), the ActiveWaypoint is changed to 3 by user input:

```
3 (>C:fs9gps:FlightPlanActiveWaypoint)
```

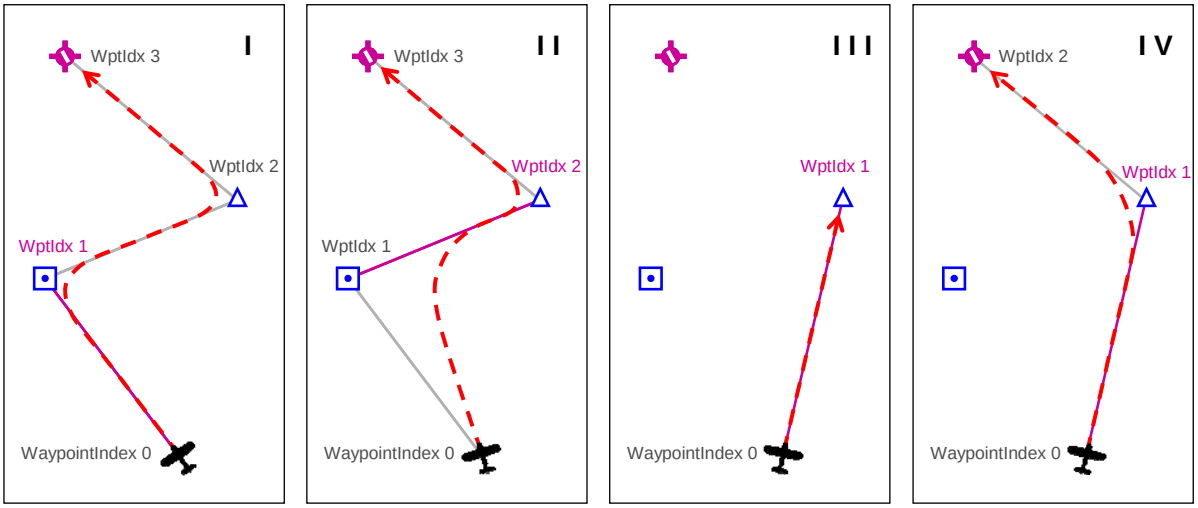
The aircraft now turns to intercept the flight leg from Waypoint 2 to Waypoint 3, bypassing Waypoint 2. The Flight Plan remains unchanged as shown in the table below. ActiveWaypointLocked remains 0.

```
FLIGHT PLAN

FlightPlanTitle: SCEL to SCSF
       4 :FlightPlanWaypointsNumber        3 :FlightPlanActiveWaypoint     0 :FlightPlanIsActiveWaypointLocked
       1 :FlightPlanRouteType              2 :FlightPlanFlightPlanType     1 :FlightPlanIsActiveWaypoint

---------------------------------------------- FlightPlanWaypoint ----------------------------------------------
     ICAO     111                        Mag                                    Dist      Dist      Rem
Idx  123456789012  Ident   Alt Type  Hdg       Lat        Lon      Dist       Ttl       Rem      Dist       ETE
0    A      SCEL    SCEL  1554    1    0  -33.40935  -70.78492     0.00       0.0      43.6       0.0      0.00
1    WSCSCELSANOK  SANOK     0    2  352  -33.25284  -70.79138     9.40       9.4      34.2       0.0      2.82
2                          0    5   24  -32.95880  -70.58730    20.41      29.8      13.8       0.0      6.12
3    A      SCSF    SCSF  2160    1  330  -32.74934  -70.70198    13.83      43.6       0.0      21.8      4.13
```

Finally, the aircraft reaches the destination waypoint but does not land (Figure **F**). At this point, under autopilot control, it begins repetitive 360° turns to intercept Waypoint 3 because there is no further waypoint to fly to. As shown in the table below, ActiveWaypoint remains 3, ActiveWaypointLocked remains 0, but now, FlightPlanIsActiveWaypoint has changed from 1 to 0 indicating that there is no longer an active waypoint.

```
FLIGHT PLAN

FlightPlanTitle: SCEL to SCSF
       4 :FlightPlanWaypointsNumber        3 :FlightPlanActiveWaypoint     0 :FlightPlanIsActiveWaypointLocked
       1 :FlightPlanRouteType              2 :FlightPlanFlightPlanType     0 :FlightPlanIsActiveWaypoint

---------------------------------------------- FlightPlanWaypoint ----------------------------------------------
     ICAO     111                        Mag                                    Dist      Dist      Rem
Idx  123456789012  Ident   Alt Type  Hdg       Lat        Lon      Dist       Ttl       Rem      Dist       ETE
0    A      SCEL    SCEL  1554    1    0  -33.40935  -70.78492     0.00       0.0      43.6       0.0      0.00
1    WSCSCELSANOK  SANOK     0    2  352  -33.25284  -70.79138     9.40       9.4      34.2       0.0      2.82
2                          0    5   24  -32.95880  -70.58730    20.41      29.8      13.8       0.0      6.12
3    A      SCSF    SCSF  2160    1  330  -32.74934  -70.70198    13.83      43.6       0.0       0.0      4.13
```

Example 4: Changing the Active Waypoint



ActiveWaypoint can be changed different ways. The examples above demonstrate a Flight Plan from Waypoint 0 to 1 to 2 to 3 (Figure **I**). The aircraft position is Waypoint 0 and the flight path is shown in a red dashed line. The ActiveWaypoint is identified by a magenta flight leg and text color.

In Figure **II**, the Flight Plan has been edited to make Waypoint 2 the ActiveWaypoint:

```
2 (>@c:FlightPlanActiveWaypoint)
```

On autopilot, this will cause the aircraft to intercept the Waypoint 2 leg, by-passing Waypoint 1. Upon reaching Waypoint 2, the aircraft will proceed to the destination point, Waypoint 3, as usual. The Flight Plan is un-altered.

In Figure **III**, the Flight Plan is changed to make Waypoint 2 the DirectTo destination:

```
2 (>@c:FlightPlanWaypointIndex)
(@c:FlightPlanWaypointICAO) (>@c:FlightPlanNewWaypointICAO)
(>@c:FlightPlanDirectToDestination)
```

The aircraft will proceed directly to original Waypoint 2 which is now Waypoint 1. DirectToDestination always creates a two waypoint Flight Plan with the aircraft position as Waypoint 0 and the destination point (termination of the Flight Plan) as Waypoint 1.

Another alternative would be to delete Waypoint 1, as in Figure **IV**:

```
1 (>@c:FlightPlanWaypointIndex)
(@c:FlightPlanWaypointICAO) (>@c:FlightPlanNewWaypointICAO)
(>@c: FlightPlanDeleteWaypoint)
```

When Waypoint 1 is deleted, then previous Waypoint 2 becomes the new Waypoint 1, and so forth.

NEWAPPROACH GROUP: ADDING OR CHANGING AN APPROACH

The NewApproach group is a small group of Set-only variables that can be used to add or change Instrument Approaches and Transitions.

The following information is needed:

- The destination airport. Specifically, the ICAO of the destination airport must be known.
- The desired approach and transition index. Every airport having an Instrument Approach Procedure has an indexed approach list containing at least one approach. The index pointer of the desired approach is the required information. Similarly, the index pointer of the desired approach transition is also required. If omitted, the default index value of zero will be assumed for each.

❑ FlightPlanNewApproachAirport (string) [Set]

FlightPlanNewApproachAirport is the full ICAO of the destination airport for the approach you wish to add or change. The airport can be the same one currently in the Flight Plan or a different one. Any source of the ICAO will do:

- ICAOSearchCurrentICAO
- NameSearchCurrentICAO
- Waypoint or Facility Group ICAOs such as WaypointAirportICAO
- Nearest Group ICAOs such as NearestVorCurrentICAO

An example xml statement:

```
(@c:IcaoSearchCurrentICAO) (>@c:FlightPlanNewApproachAirport)
```

❑ FlightPlanNewApproachApproach (enum) [Set]

FlightPlanNewApproachApproach is the index pointer to the approach you want to add or change. The list of instrument approaches is found in the WaypointAirport Group and can be viewed by looping through WaypointAirportCurrentApproach.

❑ FlightPlanNewApproachTransition (enum) [Set]

FlightPlanNewApproachTransition is the index pointer to the desired transition associated with the desired approach. The list of instrument approach transitions is found in the WaypointAirport Group and can be viewed by looping through WaypointAirportApproachCurrentTransition.

❑ FlightPlanNewApproachMissed (bool) [Set]

FlightPlanNewApproachMissed is used to include or exclude the Missed Approach Procedure in the Approach to be added / edited.

- ❑ 0 The Missed Approach Procedure will be **excluded** from the new Approach Procedure. In this situation, if the aircraft does not land at the NewApproachAirport, then (if most common autopilots are controlling the aircraft) it will proceed to the destination waypoint indicated in the Flight Plan. After crossing the destination waypoint, the aircraft will fly a 360° turn to re-intercept the waypoint and will continue to repeat the 360° turn.
- ❑ 1 The Missed Approach Procedure will be **included** in the new Approach Procedure. FlightPlanNewApproachMissed is a Boolean, so any number other than zero will include the Missed Approach Procedure.

  If omitted, the default value of 1, include Missed Approach Procedure, will be assumed by fs9gps.

❑ FlightPlanNewApproachAddInitialLeg (enum) [Set]

FlightPlanNewApproachAddInitialLeg is used to add an initial approach segment. It facilitates routing the aircraft to the Approach Transition Waypoint. The additional segment starts at either the current aircraft location or the Termination Point of the Flight Plan, and extends to the Transition Waypoint of the Approach. The arguments are:

- ❑ 0 **No initial approach segment** will be added.
- ❑ 1 An initial segment from the **aircraft location to the Transition Waypoint** will be added when the approach is **loaded**.
- ❑ 2 An initial segment from the **Termination Point of the Flight Plan to the Transition Waypoint** will be added when the approach is **loaded**.
- ❑ 3 An initial segment from the **aircraft location to the Transition Waypoint** will be added when the approach is **activated**.
- ❑ 4 An initial segment from the **Termination Point of the Flight Plan to the Transition Waypoint** will be added when the Approach is **activated**.
- ❑ 5+ Same as 0

For all cases, the FlightPlanWaypointApproach list will not contain the new leg as a separate entry, but distances of the first approach leg are adjusted to account for the added initial leg (FlightPlanApproachSegmentLength and Distance, ApproachRemainingTotalDistance).
Note that the Microsoft ESP web page (http://msdn.microsoft.com/en-us/library/cc526954.aspx#FlightPlanData) lists AddInitialLeg units as Unavailable, which might be interpreted to suggest that this variable is inactive. It works fine, however, at least in FS9.

❑ FlightPlanLoadApproach (enum) [Set]

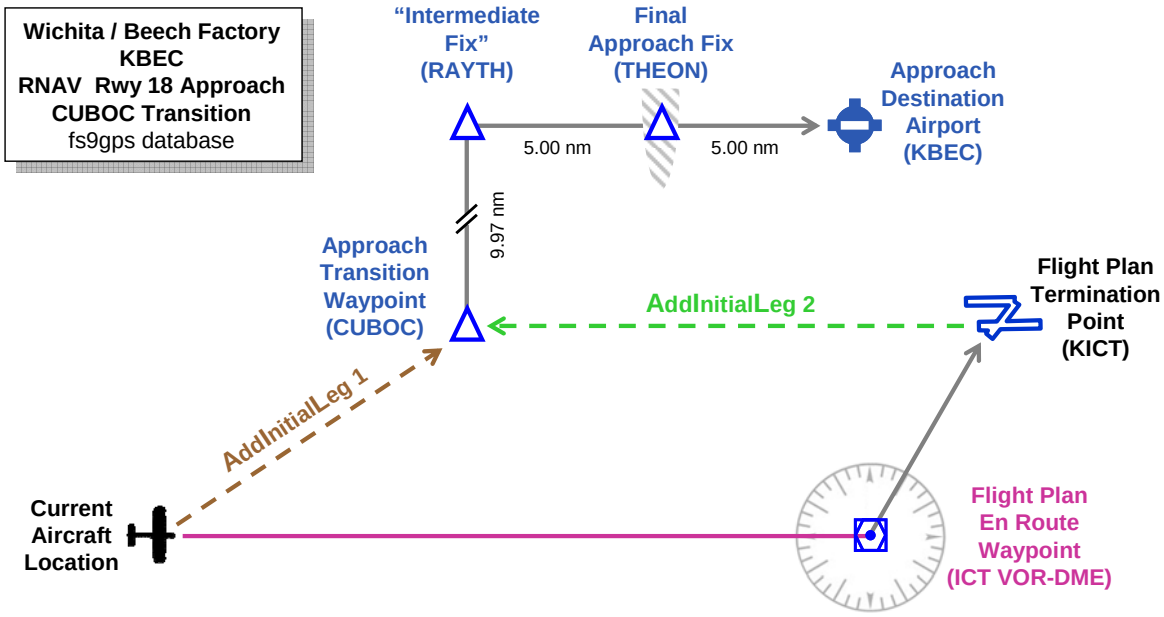
FlightPlanLoadApproach Loads, or Loads and Activates the desired new approach and transition, or Loads and Activates a Vectors-To-Final transition for the current approach depending on the argument used:

- ❑ 0 or negative = **No action**
- ❑ 1 **Load.** The NewApproachAirport, NewApproachApproach, NewApproachTransition, NewApproachMissed and NewApproachAddInitialLeg variables are Loaded but not Activated.
- ❑ 2 **Load and Activate.** The NewApproachAirport, NewApproachApproach, NewApproachTransition, NewApproachMissed and NewApproachAddInitialLeg variables are Loaded and Activated.
- ❑ 3 **Vectors-To-Final.** LoadApproach 3 operates on the existing loaded or activated approach. It will load *and activate* a Vectors-To-Final transition for the currently loaded / activated approach, replacing the existing transition. The new FlightPlanWaypointApproachIndex 0 becomes an extended Final Approach Fix. fs9gps adds 5.00 NMiles to the Final Approach Fix with the same bearing as the Final Approach segment to accommodate the distance that may be required to turn to the Final Approach heading after intercepting the "Vectors-To-Final Fix". You must provide an initial leg or the aircraft will fly to the FAF, not the extended FAF. See the example at the end of this section for further explanation.

  If the intention is to load or activate a *new* approach with a Vectors transition, then the proper choice is to select the new NewApproachAirport and NewApproachApproach, and then NewApproachTransition = 0 (0 is always the Vectors transition Index), followed by NewApproachMissed = 0 or 1, NewApproachAddInitialLeg = 1 or 2, and finally, LoadApproach = 1, 2, or 3.
- ❑ 4+ **Same as 2**

Example 5: Adding or Changing an Approach

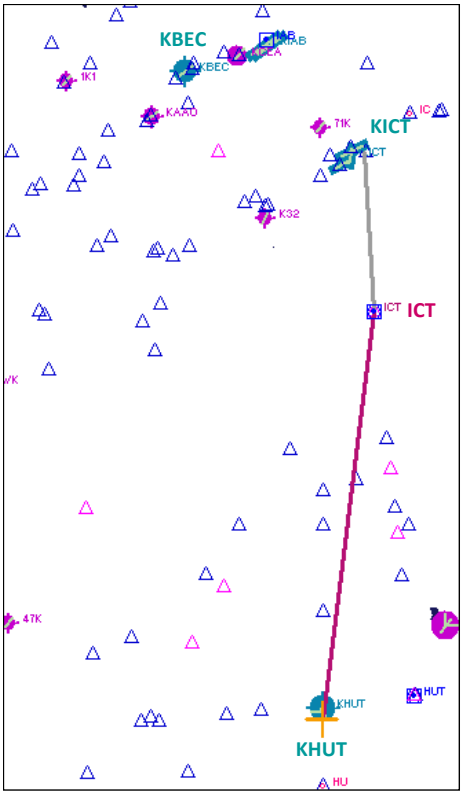
Example 5 demonstrates use of the NewApproach Group variables. It uses the fs9gps RNAV Rwy 18 Approach into the Beech Aircraft Factory Airport (KBEC), Wichita, Kansas, USA. The stock fs9gps database seems not to contain current RNAV Waypoints; instead, the approach is defined using fs9gps terminal waypoints (△). The waypoint names and positions and approach nomenclature used in Example 5 are shown below.



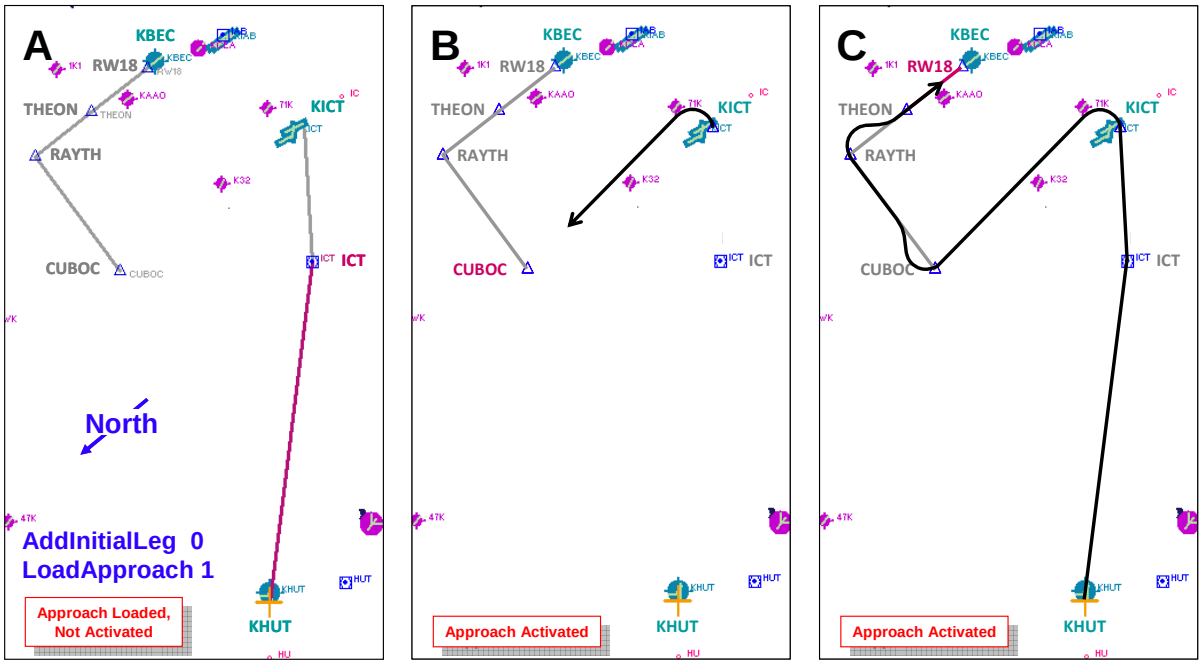
The example begins with a Flight Plan from KHUT to KICT via the ICT VOR-DME en route waypoint.

