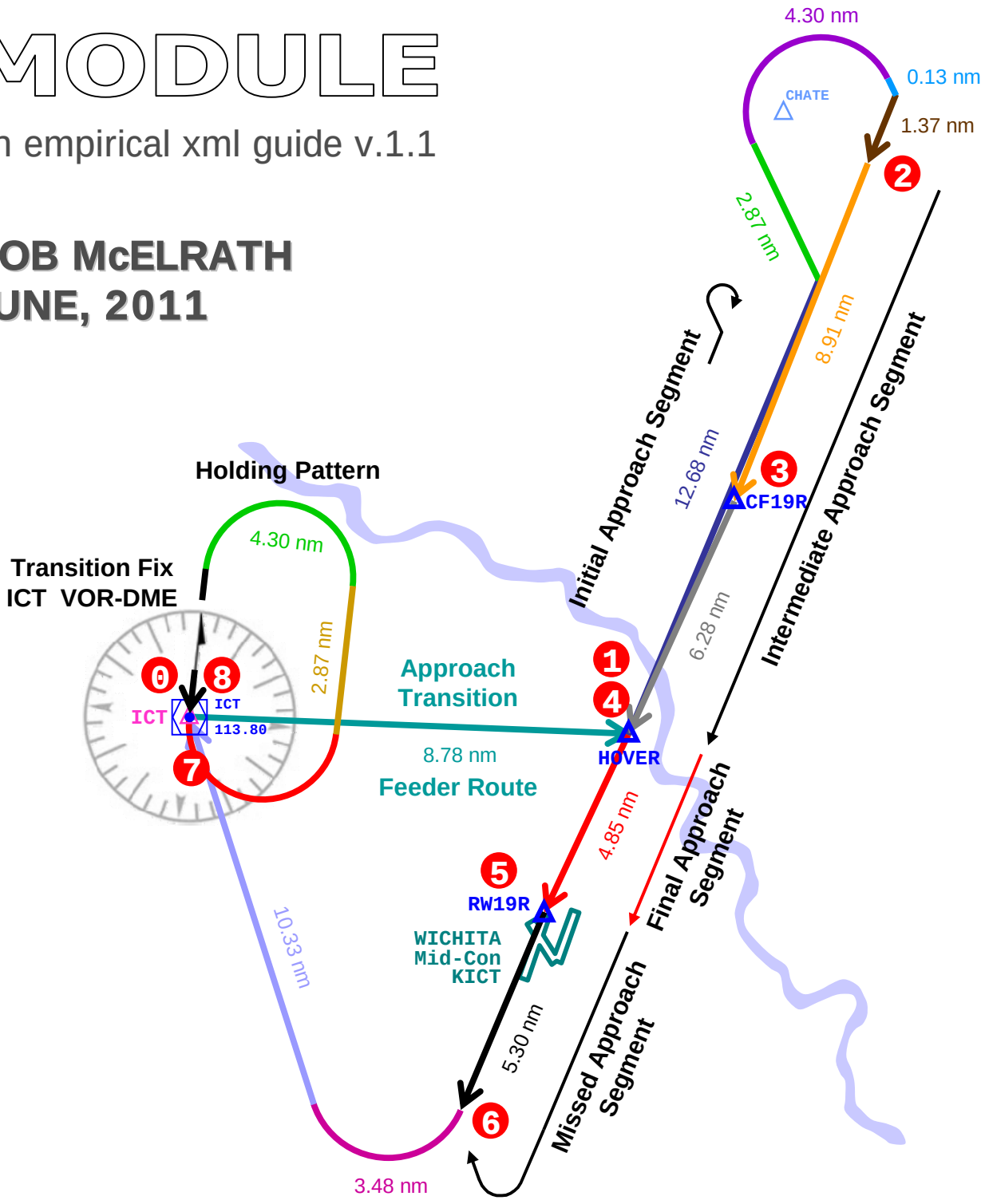
FS9GPS MODULE

an empirical xml guide v.1.1

BOB McELRATH

JUNE, 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).