It is subsequently edited to add the KBEC RNAV18 Approach. In the examples that follow, the Missed Approach Procedure is excluded from the Approach, and, in Examples 5.1 through 5.4, the Approach is loaded, but not activated.

The aircraft does not land at KICT. Instead, it executes the approach into KBEC after passing over KICT, the termination point of the Flight Plan. fs9gps automatically activates a loaded, but not activated, Approach when the aircraft reaches the termination point of the Flight Plan but does not land there.



Example 5.1 NewApproachAddInitialLeg = 0, LoadApproach = 1



In Figure **A**, above, the new approach, KBEC RNAV18, has been added with NewApproachAddInitialLeg set to 0. The xml (the order is important):

```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
0 (>@c:FlightPlanNewApproachAddInitialLeg)
1 (>@c:FlightPlanLoadApproach)
```

```
FLIGHT PLAN

FlightPlanTitle: KHUT to KICT
       3 :FlightPlanWaypointsNumber        1 :FlightPlanActiveWaypoint     0 :FlightPlanIsActiveWaypointLocked
       1 :FlightPlanRouteType              2 :FlightPlanFlightPlanType     1 :FlightPlanIsActiveWaypoint

------------------------------------------- FlightPlanWaypoint -------------------------------------------
     ICAO     111                      Mag                                  Dist      Dist      Rem
Idx  123456789012  Ident   Alt Type  Hdg       Lat         Lon      Dist      Ttl       Rem      Dist      ETE
0    A      KHUT    KHUT  1541    1    0   38.07383   -97.87067      0.00      0.0      33.2       0.0     0.00
1    VK3    ICT      ICT  1470    3  138   37.74517   -97.58383     23.95     23.9       9.3      23.9     6.82
2    A      KICT    KICT  1332    1  128   37.63550   -97.44583      9.29     33.2       0.0       9.3     2.60

FLIGHT PLAN NEW APPROACH: KBEC

       4 :ApprWaypointsNumber              6 :FlightPlanApprType           3 :FlightPlanWaypointApproachIndex
 RNAV 18 :FlightPlanApprName             CUBOC :FlightPlanApprTransName    0 :FlightPlanActiveApproachWaypoint
       0 :FlightPlanIsActiveApproach         2 :FlightPlanApproachIndex    5 :FlightPlanApproachTransitionIndex

------------------------------------- FlightPlanWaypointApproach -------------------------------------
     ICAO     111                      Latitude     Longitude    Latitude    Longitude                Leg  Course
Idx  123456789012   Name Type Mode    (Deg Min)     (Deg Min)   (Degrees)   (Degrees)   Alt  Trgt    Dist   (mag)
0    WK3KBECCUBOC  CUBOC    1    1  37 54.1838   -97 22.8128   37.903064  -97.380214  4000     0   0.000   -1.00
1    WK3KBECRAYTH  RAYTH    1    1  37 52.1593   -97 10.4420   37.869322  -97.174033  3000     0   9.972  101.78
2    WK3KBECTHEON  THEON    1    1  37 47.2380   -97 11.5962   37.787300  -97.193269  3000     0   5.006  190.49
3    RK3KBECRW18    RW18    1    2  37 42.3165   -97 12.7477   37.705275  -97.212461  1453     0   5.006  184.00
```

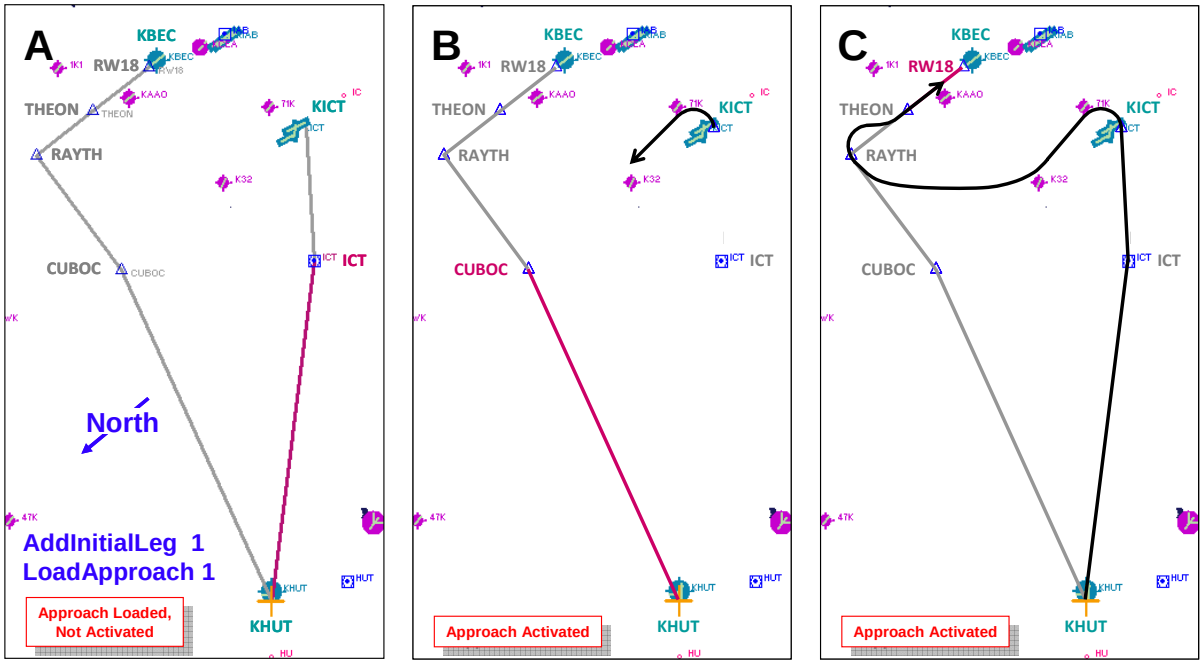
The Flight Plan and Approach segments are listed above.

In the absence of an approach segment between the aircraft and the Transition Waypoint, the aircraft flies directly to the Transition Waypoint.

Figure **B** shows the first approach segment after the approach is (automatically) activated. No Initial Leg has been added, and the aircraft proceeds directly to the Transition Waypoint, CUBOC.

Figure **C** shows the complete flight path.

Example 5.2 NewApproachAddInitialLeg = 1, LoadApproach = 1



The xml:

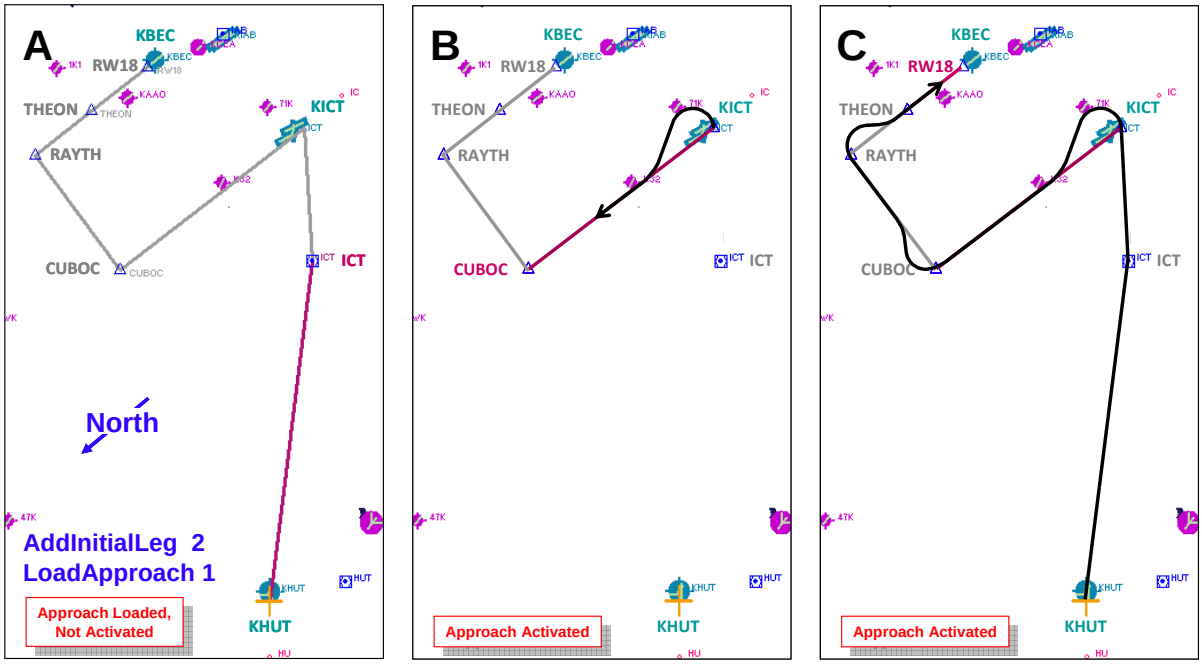
```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
1 (>@c:FlightPlanNewApproachAddInitialLeg)
1 (>@c:FlightPlanLoadApproach)
```

Figure **A** shows that an initial approach leg from the aircraft position to the Transition Waypoint has been added. Because the approach has not been activated, it is an inactive approach segment (gray color) and the aircraft will fly towards ICT VOR-DME according to the Flight Plan, which is active.

Figure **B** shows the first approach segment after the approach is (automatically) activated. Now, the added leg (original aircraft location to Transition Waypoint) is active and the aircraft flies to intercept that segment. It is not flying directly to CUBOC, the Transition Waypoint, rather, it is flying to intercept the active approach segment. The aircraft is actually a little ahead of the waypoint, and because of the intercept algorithm, it never reaches CUBOC before the next approach segment becomes active and the aircraft turns to intercept that segment. This is admittedly an unrealistic scenario in that InitialLeg = 1 was selected but the aircraft was allowed to continue flying the Flight Plan rather than the Approach.

Figure **C** shows the complete flight path.

Example 5.3 NewApproachAddInitialLeg = 2, LoadApproach = 1



The xml:

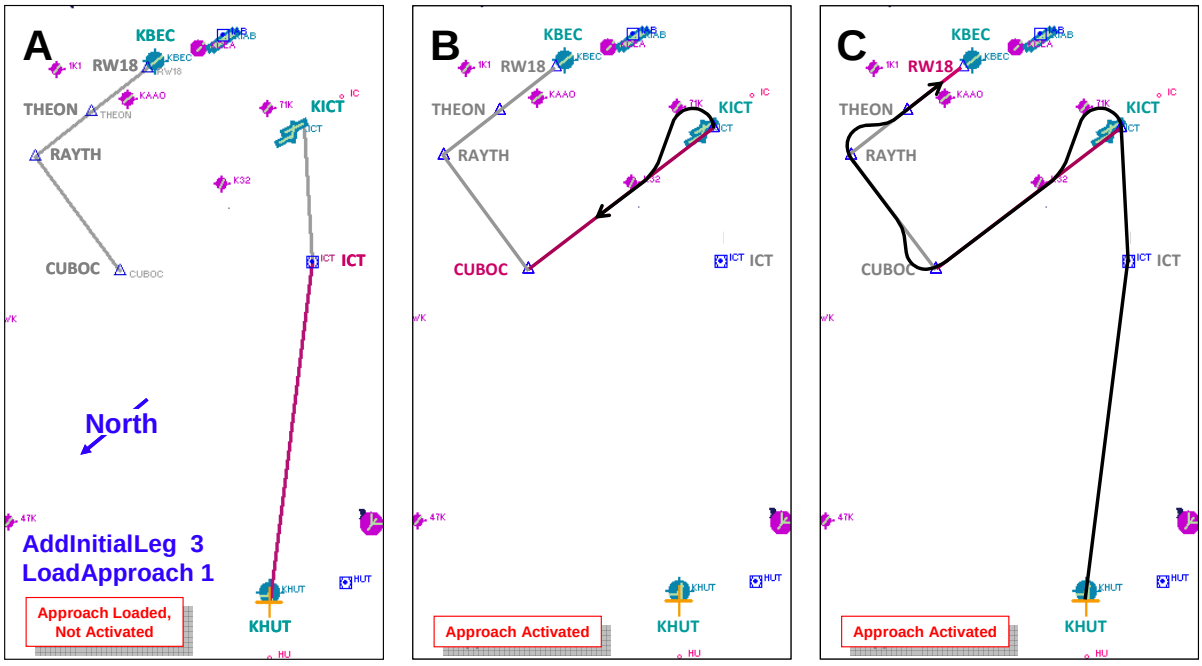
```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
2 (>@c:FlightPlanNewApproachAddInitialLeg)
1 (>@c:FlightPlanLoadApproach)
```

Figure **A** shows that an initial approach leg from the Termination Point of the Flight Plan to the Transition Waypoint has been added.

Figure **B** shows the first approach segment after the approach is (automatically) activated. The added intitial leg is now active, and the aircraft turns to intercept that segment.

Figure **C** shows the complete flight path.

Example 5.4 NewApproachAddInitialLeg = 3, LoadApproach = 1



The xml:

```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
3 (>@c:FlightPlanNewApproachAddInitialLeg)
1 (>@c:FlightPlanLoadApproach)
```

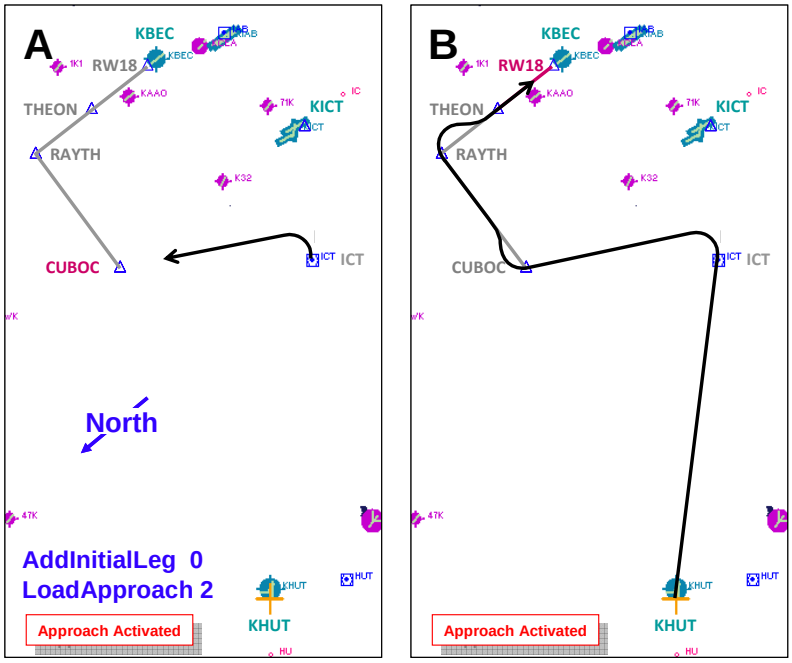
Figure **A** shows approach segments beginning at CUBOC and ending at the destination runway waypoint. No initial legs are shown.

Figure **B** shows the first approach segment after the approach is (automatically) activated. An initial approach leg from the Termination Point of the Flight Plan to the Transition Waypoint was automatically added when the approach was activated, and the aircraft turns to intercept that segment.

Figure **C** shows the complete flight path.

The next series demonstrates loading **and activating** the KBEC RNAV18 Approach in flight. The aircraft begins under control of the Flight Plan until it reaches ICT VOR-DME, at which point, the Approach is **loaded and activated**.

Example 5.5 NewApproachAddInitialLeg = 0, LoadApproach = 2



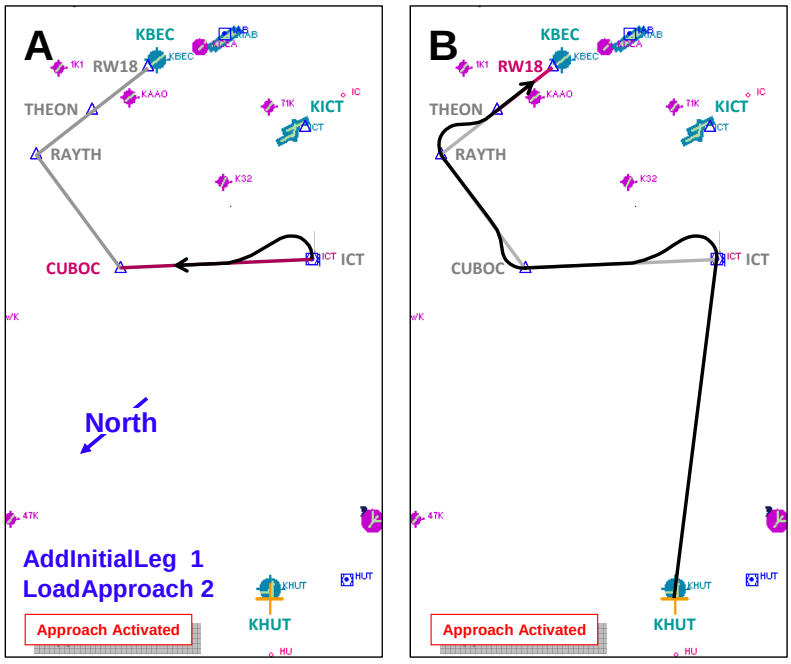
The xml:

```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
0 (>@c:FlightPlanNewApproachAddInitialLeg)
2 (>@c:FlightPlanLoadApproach)
```

Figure **A** shows the first approach segment after the approach is activated. There is no leg connecting the aircraft and the Transition Waypoint. In the absence of an approach segment between the aircraft and the Transition Waypoint, the aircraft flies directly to the waypoint.

Figure **B** shows the complete flight path.

Example 5.6 NewApproachAddInitialLeg = 1, LoadApproach = 2



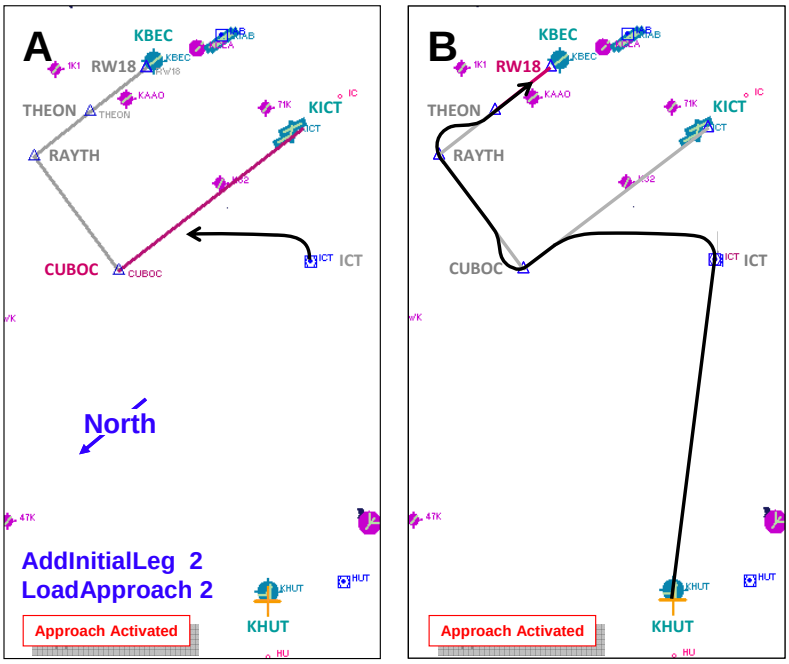
The xml:

```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
1 (>@c:FlightPlanNewApproachAddInitialLeg)
2 (>@c:FlightPlanLoadApproach)
```

Figure **A** shows the first approach segment after the approach is activated. AddInitialLeg = 1 resulted in a new segment added between the aircraft location and the Transition Waypoint. The aircraft turns to intercept the new segment, it does not fly directly to the Transition Waypoint.

Figure **B** shows the complete flight path.

Example 5.7 NewApproachAddInitialLeg = 2, LoadApproach = 2



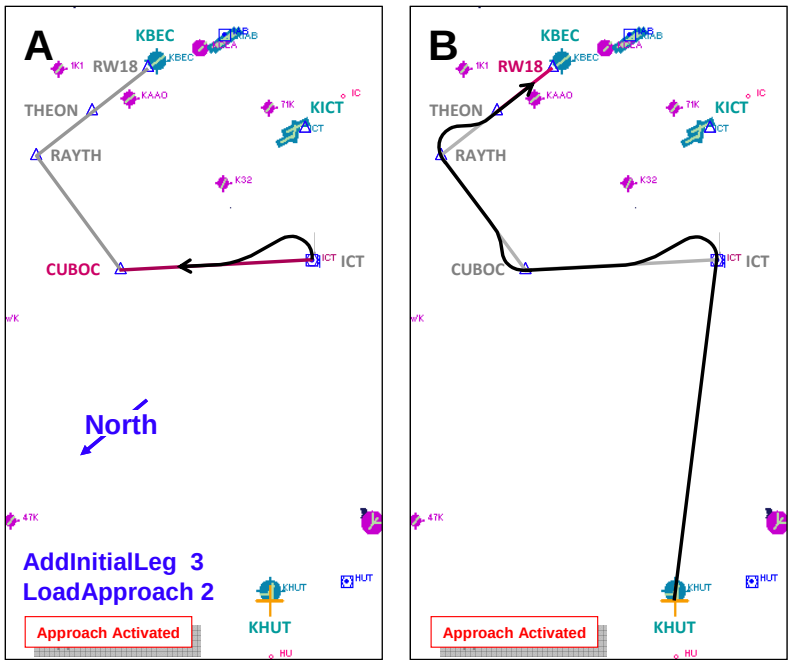
The xml:

```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
2 (>@c:FlightPlanNewApproachAddInitialLeg)
2 (>@c:FlightPlanLoadApproach)
```

Figure **A** shows the first approach segment after the approach is activated. AddInitialLeg = 2 resulted in a new segment added between the Termination Point of the Flight Plan and the Transition Waypoint. The aircraft turns to intercept the new segment; it does not fly directly to the Transition Waypoint.

Figure **B** shows the complete flight path.

Example 5.8 NewApproachAddInitialLeg = 3, LoadApproach = 2



The xml:

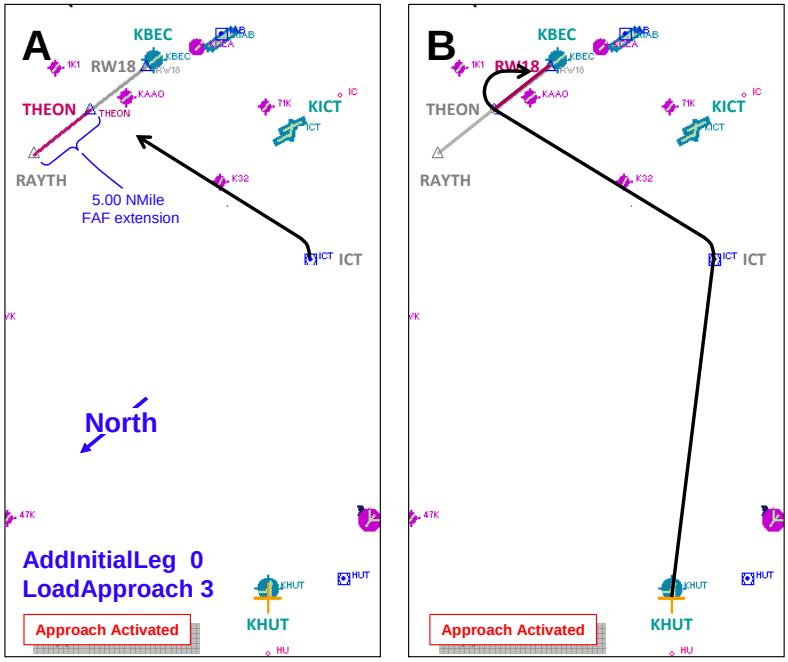
```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
3 (>@c:FlightPlanNewApproachAddInitialLeg)
2 (>@c:FlightPlanLoadApproach)
```

Figure **A** shows the first approach segment after the approach is activated. This case is the same as AddInitialLeg = 1 because the Approach was activated at the same time it was loaded.

Figure **B** shows the complete flight path.

Finally, the last example demonstrates FlightPlanLoadApproach = 3, the Activate Vectors-To-Final instruction.

Example 5.9 NewApproachAddInitialLeg = 0, LoadApproach = 3



The xml:

```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
1 (>@c:FlightPlanLoadApproach)
```

Vectors-To-Final replaces the existing transition with a Vectors transition. It requires that at least an approach (but not necessarily a transition, too) first be loaded or activated, or nothing will happen. If NewApproachMissed and NewApproachAddInitialLeg values have previously been entered, they will be used again. If not, the default values of zero will be used for AddInitialLeg, and 1 for Missed. In Example 5.9, the KBEC RNAV18 Approach, CUBOC Transition is already loaded when Vectors-To-Final (LoadApproach = 3) is executed.

The xml (placed following the `1 (>@c:FlightPlanLoadApproach)` statement above:

```
0 (>@c:FlightPlanNewApproachAddInitialLeg)
3 (>@c:FlightPlanLoadApproach)
```

The Flight Plan and Approach segments are listed below:

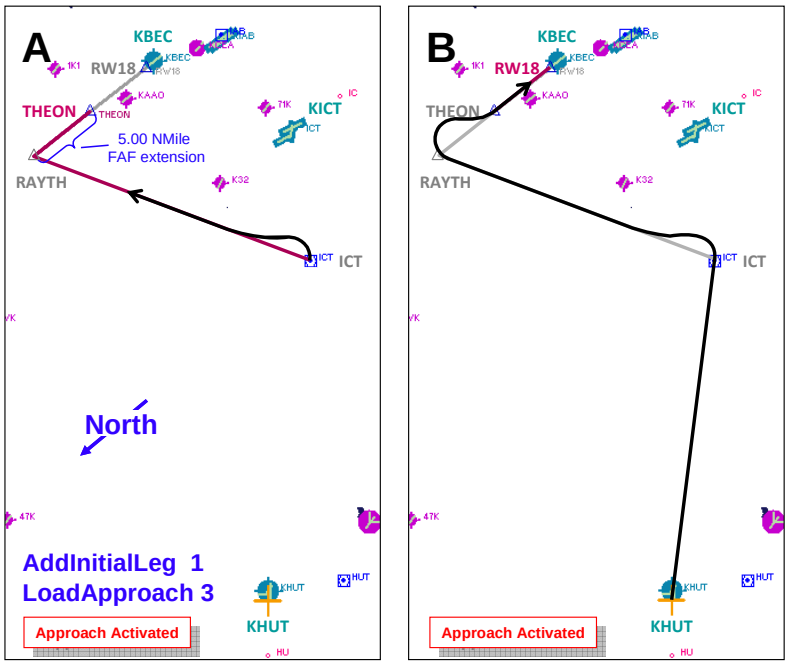
```
FLIGHT PLAN

FlightPlanTitle: KHUT to KICT
       3 :FlightPlanWaypointsNumber        2 :FlightPlanActiveWaypoint     0 :FlightPlanIsActiveWaypointLocked
       1 :FlightPlanRouteType              2 :FlightPlanFlightPlanType     1 :FlightPlanIsActiveWaypoint

------------------------------------------- FlightPlanWaypoint -------------------------------------------
     ICAO     111                    Mag                                    Dist     Dist     Rem
Idx  123456789012  Ident   Alt Type  Hdg       Lat        Lon      Dist      Ttl      Rem     Dist      ETE
0    A       KHUT    KHUT  1541    1    0   38.07383  -97.87067     0.00      0.0     33.2      0.0     0.00
1    VK3     ICT      ICT  1470    3  138   37.74517  -97.58383    23.95     23.9      9.3      0.0     6.82
2    A       KICT    KICT  1332    1  128   37.63550  -97.44583     9.29     33.2      0.0      9.4     2.60

FLIGHT PLAN NEW APPROACH: KBEC

       2 :ApprWaypointsNumber              6 :FlightPlanApprType           1 :FlightPlanWaypointApproachIndex
 RNAV 18 :FlightPlanApprName          VECTORS :FlightPlanApprTransName      0 :FlightPlanActiveApproachWaypoint
       1 :FlightPlanIsActiveApproach       2 :FlightPlanApproachIndex      0 :FlightPlanApproachTransitionIndex

----------------------------------------- FlightPlanWaypointApproach -----------------------------------------
     ICAO     111                    Latitude     Longitude   Latitude   Longitude                Leg  Course
Idx  123456789012   Name Type Mode   (Deg Min)     (Deg Min)  (Degrees)  (Degrees)   Alt  Trgt   Dist   (mag)
0    WK3KBECTHEON  THEON   11    2  37 47.2380   -97 11.5962  37.787300 -97.193269  3000     0  5.001  181.53
1    RK3KBECRW18    RW18    1    2  37 42.3165   -97 12.7477  37.705275 -97.212461  1453     0  5.006  184.00
```

Figure **A** shows the first approach segment after the Vectors-To-Final approach is loaded/activated. No initial leg is added to the approach, so the aircraft proceeds directly to the Vectors-To-Final Waypoint, the Final Approach Fix (THEON).

Note that the aircraft does <u>not</u> proceed to new Transition Waypoint which is the outboard end of the extended approach (near RAYTH). Flying direct to the Final Approach Fix results in an unacceptable, un-stabilized approach; an aircraft should never make a significant turn after reaching the Final Approach Fix. For this reason, an initial leg should be added when selecting FlightPlanLoadApproach = 3 Vectors-To-Final, as demonstrated in the next example.

Figure **B** shows the complete flight path.

Example 5.10 NewApproachAddInitialLeg = 1, LoadApproach = 3



The xml:

```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
1 (>@c:FlightPlanLoadApproach)
1 (>@c:FlightPlanNewApproachAddInitialLeg)
3 (>@c:FlightPlanLoadApproach)
```

Figure **A** shows the first approach segment after the Vectors-To-Final approach is loaded/activated. In this case, an initial leg from the aircraft location to the new Transition Waypoint near RAYTH has been added. The aircraft turns to intercept that approach segment.

Figure **B** shows the complete flight path.

Note: Throughout Example 5, the flight paths depict an exaggerated turn radius in order to more clearly demonstrate the action of the different FlightPlanNewApproach selections.

En Route Navigation

- **FlightPlanWaypointIndex (enum) [Get, Set]**

The currently indexed waypoint.

- **FlightPlanWaypointLatitude**
- **FlightPlanWaypointLongitude (degrees, radians) [Get]**

Latitude and longitude of the currently indexed Waypoint. Units are degrees (decimal format, not deg, min, sec) or radians.

- **FlightPlanWaypointAltitude (feet) [Get]**

Ground elevation, or altitude, (asl) of the currently indexed waypoint. Only Waypoint types 1, 3, and 4 (Airport, VOR, NDB) have altitudes. Intersection Waypoints do not.

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

The ICAO of the currently indexed Waypoint.

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

The Ident of the currently indexed Waypoint.

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

The Low Altitude (Victor) or High Altitude (Jet) Airway Ident if the currently indexed flight plan leg is part of an Airway and FlightPlanRouteType = 2 or 3 (Low Altitude or High Altitude Airways).

- **FlightPlanWaypointType (enum) [Get]**

Flight Plan Waypoint Type

Bit	Name and Type #	Bit	Name and Type #	Bit	Name and Type #
0	NONE = 0	3	VOR = 3	5	USER = 5
1	AIRPORT = 1	4	NDB = 4	6	ATC = 6
2	INTERSECTION = 2				

http://msdn.microsoft.com/en-us/library/cc526954.aspx#ATC_WAYPOINT_TYPE

Waypoints added by the user through FlightPlanAddWaypoint that do not correspond to Waypoint Types 1 through 4, that is, the added Waypoint is simply a point on the map, then Waypoint Type = 5 is assigned by fs9gps.

❑ FlightPlanWaypointMinAltitude (feet) [Get]

FlightPlanWaypointMinAltitude is the Minimum En route Altitude assigned to Low Altitude Victor and High Altitude Jet Airways in the fs9gps database. Only flight plan legs that are part of a Victor or Jet Airway have a WaypointMinAltitude. Additionally, WaypointMinAltitude is returned only when FlightPlanRouteType = 2 or 3.

MEAs typically vary along an Airway, so the MEA belonging to the portion of the Airway which is the currently indexed flight plan leg is returned as WaypointMinAltitude. For example, see the various WaypointMinAltitude associated with V134 and V591 in the figure below. Non-Airway legs such as Airport KLIC to VOR FQF in the figure below will return a zero value for WaypointMinAltitude.

FLIGHT PLAN WAYPOINT

KLIC to KDTA :FlightPlanTitle
18 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
2 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 18000 :FlightPlanCruisingAltitude

FlightPlanWaypoint

Idx	ICAO 111	Ident	Airwy Ident	Alt	Min Alt	Type	Mag Hdg	Spd Est	Dist	Dist Ttl	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A KLIC	KLIC		5365	0	1	0	200	0.00	0.0	430.8	0.0	0.00	0.00	0.00	0.00	242.0	0.0	0.0
1	VK2 FQF	FQF		0	0	3	290	200	51.04	51.0	379.7	51.0	15.30	0.00	0.00	0.00	0.0	11.2	0.0
2	WK2 BREWS	BREWS	V134	0	16500	2	254	200	25.95	77.0	353.8	25.9	7.77	0.00	0.00	0.00	0.0	5.7	0.0
3	WK2 FUNDS	FUNDS	V134	0	16500	2	254	200	41.33	118.3	312.5	41.3	12.38	0.00	0.00	0.00	0.0	9.0	0.0
4	WK2 DAVVY	DAVVY	V134	0	1600	2	245	200	13.75	132.1	298.7	13.7	4.12	0.00	0.00	0.00	0.0	3.0	0.0
5	WK2 LAWSN	LAWSN	V134	0	16000	2	245	200	5.50	137.6	293.2	5.5	1.65	0.00	0.00	0.00	0.0	1.2	0.0
6	WK2 HERLS	HERLS	V134	0	16000	2	245	200	11.97	149.5	281.2	12.0	3.58	0.00	0.00	0.00	0.0	2.6	0.0
7	VK2 DBL	DBL	V134	0	14000	3	244	200	7.98	157.5	273.3	8.0	2.38	0.00	0.00	0.00	0.0	1.7	0.0
8	WK2 LINDZ	LINDZ	V134	0	14000	2	244	200	12.58	170.1	260.7	12.6	3.77	0.00	0.00	0.00	0.0	2.7	0.0
9	WK2 GLENO	GLENO	V591	0	14000	2	244	200	10.09	180.2	250.6	10.1	3.02	0.00	0.00	0.00	0.0	2.2	0.0
10	WK2 SLOLM	SLOLM	V591	0	14000	2	243	200	12.40	192.6	238.2	12.4	3.72	0.00	0.00	0.00	0.0	2.7	0.0
11	WK2 EDUKY	EDUKY	V591	0	13000	2	243	200	23.03	215.6	215.2	23.0	6.90	0.00	0.00	0.00	0.0	5.0	0.0
12	WK2 PACES	PACES	V591	0	13000	2	243	200	8.23	223.8	206.9	8.2	2.47	0.00	0.00	0.00	0.0	1.8	0.0
13	VK2 JNC	JNC	V591	0	9000	3	243	200	24.76	248.6	182.2	24.8	7.42	0.00	0.00	0.00	0.0	5.4	0.0
14	WK2 EGEZE	EGEZE	V134	0	11900	2	278	200	81.72	330.3	100.4	81.7	24.50	0.00	0.00	0.00	0.0	17.9	0.0
15	VK2 PUC	PUC	V134	0	11900	3	276	200	14.96	345.3	85.5	15.0	4.48	0.00	0.00	0.00	0.0	3.3	0.0
16	WK2 ARBIH	ARBIH	V134	0	13000	2	293	200	9.98	355.3	75.5	10.0	2.98	0.00	0.00	0.00	0.0	2.2	0.0
17	A KDTA	KDTA		4755	0	1	242	200	75.50	430.8	0.0	75.5	22.63	0.00	0.00	0.00	0.0	16.5	0.0

FLIGHT PLAN WAYPOINT

KLIC to KDTA :FlightPlanTitle
6 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
3 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 34000 :FlightPlanCruisingAltitude

FlightPlanWaypoint

Idx	ICAO 111	Ident	Airwy Ident	Alt	Min Alt	Type	Mag Hdg	Spd Est	Dist	Dist Ttl	Rem Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A KLIC	KLIC		5365	0	1	0	200	0.00	0.0	420.4	0.0	0.00	0.00	0.00	0.00	242.0	0.0	0.0
1	VK2 FQF	FQF		0	0	3	290	200	51.04	51.0	369.4	51.0	15.30	0.00	0.00	0.00	0.0	11.2	0.0
2	VK2 EKR	EKR	J116	0	20000	3	268	200	153.81	204.8	215.5	153.8	46.13	0.00	0.00	0.00	0.0	33.7	0.0
3	WK2 HELPR	HELPR	J199	0	33000	2	249	200	122.20	327.0	93.3	122.2	36.65	0.00	0.00	0.00	0.0	26.8	0.0
4	WK2 OFFIG	OFFIG	J199	0	33000	2	247	200	41.31	368.4	52.0	41.3	12.38	0.00	0.00	0.00	0.0	9.0	0.0
5	A KDTA	KDTA		4755	0	1	239	200	52.03	420.4	0.0	52.0	15.60	0.00	0.00	0.00	0.0	11.4	0.0

There is one peculiarity to be aware of. If you want WaypointMinAltitude in **feet**, you must either omit units or specify units as 'meters'. For example, V10 airway between Hutchinson VOR intersection (Kansas, USA) and STAFF intersection (Kansas, USA) has a MEA of 3700 feet:

```
(C:fs9gps:FlightPlanWaypointMinAltitude) = 3700
```

or,

```
(C:fs9gps:FlightPlanWaypointMinAltitude, meters) = 3700
```

But, if you specify units = feet:

```
(C:fs9gps:FlightPlanWaypointMinAltitude, feet) = 12140
```

the database altitude apparently will be internally interpreted as *meters*, then multiplied by 3.281, resulting in 12,140 -- which is not the correct value. If you want meters units, then you must do the conversion manually:

```
(C:fs9gps:FlightPlanWaypointMinAltitude) 3.281 /
(>L:FlightPlanWaypointMinAltitude, meters)
```

As an additional note, not all segments of all airways within the fs9gps database have assigned MEAs. Isolated segments of many airways simply have a zero, or another obviously incorrect value while the adjoining segments may have the proper MEA.