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)
```

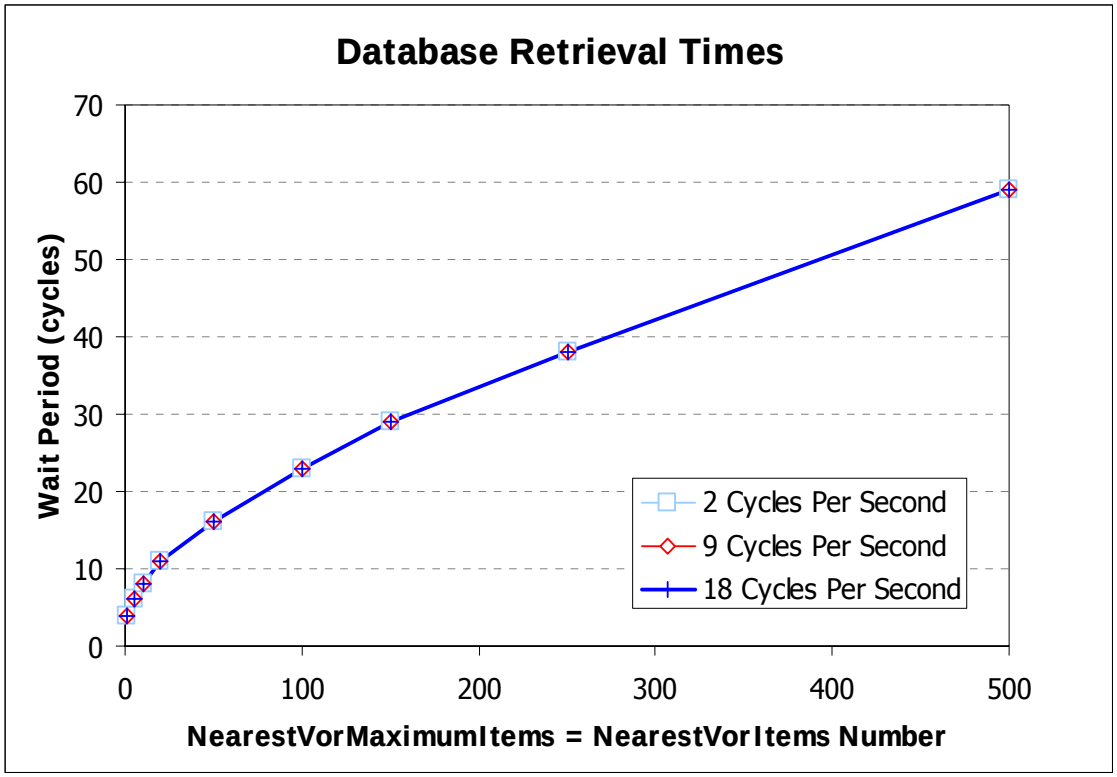
it is acceptable to omit the trailing spaces.

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

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.

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.

❑ 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.

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.

❑ 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.

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.

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>
```