❑ FlightPlanWaypointFrequency (MHz) [Get]

FlightPlanWaypointFrequency is not implemented in fs9gps according to an old blog by MSFT's Susan Ashcroft (when ACE's was still around). However, WaypointFrequency does return a value. It is 100.00 for all waypoint types except VORs. For VORs, it is a value that is about twice the actual VOR frequency, but it does not make sense, nor is it predictable.

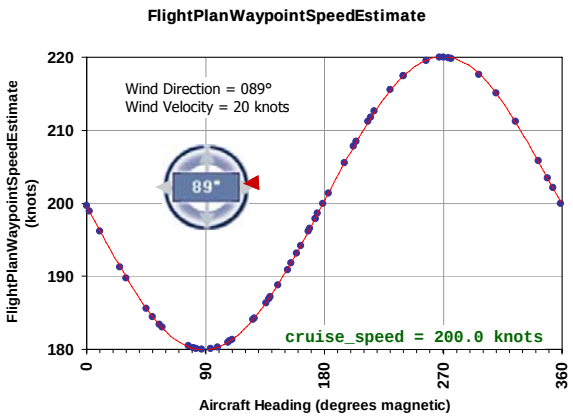
❑ FlightPlanWaypointMagneticHeading (degrees) [Get]

FlightPlanWaypointMagneticHeading is the magnetic bearing to the currently indexed Waypoint from the previous Waypoint.

WaypointMagneticHeading is not the same as FlightPlanWaypointApproachCourse. Both return magnetic bearing but WaypointMagneticHeading applies to en route Waypoints and WaypointApproachCourse applies to approach segments.

❑ FlightPlanWaypointSpeedEstimate (knots) [Get]

FlightPlanWaypointSpeedEstimate is a ground speed estimate that fs9gps derives from the aircraft's cruising airspeed defined in the aircraft.cfg file. It is used to calculate WaypointETE. WaypointETE is not updated using actual groundspeed as the flight progresses. As demonstrated in the figure, wind speed and direction affect WaypointSpeedEstimate. Note if the wind changes in-flight, SpeedEstimate and WaypointETE will not change.



❑ FlightPlanWaypointDistance (nmiles) [Get]

FlightPlanWaypointDistance is the length of the currently indexed Flight Plan leg.

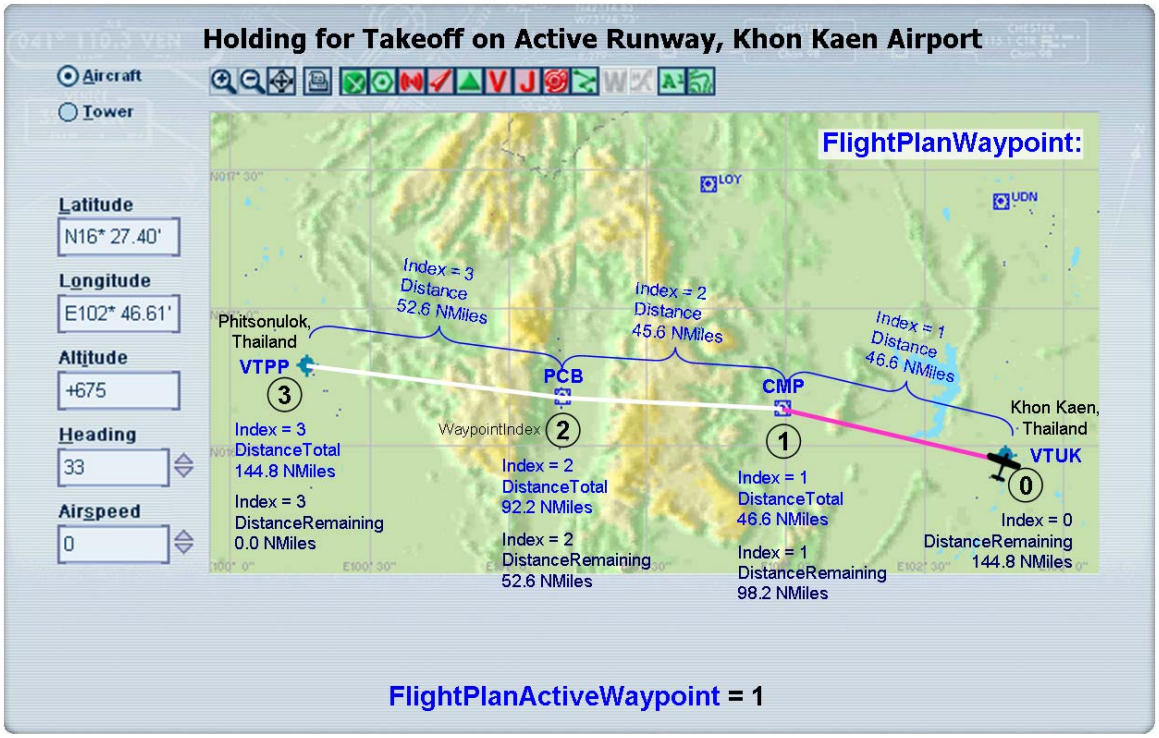
❑ FlightPlanWaypointDistanceTotal (nmiles) [Get]

FlightPlanWaypointDistanceTotal is the cumulative distance of all Flight Plan legs starting at Waypoint index zero (the departure airport) through the currently indexed Waypoint. When the index points to the last Waypoint (the destination airport), DistanceTotal is the total length of the flight plan measured along flight legs.

❑ FlightPlanWaypointDistanceRemaining (nmiles) [Get]

FlightPlanWaypointDistanceRemaining is the distance from the currently indexed Waypoint to the last Waypoint, the destination airport.

FlightPlanWaypointDistance, DistanceTotal, DistanceRemaining



FLIGHT PLAN WAYPOINT

VTUK to VTPP :FlightPlanTitle
4 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 8000 :FlightPlanCruisingAltitude

FlightPlanWaypoint

Idx	ICAO 111 123456789012	Ident	Alt	Type	Mag Hdg	Spd Est	Dist	Dist Ttl	Dist Rem	Rem Dist	Rem Ttl Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A VTUK	VTUK	670	1	0	200	0.00	0.0	144.8	0.0	0.0	0.00	0.00	0.00	0.00	242.0	0.0	0.0
1	VVT CMP	CMP	650	3	284	200	46.60	46.6	98.2	46.6	46.6	13.97	0.00	0.00	0.00	0.0	10.2	0.0
2	VVT PCB	PCB	449	3	274	200	45.58	92.2	52.6	45.6	92.2	13.67	0.00	0.00	0.00	0.0	10.0	0.0
3	A VTPP	VTPP	145	1	277	200	52.59	144.8	0.0	52.6	144.8	15.77	0.00	0.00	0.00	0.0	11.5	0.0

❑ FlightPlanWaypointRemainingDistance (nmiles) [Get]

FlightPlanWaypointRemainingDistance is the leg distance remaining to be flown. For the active waypoint, that is, for the segment currently being flown, it is the remaining distance from the aircraft's current position to the next Waypoint. For Waypoints beyond that, it is just the total length of that leg. For Waypoints already passed, WaypointRemainingDistance is 0.0.

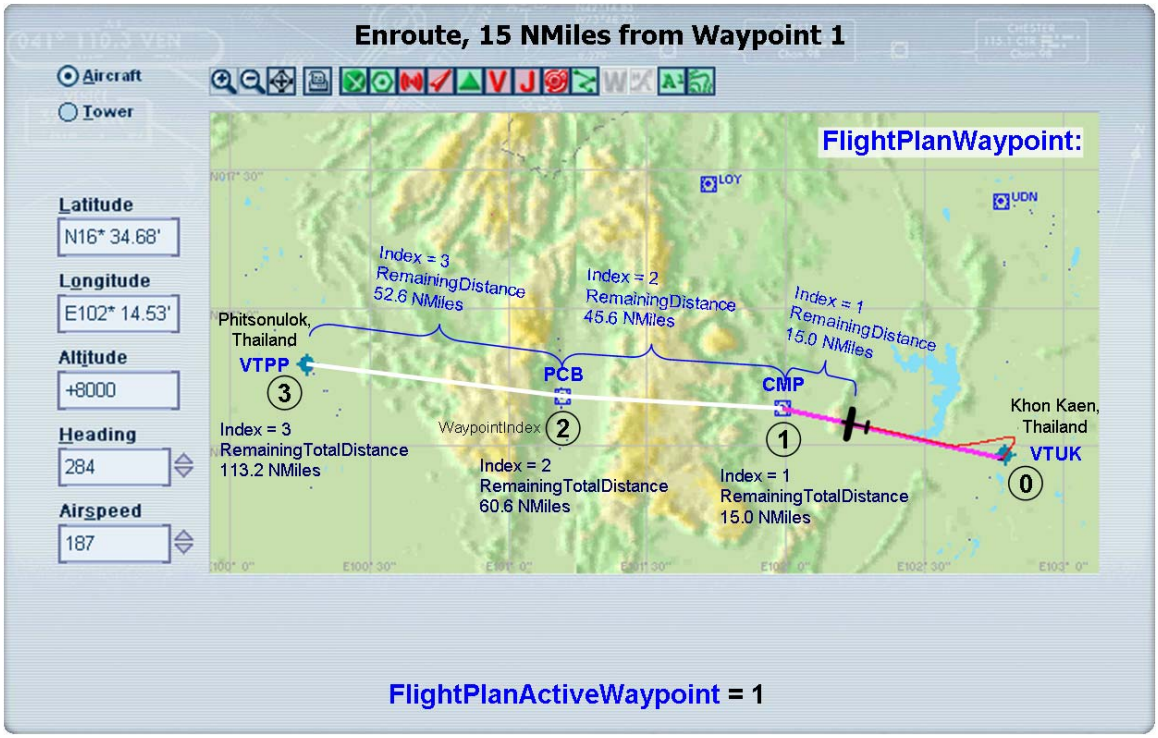
In the figure below, the aircraft is en route, and 15.0 NMiles remain before reaching Waypoint 1, CMP VOR. FlightPlanActiveWaypoint = 1. WaypointRemainingDistance for Index 1 is therefore, 15.0. Because the aircraft is not yet on leg 2 (ActiveWaypoint = 2) or leg 3 (ActiveWaypoint = 3), WaypointRemainingDistance for those Waypoints is still the total length of the respective Flight Plan leg.

WaypointRemainingDistance for the ActiveWaypoint counts down as the flight progresses.

❑ FlightPlanWaypointRemainingTotalDistance (nmiles) [Get]

FlightPlanWaypointRemainingTotalDistance is the cumulative remaining distance from current aircraft position to the indexed waypoint. It is the same as RemainingDistance when the currently indexed waypoint is the Active Waypoint. When the indexed waypoint is the destination airport, RemainingDistance represents the total distance remaining in the flight. RemainingDistance is measured along the flight path; it is not a direct-to measurement.

FlightPlanWaypointRemainingDistance, RemainingTotalDistance



FLIGHT PLAN WAYPOINT

VTUK to VTPP :FlightPlanTitle
4 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 8000 :FlightPlanCruisingAltitude

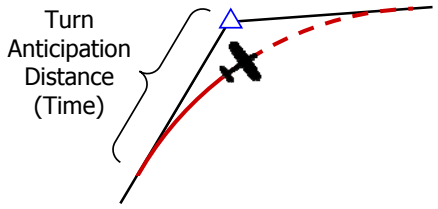
FlightPlanWaypoint

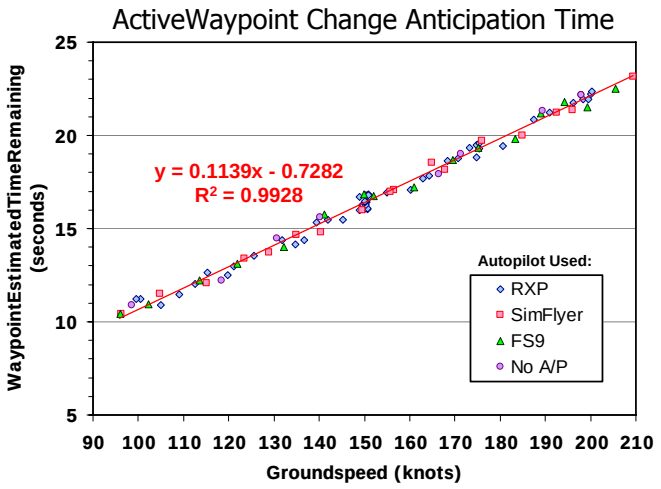
Idx	ICAO 111 1234567890112	Ident	Alt	Type	Mag Hdg	Spd Est	Dist	Dist Ttl	Dist Rem	Rem Dist	Rem Ttl Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A	VTUK	VTUK	670	1	0	200	0.00	0.0	144.8	0.0	0.0	0.00	0.00	0.00	10.72	241.6	0.0	0.0
1	VVT	CMP	CMP	650	3	284	200	46.60	46.6	98.2	15.0	15.0	13.97	0.00	4.28	10.99	0.0	10.2	0.0
2	VVT	PCB	PCB	449	3	274	200	45.58	92.2	52.6	45.6	60.6	13.67	0.00	13.00	11.21	0.0	10.0	0.0
3	A	VTPP	VTPP	145	1	277	200	52.59	144.8	0.0	52.6	113.2	15.77	0.00	15.00	11.46	0.0	11.5	0.0

TURN ANTICIPATION

To accomplish a smooth turn to the next heading as the aircraft approaches a waypoint, the aircraft must begin its turn before actually reaching the waypoint. The distance at which this happens is the Turn Anticipation Distance. Turn Anticipation Distance algorithms and rules of thumb in the literature vary but most are a function of the amount of turn; how many degrees change of direction. With the fs9gps module, however, Turn Anticipation is a function of groundspeed, and the ActiveWaypoint changes, advancing by 1, when a certain seconds-to-waypoint (WaypointEstimatedTimeRemaining) time is reached. This is independent of the amount of degrees turned. Several (maybe most) popular flight sim autopilots, including the stock FS9 Bendix-King, Reality-XP STEC55X, and the Simflyer STEC55X autopilots which I have tested, initiate the turn when the ActiveWaypoint changes, or, when on approach, when FlightPlanActiveApproachWaypoint changes.

I have timed hundreds of ActiveWaypoint changes at random groundspeeds and direction changes using a variety of autopilots and also without autopilot. The conclusions are that the timing of the ActiveWaypoint change is independent of the use of an autopilot (of course) and the amount of turn, and that TimeRemaining when the ActiveWaypoint change occurs is a linear function of groundspeed (which means it's a power function of distance), as shown in the graph below. The Groundspeed vs. WaypointEstimatedTimeRemaining relationship is built into fs9gps.





The table below captures FlightPlanWaypoint variable status the moment *before* ActiveWaypoint changes from 1 to 2. The aircraft is 0.38 minutes = 24 seconds

EstimatedTimeRemaining from Waypoint 2. There are 1.40 NMiles RemainingDistance in Flight Plan leg 1.

FLIGHT PLAN WAYPOINT

VTUK to VTPP :FlightPlanTitle
4 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 8000 :FlightPlanCruisingAltitude

FlightPlanWaypoint

Idx	ICAO 111	Ident	Alt	Type	Mag Hdg	Spd Est	Dist	Dist Ttl	Dist Rem	Rem Dist	Rem Ttl Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A VTUK	VTUK	670	1	0	200	0.00	0.0	144.8	0.0	0.0	0.00	0.00	0.00	10.72	241.6	0.0	0.0
1	VVT CMP	CMP	650	3	284	200	46.60	46.6	98.2	1.4	1.4	13.97	0.00	0.40	10.99	0.0	10.2	0.0
2	VVT PCB	PCB	449	3	274	200	45.58	92.2	52.6	45.6	47.0	13.67	0.00	13.00	11.21	0.0	10.0	0.0
3	A VTPP	VTPP	145	1	277	200	52.59	144.8	0.0	52.6	99.6	15.77	0.00	15.00	11.46	0.0	11.5	0.0

ActiveWaypoint changes to 2 when EstimatedTimeRemaining = 0.38 (23 seconds) at which point, the aircraft is 1.34 NMiles from Waypoint 2. At the instant ActiveWaypoint changes, Waypoint 1 RemainingDistance becomes 0.0 and the 1.34 NMiles is moved to Waypoint 2, as demonstrated in the table below. Additionally, Waypoint Index 1 EstimatedTimeRemaining becomes 0.0. Waypoint Index 2 EstimatedTimeRemaining increases from 13.00 to 13.37 minutes to accommodate the additional 1.3 NMiles.

ActiveWaypoint Change Anticipation Time

FLIGHT PLAN WAYPOINT

VTUK to VTPP :FlightPlanTitle
4 :FlightPlanWaypointsNumber 2 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 8000 :FlightPlanCruisingAltitude

FlightPlanWaypoint

Idx	ICAO 111	Ident	Alt	Type	Mag Hdg	Spd Est	Dist	Dist Ttl	Dist Rem	Rem Dist	Rem Ttl Dist	ETE	ATE	Est Time Rem	ETA	Fuel Rem @ Arvl	Est Fuel Cons	Act Fuel Cons
0	A VTUK	VTUK	670	1	0	200	0.00	0.0	144.8	0.0	0.0	0.00	0.00	0.00	10.72	241.6	0.0	0.0
1	VVT CMP	CMP	650	3	284	200	46.60	46.6	98.2	0.0	0.0	13.97	15.93	0.00	10.99	228.8	10.2	12.8
2	VVT PCB	PCB	449	3	274	200	45.58	92.2	52.6	1.3	1.3	13.67	0.00	13.37	11.21	0.0	10.0	0.0
3	A VTPP	VTPP	145	1	277	200	52.59	144.8	0.0	52.6	53.9	15.77	0.00	15.00	11.46	0.0	11.5	0.0

A second later, Waypoint Index 2 RemainingDistance is updated by adding Waypoint Index 2 distance, 45.6 NMiles, which results in Waypoint 2 RemainingDistance = 46.9 NMiles.

```
FLIGHT PLAN WAYPOINT

VTUK to VTPP :FlightPlanTitle
 4 :FlightPlanWaypointsNumber    2 :FlightPlanActiveWaypoint
 0 :FlightPlanRouteType          2 :FlightPlanFlightPlanType     8000 :FlightPlanCruisingAltitude

------------------------------------------------- FlightPlanWaypoint -------------------------------------------------
                                                                                  Rem                  Est        Fuel   Est   Act
     ICAO     111                        Mag  Spd         Dist  Dist   Rem    Ttl                  Time       Rem @  Fuel  Fuel
Idx  123456789012  Ident   Alt Type  Hdg  Est   Dist    Ttl   Rem   Dist   Dist    ETE   ATE     Rem   ETA  Arvl  Cons  Cons
0    A      VTUK    VTUK   670    1    0  200   0.00    0.0 144.8    0.0    0.0   0.00  0.00    0.00 10.72 241.6   0.0   0.0
1    VVT    CMP      CMP   650    3  284  200  46.60   46.6  98.2    0.0    0.0  13.97 15.93    0.00 10.99 228.8  10.2  12.8
2    VVT    PCB      PCB   449    3  274  200  45.58   92.2  52.6   46.9   46.9  13.67  0.00   13.37 11.21   0.0  10.0   0.0
3    A      VTPP    VTPP   145    1  277  200  52.59  144.8   0.0   52.6   99.5  15.77  0.00   15.00 11.46   0.0  11.5   0.0
```

It's tedious to go through the steps of a waypoint change like this, but it's important to understand because Turn Anticipation, or, more to the point, Waypoint Change Anticipation affects all En Route Waypoints and Approach Segments and Sub-Segments in fs9gps.

❑ FlightPlanWaypointTimeZoneDeviation (minutes) [Get]

Because ETA is in Local Time, Time Zone Deviation is necessary to adjust to Local Time for flights that cross time zone boundaries.

❑ FlightPlanWaypointETE (minutes) [Get]

FlightPlanWaypointETE is the estimated en route time to the currently indexed waypoint. This estimate is calculated using FlightPlanWaypointSpeedEstimate which in turn, is derived from the aircraft cruising airspeed found in the aircraft.cfg file incorporating wind speed and direction.

FlightPlanWaypointETE is established when the flight plan is loaded and does not change during flight regardless of the groundspeed, position, or heading of the aircraft.

❑ FlightPlanWaypointATE (minutes) [Get]

FlightPlanWaypointATE is the actual elapsed time from ActiveWaypoint change to ActiveWaypoint change. Before reaching the next Waypoint, or, more precisely, before the ActiveWaypoint changes, WaypointATE returns zeros.

❑ FlightPlanWaypointEstimatedTimeRemaining (minutes) [Get]

FlightPlanWaypointEstimatedTimeRemaining is the time remaining until the currently indexed waypoint is reached. It is calculated by dividing WaypointDistanceRemaining by the current aircraft ground speed (A:GROUND VELOCITY). EstimatedTimeRemaining is associated with individual flight plan legs and is not cumulative involving multiple legs.

For the active waypoint, that is, for the flight plan leg currently being flown, EstimatedTimeRemaining is the remaining time from the aircraft's current position to the

next Waypoint, so it counts down as the aircraft flies toward that Waypoint. For Waypoints beyond that, it is the time required to fly the total length of that leg at the current groundspeed. For Waypoints already passed, EstimatedTimeRemaining is 0.0.

❑ FlightPlanWaypointETA (hours) [Get]

FlightPlanWaypointETA is the estimated time of arrival at the currently indexed Waypoint. It is a Local Time reference, so the most sensible units are probably Hours. Some points to consider:

- ❑ WaypointETA for the currently indexed Waypoint is calculated by adding WaypointEstimatedTimeRemaining for the currently indexed Waypoint to Local Time (e.g., EstimatedTimeRemaining + `E:LOCAL TIME`).

- ❑ There is a slightly different rule, however, that applies to WaypointIndex 0. WaypointETA for WaypointIndex 0, the departure airport, is 0.00 while the aircraft is on the ground. WaypointETA is set to `E:LOCAL TIME` at the moment the aircraft becomes airborne, when `A:SIM ON GROUND, bool` = 0. It does not matter where the aircraft starts its flight plan: at a Parking Gate, on the Active Runway, lots of taxiing or little taxiing, WaypointETA for WaypointIndex 0 is 0.00 until takeoff.

- ❑ When the aircraft passes a Waypoint (or, being precise, when ActiveWaypoint changes), WaypointETA equals Local Time, and it does not change after that regardless of the groundspeed, position, or heading of the aircraft because EstimatedTimeRemaining for a Waypoint that has been passed is zero. In other words, WaypointETA becomes Waypoint Actual Time of Arrival when a Waypoint is passed.

❑ FlightPlanWaypointFuelRemainedAtArrival (gallons) [Get]

Fuel calculations are based on fuel consumption rates derived from the aircraft model design and configuration and fuel quantity values. Fuel consumption rates and quantity can be accessed from A:Vars, for example, `(A:ENG1 FUEL FLOW GPH, gallons per hour)` and `(A:FUEL TOTAL QUANTITY, gallons)`.

❑ FlightPlanWaypointEstimatedFuelConsumption (gallons) [Get]

Fuel calculations are based on fuel consumption rates derived from the aircraft model design and configuration and fuel quantity values. Fuel consumption rates and quantity can be accessed from A:Vars, for example, `(A:ENG1 FUEL FLOW GPH, gallons per hour)` and `(A:FUEL TOTAL QUANTITY, gallons)`.

❑ FlightPlanWaypointActualFuelConsumption (gallons) [Get]

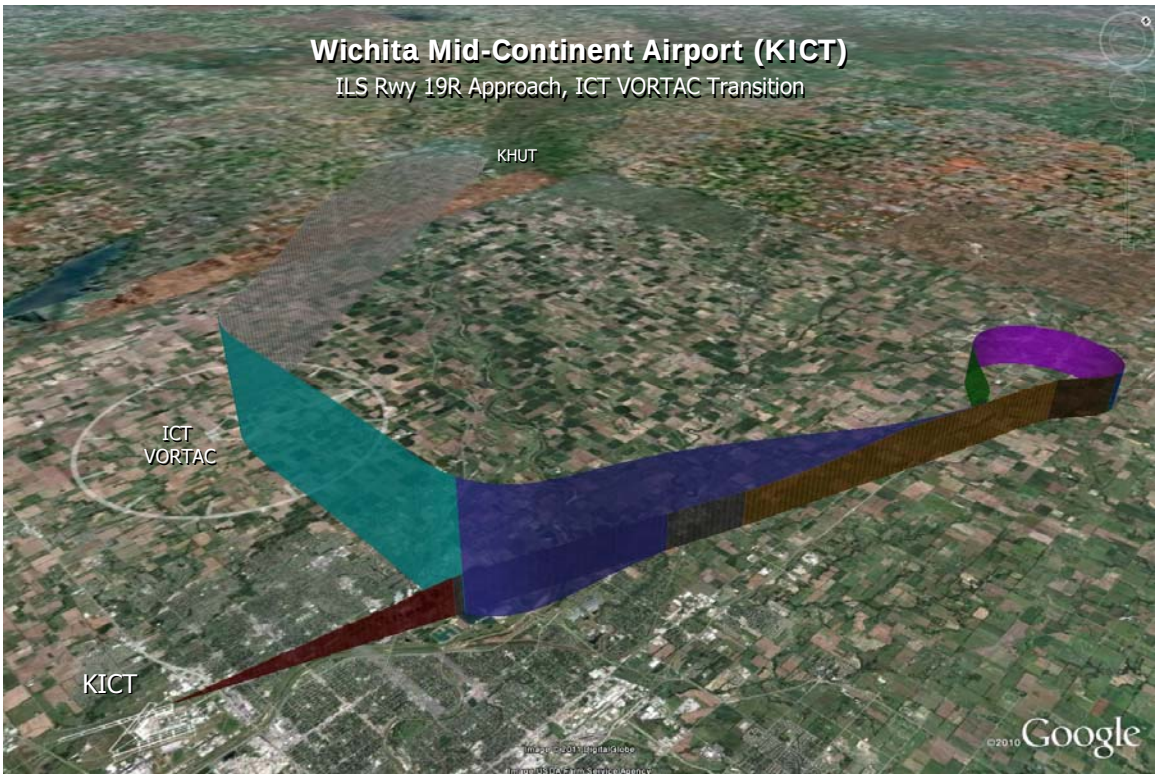
Fuel calculations are based on fuel consumption rates derived from the aircraft model design and configuration and fuel quantity values. Fuel consumption rates and quantity can be accessed from A:Vars, for example, `(A:ENG1 FUEL FLOW GPH, gallons per hour)` and `(A:FUEL TOTAL QUANTITY, gallons)`.

Instrument Approaches

Variables of the FlightPlanApproach and FlightPlanWaypointApproach groups define the flight path according to Instrument Approach Procedures from the en route approach transition point through the approach procedure, to the landing point, and finally to the missed approach procedure and holding pattern.

Throughout the discussion of Approach variables, an example instrument flight from Hutchinson Municipal Airport (Hutchinson, Kansas, USA, "KHUT") to Wichita Mid-Continent Airport (Wichita, Kansas, USA, "KICT") is used, incorporating the ILS Rwy 19R Approach, ICT VORTAC Transition into Wichita.

In the example shown below, the aircraft departs Hutchinson and proceeds to the Transition Fix, ICT VORTAC. The Approach Transition in this particular simulation is flown at 7000' altitude and from there, according to the descent profile for the approach. Between rotate and flare, the aircraft is flown by the stock FS9 Bendix-King Radio Autopilot with the user controlling altitude except on Final Approach. Approach segment and sub-segment colors match those used in the discussion of the KICT ILS Rwy 19 Approach in this section.

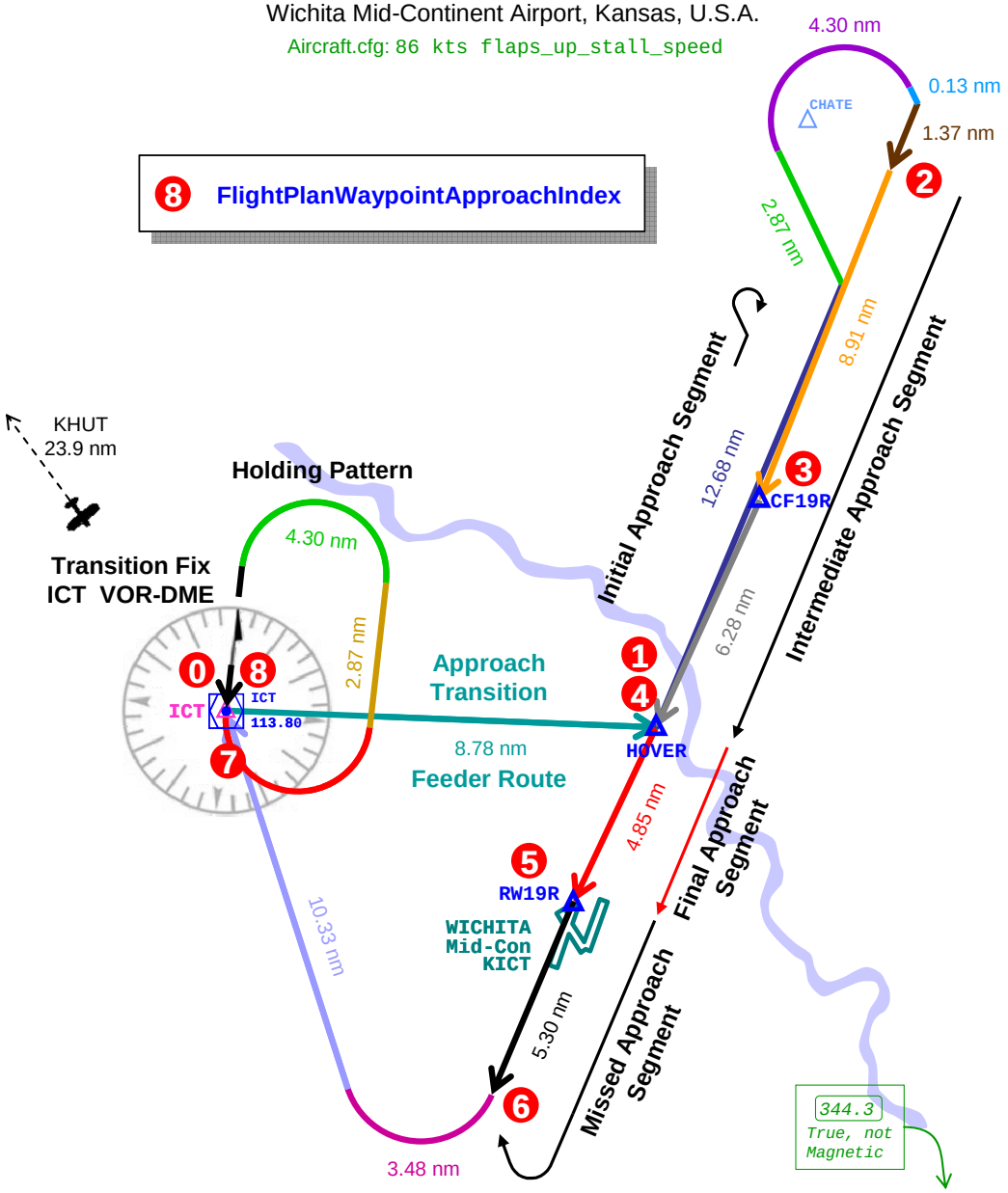


FS9 Transition and Approach Segments

KICT ILS Rwy 19R; ICT Transition

Wichita Mid-Continent Airport, Kansas, U.S.A.

Aircraft.cfg: `86 kts flaps_up_stall_speed`




```
FLIGHT PLAN WAYPOINT
KHUT to KICT :FlightPlanTitle   3000 :FltPlnCruisingAltitude
  1 :FltPlnActiveWaypoint          3 :FltPlnWaypointsNumber     1 :FltPlnRouteType     2 :FltPlnFlightPlanType
------------------------------------------ FlightPlanWaypoint ------------------------------------------

     ICAO     111                          Mag  Spd          Dist  Dist    Rem                    Est        Fuel   Est   Act
Idx  123456789012  Ident   Alt Type  Hdg  Est   Dist   Ttl   Rem   Dist   ETE   ATE   Time  ETA  Rem @ Fuel  Cons  Cons
0    A       KHUT    KHUT  1541    1    0   200   0.00   0.0  33.2   0.00  0.00  0.00  0.00  0.00 242.0  0.0   0.0
1    VK3     ICT      ICT  1470    3  138   200  23.95  23.9   9.3  23.95  7.17  0.00  0.00  0.00   0.0  4.3   0.0
2    A       KICT    KICT  1332    1  128   200   9.29  33.2   0.0   9.29  2.78  0.00  0.00  0.00   0.0  1.7   0.0

FLIGHT PLAN WAYPOINT APPROACH
 13 :FltPlnApprType   9 :ApprWptsNum   0 :ActiveApprWpt   ILS 19R :FltPlnApprName      ICT :FltPlnApprTransName
------------------------------------------ FlightPlanWaypointApproach ------------------------------------------

     ICAO     111                                                          Leg    Leg          Rem
Idx  123456789012   Name Type Mode        Lat          Lon          Alt  Trgt  Leg Dist  Ttl Dist  From Dist  Rem Dist  Ttl Dist  Course
0    VK3     ICT      ICT    1    1   37.745233  -97.583825         0     0      0.00      0.00      83.61      0.00      0.00    -1.0
1    WK3KICTHOVER  HOVER    1    1   37.737564  -97.398989      3500     0      8.78      8.78      83.61      8.78      8.78    93.1
2    WK3KICTHOVER  HOVER    3    1   37.977508  -97.297155      3500     0     21.35     30.13      74.83     21.35     30.13   328.0
3    WK3KICTCF19R  CF19R    1    2   37.835872  -97.353867      3500     0      8.91     39.04      53.48      8.91     39.04   197.5
4    WK3KICTHOVER  HOVER    1    2   37.737564  -97.398989      3000     0      6.28     45.32      44.56      6.28     45.32   193.0
5    RK3KICTRW19R  RW19R    1    2   37.661604  -97.433817      1382     0      4.85     50.17      38.29      4.85     50.17   193.0
6                           9    3   37.578144  -97.470076      3500  3500      5.30     55.46      33.44      5.30     55.46   193.0
7    VK3     ICT      ICT    1    3   37.745233  -97.583825      3500     0     13.81     69.27      28.14     13.81     69.27   344.3
8    VK3     ICT      ICT    6    3   37.745233  -97.583825      3500     0     14.34     83.61      14.34     14.34     83.61   180.0
```