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>
```

❑ 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).

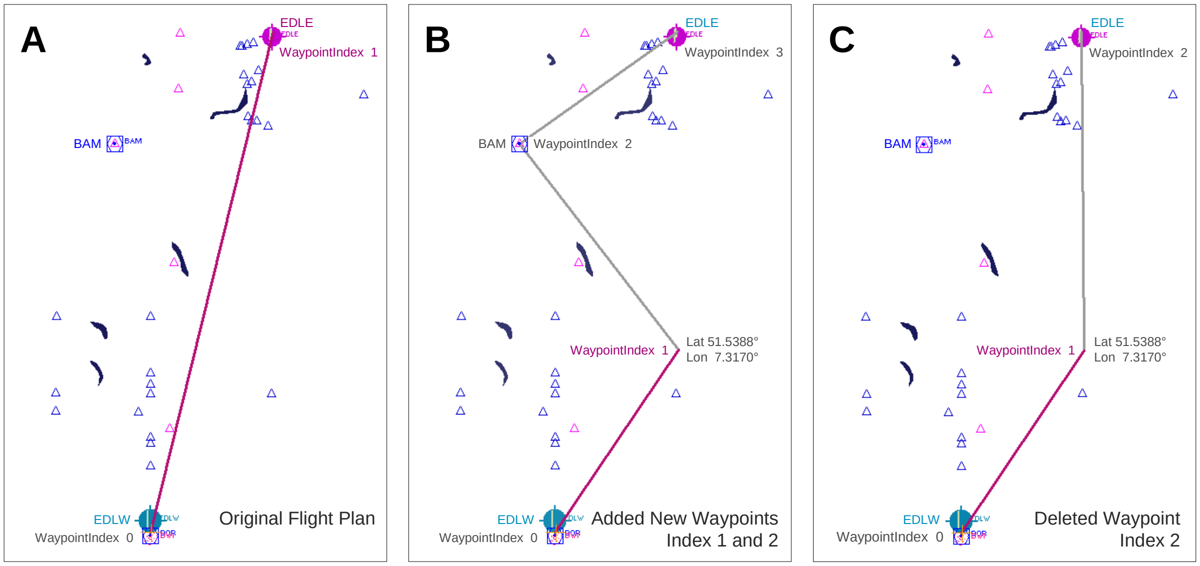
Restating this, the required information for a new en route waypoint is:

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.

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:

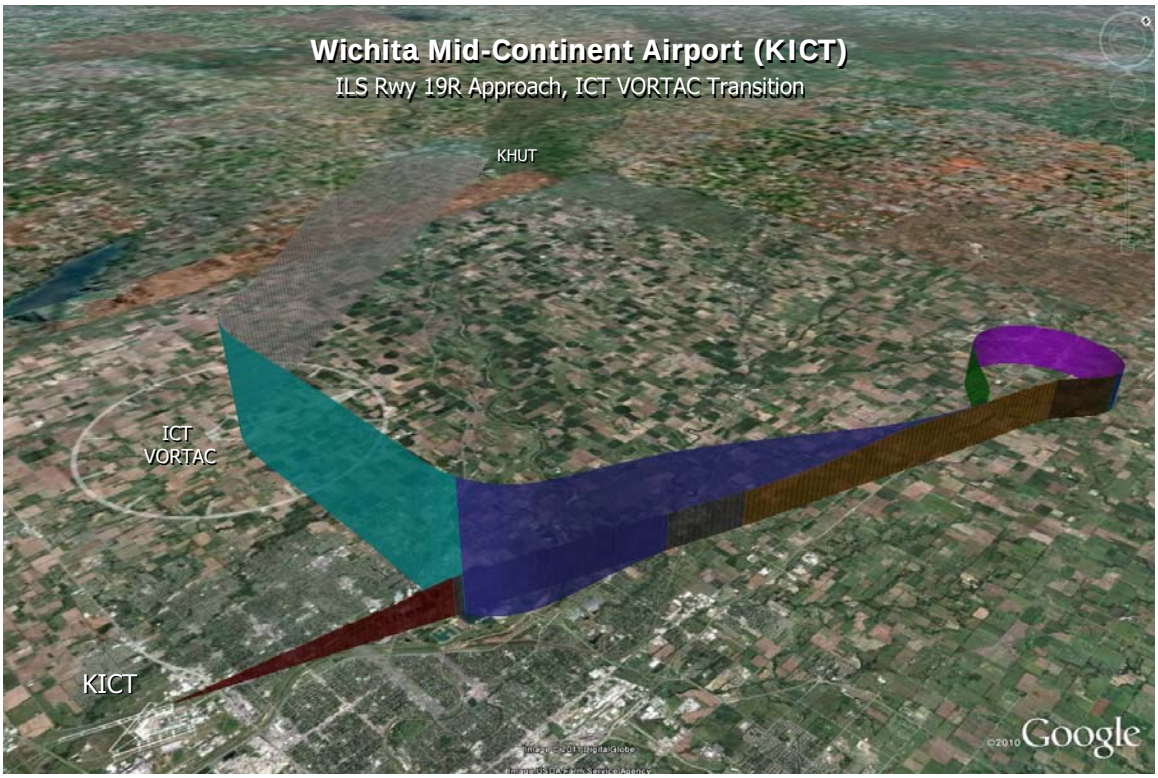
```
<!-- The first new Waypoint -->
   'VED    BAM' (>C:fs9gps:FlightPlanNewWaypointICAO)
   1 (>C:fs9gps:FlightPlanAddWaypoint)

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

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.



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.

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