KICT
WICHITA MID-CONTINENT

JEPPESEN
11-4

WICHITA, KAN
ILS Rwy 19R

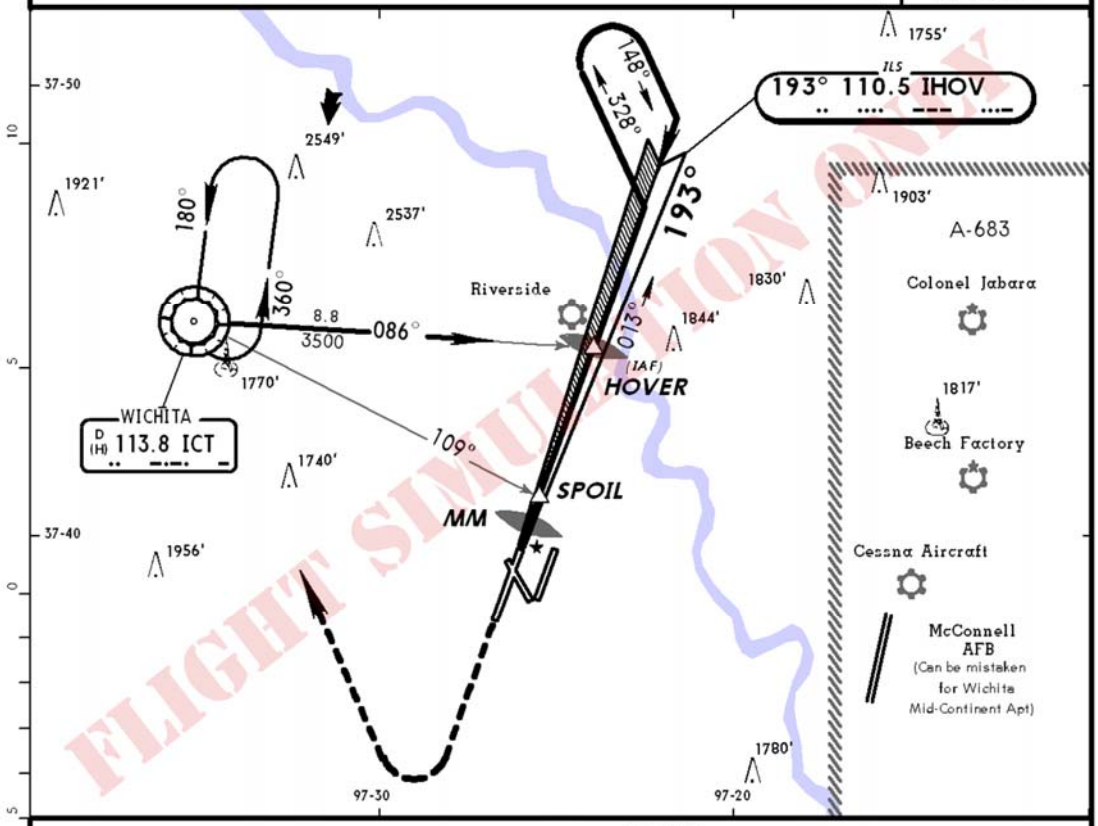
ATIS	WICHITA Approach (R)	WICHITA Tower	Ground
125.15	126.7	118.2	121.9

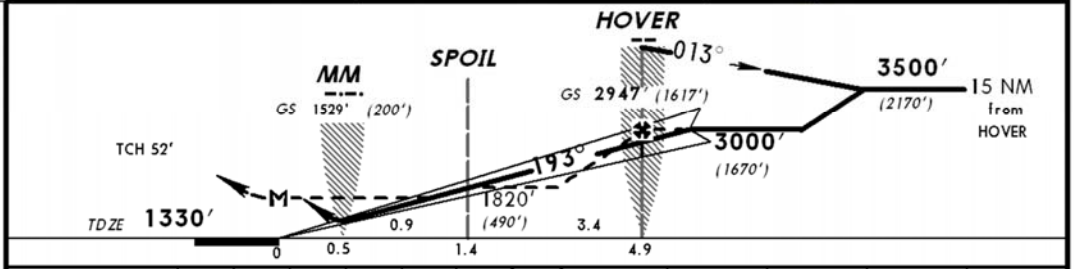
BRIEFING STRIP

LOC IHOV	Final Apch Crs	GS HOVER	ILS DA(H)	Apt Elev / TDZE
110.5	193°	2947′ (1617′)	1530′ (200′)	1333′ / 1330′

MISSED APCH: Climb to 3500′ then RIGHT turn direct ICT VOR and hold.

4100′
MSA ICT VOR





Gnd speed-Kts	70	90	100	120	140	160
GS 3.00°	377	485	539	647	754	862
HOVER to MAP 4.9	4:12	3:16	2:56	2:27	2:06	1:50

MALSR | 3500′ | RT | D | ICT 113.8

STRAIGHT-IN LANDING RWY 19R

ILS DA(H) 1530′ (200′)

	FULL	RAIL or ALS out
A, B, C	RVR 24 or 1/2	RVR 40 or 3/4
D	RVR 40 or 3/4	RVR 50 or 1

LOC (GS out)

	MDA(H) 1660′ (330′) With Spoil			MDA(H) 1820′ (490′) Without Spoil		
		RAIL out	ALS out		RAIL out	ALS out
A, B	RVR 24 or 1/2	RVR 40 or 3/4	RVR 50 or 1	RVR 24 or 1/2	RVR 40 or 3/4	RVR 50 or 1
C				RVR 40 or 3/4	RVR 60 or 1 1/4	
D	RVR 40 or 3/4	RVR 50 or 1		RVR 50 or 1	1 1/2	

CIRCLE-TO-LAND

Max Kts	With Spoil MDA(H)	Without Spoil MDA(H)
90, 120	1800′ (467′) -1	1820′ (487′) -1
140	1800′ (467′) -1 1/2	1820′ (487′) -1 1/2
165	1900′ (567′) -2	1900′ (567′) -2

❑ FlightPlanApproachWaypointType (enum) [Get]

FlightPlanApproachWaypointType is an enum referring to the navigation procedure objective of the currently *active* (as opposed to currently indexed) approach segment.

Approach Waypoint Type

Bit	Name and Type #	Bit	Name and Type #	Bit	Name and Type #
0	NONE = 0	4	DME_ARC_LEFT = 4	8	DISTANCE = 8
1	FIX = 1	5	DME_ARC_RIGHT = 5	9	ALTITUDE = 9
2	PROC_TURN_LEFT = 2	6	HOLDING_LEFT = 6	10	MANUAL_SEQ = 10
3	PROC_TURN_RIGHT = 3	7	HOLDING_RIGHT = 7	11	VECTORS_TO_FINAL = 11

http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS_APPROACH_WAYPOINT_TYPE

❑ FlightPlanApproachMode (enum) [Get]

FlightPlanApproachMode is a number used to describe the *active* segment of the approach. ApproachMode has some similarity to US F.A.A. Approach Segment nomenclature as demonstrated below.

Approach Mode

Bit	Name and Type #	Bit	Name and Type #
0	NONE = 0	2	FINAL = 2
1	TRANSITION = 1	3	MISSED = 3

http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS_APPR_TYPE

F.A.A. Instrument Approach Procedure Segments	FS9 FlightPlanApproachMode		F.A.A. Instrument Approach Procedure Segments	FS9 FlightPlanApproachMode	
En Route	0	NONE	Final Approach	2	FINAL
Feeder Route	1	TRANSITION	Missed Approach	3	MISSED
Initial Approach	1	TRANSITION	Holding Pattern	3	MISSED
Intermediate Approach	2	FINAL			

❑ FlightPlanApproachSegmentType (enum) [Get]

FlightPlanApproachSegmentType is a number indicating the direction of turn within an approach segment.

- ❑ 0 = No turn
- ❑ 1 = Right turn
- ❑ 2 = Left turn

Approach Segments and Sub-Segments: Approach segments containing both turns and straight sections *within the segment* are divided into sub-segments. SegmentType is applied to the sub-segment representing the continuous turn (e.g., procedure turn, missed approach turn to holding fix, holding pattern turns).

FlightPlanApproachSegmentType is an intra-segment variable and does not apply to turns from one straight approach segment to the next such as the turn to the outbound initial approach segment that occurs when the aircraft reaches the Initial Approach Fix.

Confusing without a picture, so refer to the detailed dissection of KICT ILS 19R ICT Transition at the end of this section for additional clarification.

❑ FlightPlanApproachSegmentDistance (nmiles) [Get]

FlightPlanApproachSegmentDistance is the remaining distance within the currently indexed approach segment or sub-segment between the aircraft position and the termination point of the segment. It counts downs as the aircraft proceeds toward the segment termination point.

ApproachSegmentDistance can be used to measure and keep track of the length and remaining distance of sub-segments whereas WaypointApproachLegDistance and WaypointApproachRemainingDistance do not get into the sub-segment level.

Also handy is that ApproachSegmentDistance and ApproachSegmentLength are also active during the En Route flight phase, returning the same values as WaypointRemainingDistance and WaypointDistance, respectively.

❑ FlightPlanApproachSegmentLength (nmiles) [Get]

FlightPlanApproachSegmentLength is the total length of the indexed approach segment. When the approach segment involves turns, (e.g., a procedure turn or missed approach turn to a holding fix) ApproachSegmentLength is the total length of the active sub-segment. As discussed later on, sub-segments do not have separate index pointers.

❑ FlightPlanApproachIsWaypointRunway (bool) [Get]

Final Approach to a Runway. FlightPlanApproachIsWaypointRunway equals 1 when the active approach segment is the Final Approach segment and the termination point is a destination runway waypoint (e.g., RW18). Also, FlightPlanWaypointApproachMode = 2 (Final).

For all other approach segments, ApproachIsWaypointRunway equals 0.

❑ FlightPlanApproachAirportIdent (string) [Get]

FlightPlanApproachAirportIdent is the Ident of the Approach procedure destination airport.

❑ FlightPlanApproachType (enum) [Get]

FlightPlanApproachType is an enum representing the type of approach procedure.

FlightPlanApproachType

#	Approach Type	#	Approach Type	#	Approach Type
0	UNKNOWN = 0	5	LORAN = 5	10	LDA = 10
1	VFR = 1	6	RNAV = 6	11	LOC = 11
2	HEL = 2	7	VOR = 7	12	MLS = 12
3	TACAN = 3	8	GPS = 8	13	ILS = 13
4	NDB = 4	9	SDF = 9		

http://msdn.microsoft.com/en-us/library/cc526954.aspx#AirportApproachType

In the KICT example, it is an ILS approach, FlightPlanApproachType = 13.

❑ FlightPlanApproachIndex (enum) [Get]

FlightPlanApproachIndex is an enum representing the selected approach for the destination airport. In the KICT example used throughout this section, ILS 19R approach has been selected, consequently FlightPlanApproachIndex = 3.

Approaches available for Wichita Mid-Continent Airport (KICT):

0 BLOC 19L
1 ILS 01L
2 ILS 01R
3 ILS 19R
4 NDB 01R (gps)
5 GPS 19L
6 GPS 32
7 RNAV 01L (gps)
8 RNAV 19R (gps)
9 VOR 14 (gps)

❑ FlightPlanApproachName (string) [Get]

FlightPlanApproachName is the name of the selected approach. In the KICT example, "**ILS 19R**".

❑ FlightPlanApproachTransitionIndex (enum) [Get]

FlightPlanApproachTransitionIndex is an enum representing the desired transition for the previously selected approach. In the KICT example used throughout this section, ILS 19R approach has been selected utilizing the ICT transition, consequently FlightPlanApproachTransitionIndex = 1.

Transitions available for the ILS 19R approach at Wichita Mid-Continent Airport (KICT):

0 VECTORS
1 ICT
2 HOVER

❑ FlightPlanApproachTransitionName (string) [Get]

FlightPlanApproachTransitionName is the name of the selected transition. In the KICT example, "**ICT**".

❑ FlightPlanIsApproachFinal (bool) [Get]

FlightPlanIsApproachFinal is a bool equaling 1 when FlightPlanWaypointApproachIndex 0 is the Final Approach Fix and WaypointApproachIndex 1 is the destination runway waypoint. The normal circumstance for this is a Vectors-To-Final Transition. Other than this situation, FlightPlanIsApproachFinal equals 0.

For a Vectors-To-Final Transition, fs9gps adds a 5.00 NMile sub-segment onto the Final Approach segment. The heading is the same as the Final Approach segment, and it is placed outboard of the Final Approach Fix. The sub-segment provides room to accommodate a turn to the Final Approach heading prior to reaching the Final Approach Fix.

Unlike the other sub-segments discussed in this section, the length of a Vectors-To-Final sub-segment is not a function of `flaps_up_stall_speed` specified in the aircraft.cfg file. It appears to always be 5.00 NMiles.

FlightPlanIsApproachFinal is set through use of FlightPlanNewApproachTransition or FlightPlanLoadApproach.

❑ FlightPlanIsApproachMissed (bool) [Get]

FlightPlanIsApproachMissed is a Boolean representing whether or not the Missed Approach procedure is included in the Approach procedure.

- ❑ 0 No Missed Approach procedure included
- ❑ 1 Missed Approach procedure is included in Approach

It is set through use of FlightPlanNewApproachMissed.

❑ FlightPlanApproachWaypointsNumber (enum) [Get]

FlightPlanApproachWaypointsNumber is the number of segments in the selected approach and transition. In the KICT ILS 19R / ICT Transition example used throughout this section, there are 9 approach segments (Index 0 through 8) going from the En route Fix that defines the Transition (Index 0) through the Holding Pattern at the Missed Approach Holding Waypoint (Index 8).

Note that when the sub-segments that define intra-segment turns are included, there is a combined total of 16 segments plus sub-segments.

❑ FlightPlanWaypointApproachIndex (enum) [Get, Set]

FlightPlanWaypointApproachIndex is the index pointer for the WaypointApproach variables. The WaypointApproach variables describe length, location, direction, etc., attributes of the approach segments.

❑ FlightPlanWaypointApproachICAO (string) [Get]

The ICAO ident of the currently indexed segment. Approach segments are defined by their termination points, so FlightPlanWaypointApproachICAO, ApproachName, ApproachLatitude and Longitude, and ApproachAltitude refer to the termination point of the segment.

❑ FlightPlanWaypointApproachName (string) [Get]

FlightPlanWaypointApproachName is the navaid ident or runway name associated with the termination point of the currently indexed approach segment.

❑ FlightPlanWaypointApproachType (enum) [Get]

FlightPlanWaypointApproachType is an enum referring to the navigation procedure objective of the currently indexed (but not necessarily the currently active) approach segment.

Waypoint Approach Type

Bit	Name and Type #	Bit	Name and Type #	Bit	Name and Type #
0	NONE = 0	4	DME_ARC_LEFT = 4	8	DISTANCE = 8
1	FIX = 1	5	DME_ARC_RIGHT = 5	9	ALTITUDE = 9
2	PROC_TURN_LEFT = 2	6	HOLDING_LEFT = 6	10	MANUAL_SEQ = 10
3	PROC_TURN_RIGHT = 3	7	HOLDING_RIGHT = 7	11	VECTORS_TO_FINAL = 11

http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS_APPROACH_WAYPOINT_TYPE

❑ FlightPlanWaypointApproachMode (enum) [Get]

FlightPlanWaypointApproachMode is a number used to describe the *currently indexed* segment of the approach.

Waypoint Approach Mode

Bit	Name and Type #	Bit	Name and Type #
0	NONE = 0	2	FINAL = 2
1	TRANSITION = 1	3	MISSED = 3

http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS_APPR_TYPE

- **FlightPlanWaypointApproachLatitude**
- **FlightPlanWaypointApproachLongitude (degrees) [Get]**

The latitude and longitude of the termination point – the waypoint - of the currently indexed approach segment. The termination point of the segment and the waypoint are one and the same.

- **FlightPlanWaypointApproachAltitude (feet) [Get]**

The altitude of the currently indexed approach waypoint.

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

For straight approach segments, FlightPlanWaypointApproachCourse is the bearing of the approach segment pointing toward the termination point.

Unfortunately, fs9gps returns a mixture of true and magnetic bearings in what appears to be a software bug. They should be magnetic. Interestingly, the CustomDraw utility within fs9gps renders the flight path correctly, but the ApproachCourse variable that an autopilot accesses is not always the magnetic course that an autopilot expects, causing some turns to initially be over or under executed.

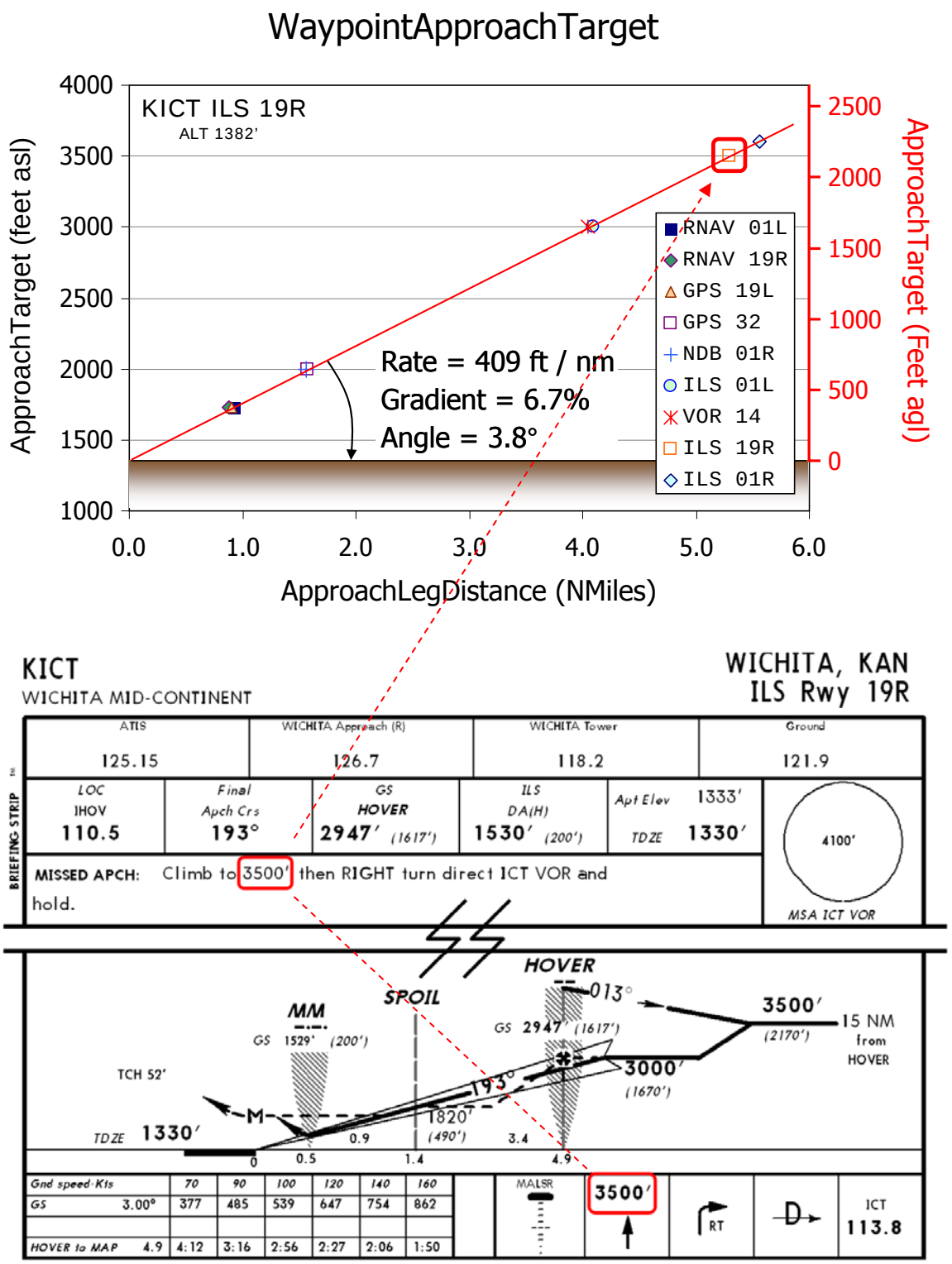
For segments that include turn sub-segments, the following applies:

- **Procedure Turn**: ApproachCourse is the magnetic bearing of the 45° turn. In the KICT ILS Rwy 19R example, 328°. Fs9gps correctly returns 328° M.
- **Missed Approach Turn toward Holding Fix**: ApproachCourse is the bearing of the culmination of the Missed Approach turn. Fs9gps returns a true bearing for this sub-segment. In the KICT ILS 19R Missed Approach example, this causes the aircraft to turn 7.1° too far to the north before ultimately turning direct to the Holding Fix, resulting in a "U" shaped Missed Approach Turn being flown rather than the smooth turn intended.
- **Holding Pattern Turn**: ApproachCourse is the bearing of the final sub-segment of the Holding Pattern, the segment that terminates at the Holding Fix. In the KICT ILS Rwy 19R example, it is 180° M. Fs9gps correctly returns 180°M.

The WaypointApproachCourse bearings listed in *green italic* font in the FS9 Transitions and Approach Segments ILS 19R approach diagram (Page 158) are degrees True, but fs9gps should have returned degrees Magnetic.

- **FlightPlanWaypointApproachTarget (feet) [Get]**

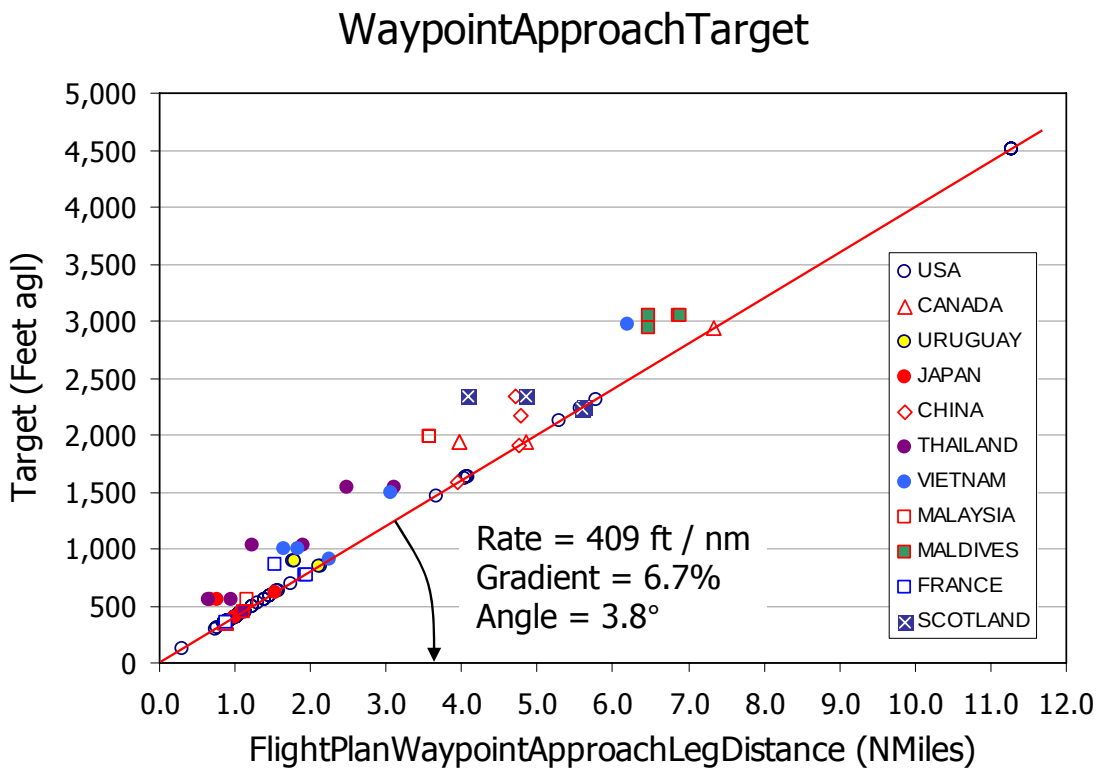
FlightPlanWaypointApproachTarget is the Missed Approach straight climb out target altitude. It is a function of ApproachLegDistance, at about a 3.8° climb angle, as shown in the graph below. The graph plots the ApproachTarget - LegDistance pairs for the Wichita, Kansas U.S.A. (KICT) airport approaches.



The approach segment that contains WaypointApproachTarget will be the first approach segment with WaypointApproachMode = 3. It may or may not have an ICAO or Name. At minimum, the Target waypoint has Lat and Lon, Type and Mode, which are sufficient information to define an Approach Waypoint.

In practical fs9gps terms, WaypointApproachTarget may be the altitude that FS9 ATC gives in its missed approach instructions.

The figure below shows a very small sample of WaypointApproachTarget from various approaches around the world. Many have a steeper Missed Approach climb-out rate, but it appears that there is a consistent minimum rate of 409 ft / NMile = 3.8°. There is also a noteworthy population of low altitude WaypointApproachTarget values. As a rule, an aircraft should not turn on climb-out under 400 feet agl, but that doesn't really explain why fs9gps has so many WaypointApproachTarget values around 500 feet agl.



❑ FlightPlanWaypointApproachLegDistance (nmiles) [Get]

FlightPlanWaypointApproachLegDistance is the length of the currently indexed approach segment. For approach segments that involve turns within the segment, such as a procedure turn, ApproachLegDistance is the combined flight distance of all parts, or sub-segments, of the approach segment, including the turn. Refer to the KICT ILS Rwy 19R ICT Transition segment discussion at the end of this section for further clarification.

FlightPlanWaypointApproachIndex = 0 is the En Route Fix and represents the starting point of the approach transition. ApproachLegDistance for this point is 0.00 before the approach is activated. Once activated, ApproachLegDistance for WaypointApproachIndex = 0 becomes the remaining distance from the aircraft position to the En Route Fix. This is the same as FlightPlanWaypointRemainingDistance of the active waypoint when the approach is activated.

❑ FlightPlanWaypointApproachLegTotalDistance (nmiles) [Get]

FlightPlanWaypointApproachLegTotalDistance is the cumulative distance of all Approach segments starting at WaypointApproachIndex = 0 through the currently indexed Approach Waypoint. When the index points to the last Approach Waypoint (the Holding fix usually), LegTotalDistance is the total length of the flight measured along flight segments, including one circuit around the holding pattern.

FlightPlanWaypointApproachLegTotalDistance of the Approach phase is analogous to FlightPlanWaypointDistanceTotal of the en route phase.

❑ FlightPlanWaypointApproachLegFromDistance (nmiles) [Get]

FlightPlanWaypointApproachLegFromDistance is the distance from the currently indexed Approach Segment to the termination point of the last Approach Segment.

FlightPlanWaypointApproachLegFromDistance of the Approach phase is analogous to FlightPlanWaypointDistanceRemaining of the en route phase.

❑ FlightPlanWaypointApproachRemainingDistance (nmiles) [Get]

FlightPlanWaypointApproachRemainingDistance is the segment distance remaining to be flown. For the active approach segment, that is, for the segment currently being flown, it is the remaining distance from the aircraft's current position to the termination point of the current segment. For segments beyond that, it is just the total length of the approach segment. For approach segments already passed, WaypointApproachRemainingDistance is 0.0.
FlightPlanWaypointApproachRemainingDistance is a segment measurement; it does not measure distances of *sub*-segments. FlightPlanApproachSegmentDistance does that.

FlightPlanWaypointApproachRemainingDistance of the Approach phase is analogous to FlightPlanWaypointWaypointRemainingDistance of the en route phase.

❑ FlightPlanWaypointApproachRemainingTotalDistance (nmiles) [Get]

FlightPlanWaypointApproachRemainingTotalDistance is the cumulative remaining distance from current aircraft position to the termination point of the indexed segment. It is the same as ApproachRemainingDistance when the indexed segment is the active segment. ApproachRemainingDistance is measured along flight plan segments; it is not a direct-to measurement.

FlightPlanWaypointApproachRemainingTotalDistance of the Approach phase is analogous to FlightPlanWaypointRemainingTotalDistance of the en route phase.

Miscellaneous

DISSECTING THE KICT ILS19R APPROACH SEGMENTS

A closer look at the construction of the approach segments found in the KICT example.

The table below lists the 9 waypoints associated with the ILS 19R Approach, ICT transition into KICT. Variable Segment and Sub-segment lengths are based on Flaps Up Stall Speed = 86.0 knots.

FLIGHT PLAN NEW APPROACH: KICT

```
      9 :ApprWaypointsNumber             13 :FlightPlanApprType          8 :FlightPlanWaypointApproachIndex
 ILS 19R :FlightPlanApprName            ICT :FlightPlanApprTransName     0 :FlightPlanActiveApproachWaypoint
      0 :FlightPlanIsActiveApproach       3 :FlightPlanApproachIndex     1 :FlightPlanApproachTransitionIndex
```

FlightPlanWaypointApproach

Idx	ICAO 111 123456789012	Name	Type	Mode	Latitude (Deg Min)	Longitude (Deg Min)	Latitude (Degrees)	Longitude (Degrees)	Alt	Trgt	Leg Dist	Course (mag)
0	VK3 ICT	ICT	1	1	37 44.7140	-97 35.0295	37.745233	-97.583825	0	0	0.00	-1.00
1	WK3KICTHOVER	HOVER	1	1	37 44.2538	-97 23.9393	37.737564	-97.398989	3500	0	8.78	93.06
2	WK3KICTHOVER	HOVER	3	1	37 58.6505	-97 17.8293	37.977508	-97.297155	3500	0	21.35	328.00
3	WK3KICTCF19R	CF19R	1	2	37 50.1523	-97 21.2320	37.835872	-97.353867	3500	0	8.91	197.52
4	WK3KICTHOVER	HOVER	1	2	37 44.2538	-97 23.9393	37.737564	-97.398989	3000	0	6.28	193.00
5	RK3KICTRW19R	RW19R	1	2	37 39.6962	-97 26.0290	37.661604	-97.433817	1382	0	4.85	193.00
6			9	3	37 34.6886	-97 28.2045	37.578144	-97.470076	3500	3500	5.30	193.00
7	VK3 ICT	ICT	1	3	37 44.7140	-97 35.0295	37.745233	-97.583825	3500	0	13.81	344.34
8	VK3 ICT	ICT	6	3	37 44.7140	-97 35.0295	37.745233	-97.583825	3500	0	14.34	180.00

The waypoint segments, sub-segments and flight path are demonstrated as follows:

FlightPlanWaypointApproachIndex 0:

Enroute Fix

FlightPlanActiveApproachWaypoint = 0 (the Index)
FlightPlanWaypointApproachType = 1 (Fix)
FlightPlanWaypointApproachMode = 1 (Transition)
FlightPlanApproachIsWaypointRunway = 0

En Route Fix Segment						0 Index	
	FlightPlanWaypointApproach					Leg	
Segment	Type	Mode	Altitude	Target	Course	Distance	
Enroute Fix	1	1	N/A (0')	N/A (0')	-001°	0.00	Enroute Fix (often is the Transition name)

Index 0 is the En Route Fix, which for this Approach Transition is the ICT VOR-DME.

FlightPlanWaypointApproachIndex 1:

Approach Transition

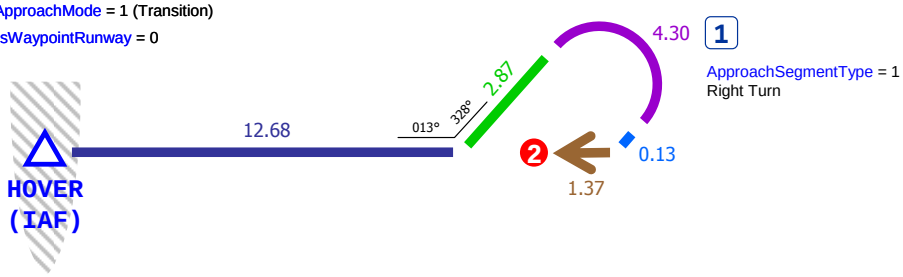
FlightPlanActiveApproachWaypoint = 1 (the Index)
FlightPlanWaypointApproachType = 1 (Fix)
FlightPlanWaypointApproachMode = 1 (Transition)
FlightPlanApproachIsWaypointRunway = 0

Approach Transition Segment						1 Index	
	FlightPlanWaypointApproach					Leg	
Segment	**Type**	**Mode**	**Altitude**	**Target**	**Course**	**Distance**	
Appr. Transition	1	1	3500	N/A (0')	093°	8.78	Begins at Enroute Fix. Ends at IAF - HOVER
						8.78 nm	:FlightPlanWaypointApproachLegDistance

FlightPlanWaypointApproachIndex 2:

Initial Approach

FlightPlanActiveApproachWaypoint = 2 (the Index)
FlightPlanWaypointApproachType = 3 (Procedure Turn Right)
FlightPlanWaypointApproachMode = 1 (Transition)
FlightPlanApproachIsWaypointRunway = 0

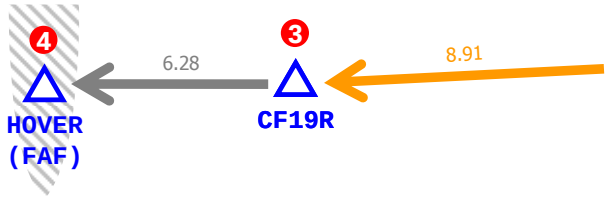


Initial Approach Segment						2 Index	
	FlightPlanWaypointApproach					Leg	
Sub-Segment	**Type**	**Mode**	**Altitude**	**Target**	**Course**	**Distance**	
Outbound	3	1	3500 ft	N/A (0')		12.68	Begins at Initial Approach Fix (IAF) - HOVER
45° Turn	3	1	3500 ft	N/A (0')	328°	2.87	328° = FlightPlanWaypointApproachCourse
180° Turn	3	1	3500 ft	N/A (0')		4.30	Right Turn. Note: ApproachSegmentType =1
45° Intercept	3	1	3500 ft	N/A (0')		0.13	
Inbound	3	1	3500 ft	N/A (0')		1.37	Ends at Intermed. Fix (IF) or Inbound to FAF
						21.35 nm	:FlightPlanWaypointApproachLegDistance

FlightPlanWaypointApproach Index 3 and 4:

Intermediate Approach

FlightPlanActiveApproachWaypoint = 3 and 4 (the Index)
FlightPlanWaypointApproachType = 1 (Fix)
FlightPlanWaypointApproachMode = 2 (Final)
FlightPlanApproachIsWaypointRunway = 0



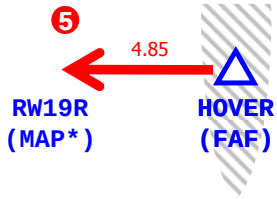
Intermediate Approach Segment 1						3 Index	
	FlightPlanWaypointApproach					Leg	
Segment	Type	Mode	Altitude	Target	Course	Distance	
Intermediate - 1	1	2	3500 ft	N/A (0')	198°	8.91	Begins at Intermed. Fix (IF) or Inbound to FAF
						8.91 nm	:FlightPlanWaypointApproachLegDistance

Intermediate Approach Segment 2						4 Index	
	FlightPlanWaypointApproach					Leg	
Segment	Type	Mode	Altitude	Target	Course	Distance	
Intermediate - 2	1	2	3000 ft	N/A (0')	193°	6.28	Ends at Final Approach Fix (FAF)
						6.28 nm	:FlightPlanWaypointApproachLegDistance

FlightPlanWaypointApproach Index 5:

Final Approach

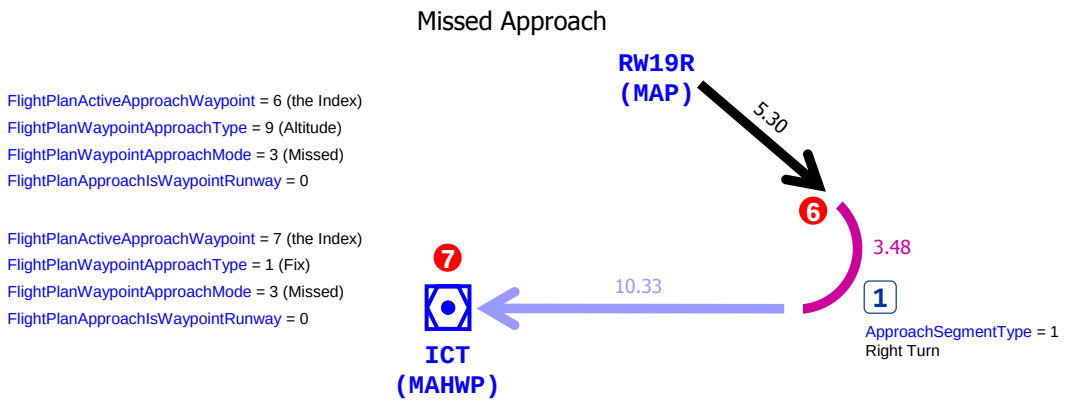
FlightPlanActiveApproachWaypoint = 5 (the Index)
FlightPlanWaypointApproachType = 1 (Fix)
FlightPlanWaypointApproachMode = 2 (Final)
FlightPlanApproachIsWaypointRunway = 1



* The Missed Approach Point

Final Approach Segment						5 Index	
	FlightPlanWaypointApproach					Leg	
Segment	Type	Mode	Altitude	Target	Course	Distance	
Final	1	2	1382 ft	N/A (0')	193°	4.85	Final Approach. Ends at MAP or Landing
						4.85 nm	:FlightPlanWaypointApproachLegDistance

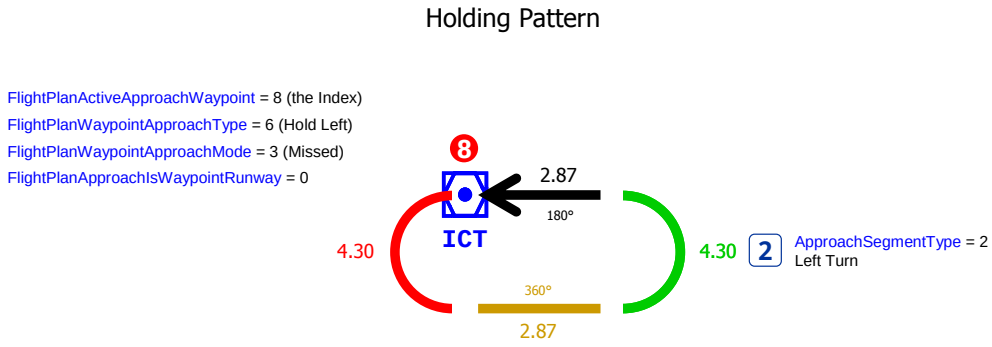
FlightPlanWaypointApproachIndex 6 and 7:



Missed Approach Climb-Out Segment						6 Index	
	FlightPlanWaypointApproach					Leg	
Segment	**Type**	**Mode**	**Altitude**	**Target**	**Course**	**Distance**	
Straight Climb-out	9	3	3500 ft	3500 ft	193°	5.30	Straight climb to Target altitude
						5.30 nm	:FlightPlanWaypointApproachLegDistance

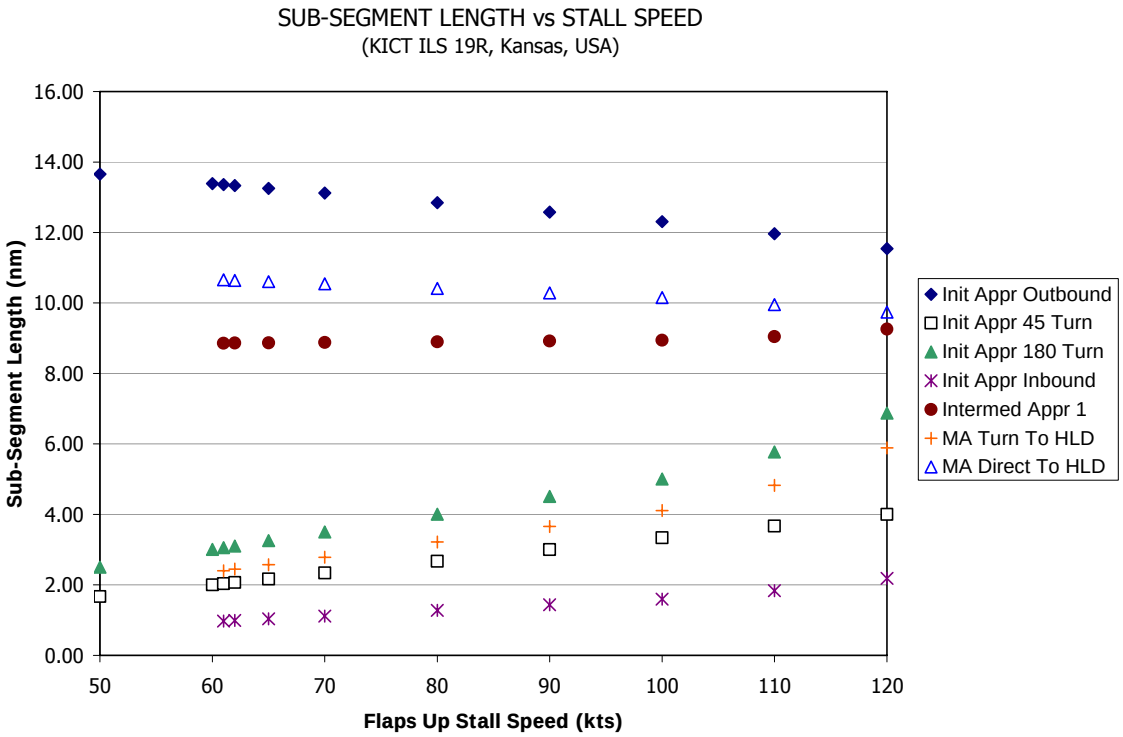
Missed Approach Holding Turn Segment						7 Index	
	FlightPlanWaypointApproach					Leg	
Sub-Segment	**Type**	**Mode**	**Altitude**	**Target**	**Course**	**Distance**	
Turn to Holding Fix	1	3	3500 ft	3500 ft	344°	3.48	Right turn toward Holding Fix. ApproachSegmentType =1
Direct to Holding Fix	1	3	3500 ft	3500 ft		10.33	Direct to Holding Fix.
						13.81 nm	:FlightPlanWaypointApproachLegDistance

FlightPlanWaypointApproachIndex 8:



Holding Pattern Segment						8 Index	
	FlightPlanWaypointApproach					Leg	
Sub-Segment	**Type**	**Mode**	**Altitude**	**Target**	**Course**	**Distance**	
180° Turn North	6	3	3500 ft	N/A (0')		4.30	Begins at MAHWP. ApproachSegmentType = 2
North leg	6	3	3500 ft	N/A (0')		2.87	
180° Turn South	6	3	3500 ft	N/A (0')		4.30	ApproachSegmentType = 2
South Leg	6	3	3500 ft	N/A (0')	180°	2.87	Ends at Missed Appr. Holding Waypoint (MAHWP)
						14.34 nm	:FlightPlanWaypointApproachLegDistance

SUB-SEGMENT LENGTH



The length of some approach sub-segments varies according to the flaps_up_stall_speed specified in the aircraft.cfg file. fs9gps apparently does this to accommodate the greater turning radius required as approach speeds increase.

The example above shows various sub-segment lengths associated with the KICT ILS19R Approach. The sub-segment names follow those used in the examples found in "**DISSECTING THE KICT ILS19R APPROACH SEGMENTS**".

Note that the turning sub-segment (e.g., ▲ Init Appr 180 Turn and + MA Turn To HLD) lengths increase with stall speed, while some of the straight sub-segments (e.g., ◆ Init Appr Outbound and △ MA Direct To HLD) decrease. The decrease is necessary to keep the overall procedure roughly the same size regardless of stall speed, and, in the case of a Procedure Turn, to try to keep the aircraft within the Manuevering Area. In the case of at least KICT ILS19R used as an example in this chapter, fs9gps does not comply with the requirement to complete the Procedure Turn within 15 NMiles of the IAF. The Initial Approach Outbound Leg sub-segment is too long.

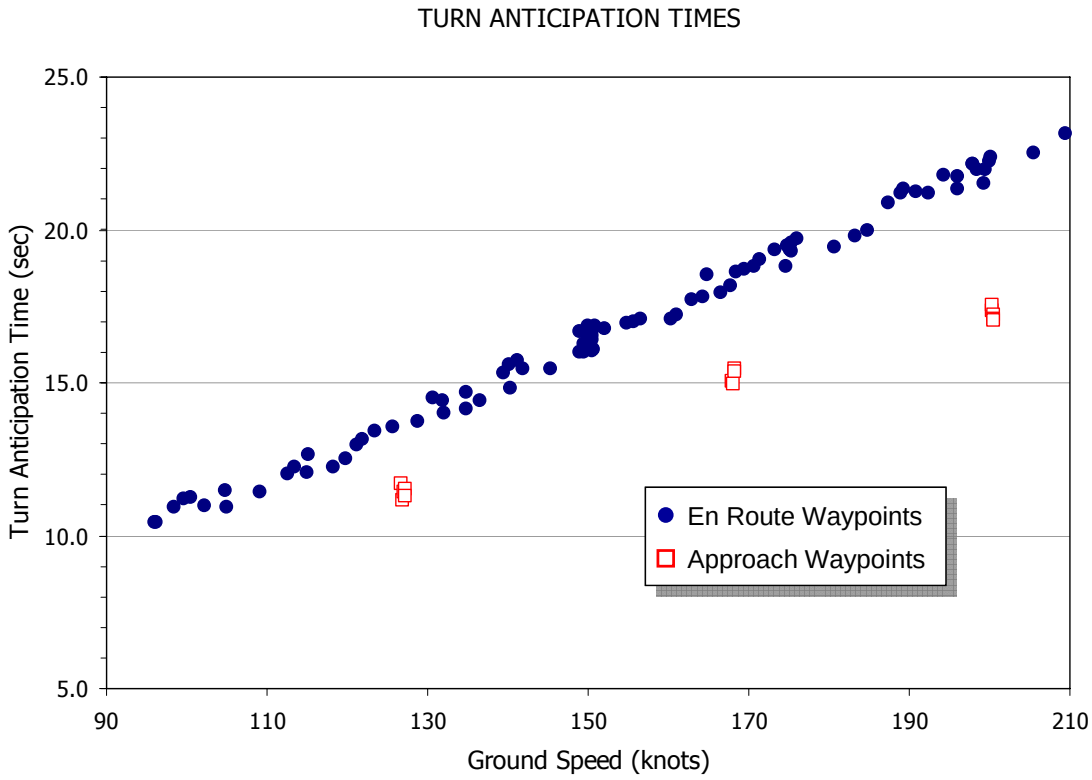
Other segments and some sub-segments are geographically fixed and do not vary by aircraft stall speed. KICT ILS19R examples include the Approach Transition, Intermediate Approach "2", Final Approach, and Missed Approach Straight Climb-out segments.

FLY-BY vs. FLY-OVER WAYPOINTS

Does fs9gps distinguish between Fly-By and Fly-Over Waypoints? Technically, no. If you stretch it … perhaps, but I think only to the extent that it distinguishes between En Route and Approach Waypoints.

Fly-Over Waypoints are usually associated with RNAV procedures, SIDs and STARs. I haven't studied enough fs9gps RNAV approaches to say if fs9gps treats RNAV Fly-over waypoints differently than RNAV Fly-By waypoints, but I suspect not.

However, I can say this much: the Turn Anticipation Time of Approach Waypoints, which include the typical Fly-Over Waypoints Missed Approach Point and Holding Pattern Fix, is less than that of En Route Waypoints, as shown in the chart below.

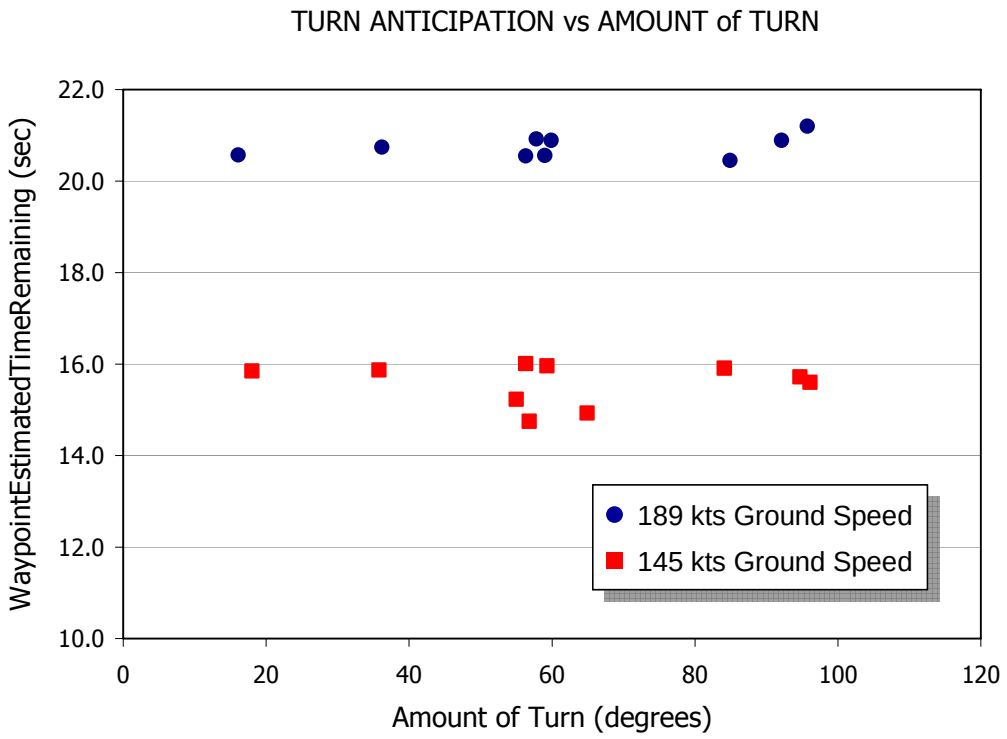


The smaller times result in more precise turns in the Approach phase when the aircraft should be traveling at approach rather than cruise airspeeds. So, closer tolerance on the Fly-Over Waypoints is part of the package.

TURN ANTICIPATION vs. AMOUNT OF TURN

Does the amount of turn (number of degrees turned) influence Turn Anticipation? No.

The chart below plots Turn Anticipation Time against Amount of Turn at two different Ground Speeds. The amount of turn has no impact on Turn Anticipation Time.

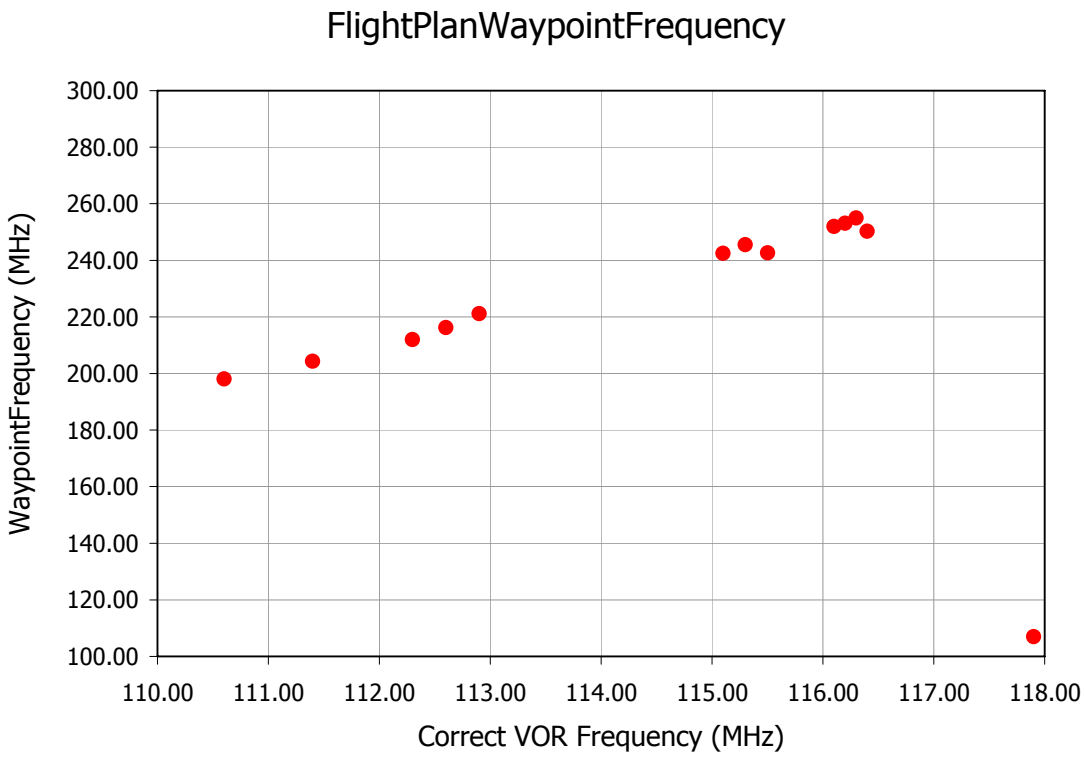


It's straight-forward for fs9gps to calculate remaining time to the next waypoint, but it might be a little more involved to adjust Turn Anticipation Time for the amount of turn ahead.

Most real-world Turn Anticipation Distance algorithms and rules of thumb are based on amount of turn, true airspeed, and bank angle. FS9's world is simpler.

FlightPlanWaypointFrequency DATAPOINTS

A few data points. A loose trend, un-predictable and non-sensical. As MSFT already stated, this variable is not implemented (not correctly anyway).



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]**

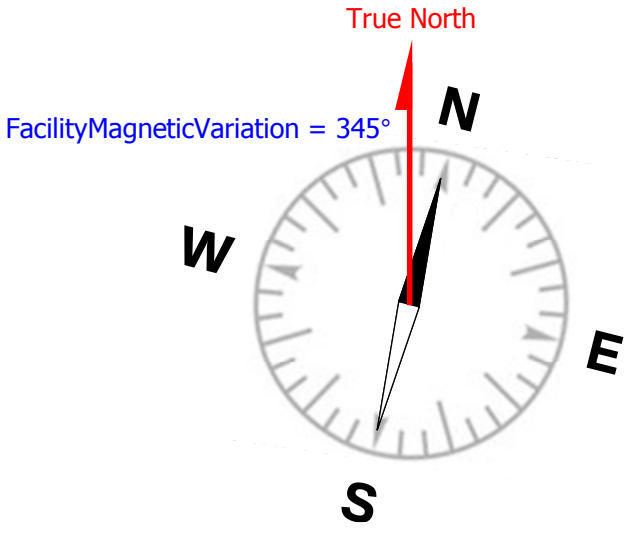
FacilityRegion returns the two letter Region Code. Region Codes don't exist for Airports.

❑ **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.

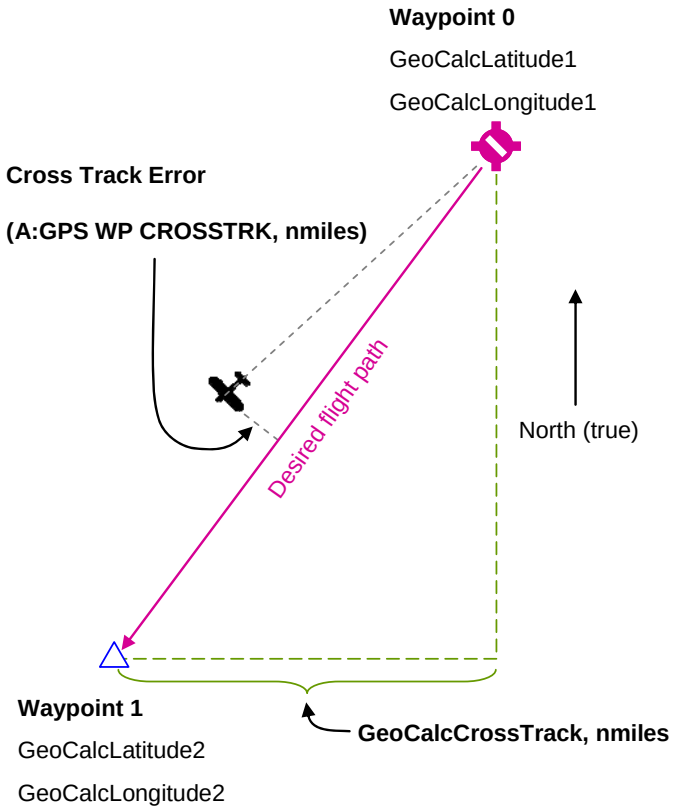
GeoCalcDistance is not the variable to be used for this.

❑ 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