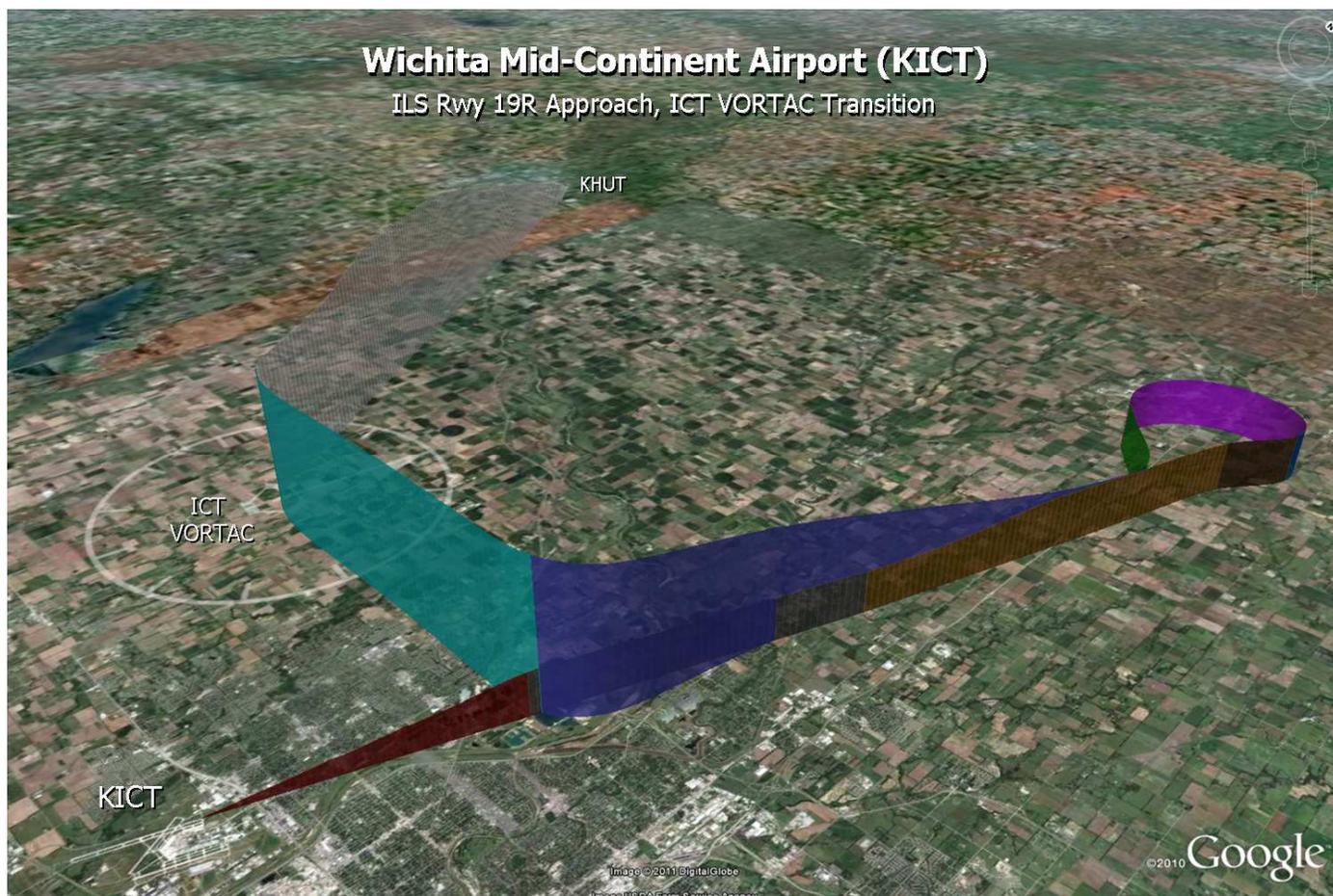


# FS9GPS MODULE GUIDEBOOK v.2.0.1

an empirical xml guide

Robert McElrath  
July, 2015





Page

1	<b>Introduction</b>
2	<b>Preliminaries</b>
3	<b>FS9 vs. FSX</b>
4	<b>Get, Set, and Units</b>
4	<b>Get and Set</b>
4	<b>Units</b>
5	<b>fs9gps Variable Groups</b>
6	<b>Flight Simulator Regions</b>
12	<b>ICAO</b>
12	<b>ICAO Examples</b>
13	<b>ICAO String Length</b>
14	<b>GPS Database Search: Search &gt; Index &gt; Display</b>
14	<b>Search and Extract</b>
16	<b>Index Pointer</b>
18	<b>Display of Extracted Data</b>
20	<b>Asynchronous Operation</b>
20	<b>When is Cycle Skipping Necessary?</b>
21	<b>Cycle Skip Example 1 – Nearest Searches</b>
22	<b>What happens when no cycle skipping code is used?</b>
24	<b>Cycle Skip Technique 1 - Cycle Counting</b>
26	<b>Cycle Skip Technique 2 - Let fs9gps tell you when it's ready</b>
28	<b>Cycle Skip Example 2 – ICAO Transfers</b>

- 28 **Cycle Skip Technique - Cycle Count** (“Let fs9gps tell you” not available)
- 31 **Cycle skipping not critical if data only displayed**
- 32 **ICAO Search – No Cycle-Skipping Required**
- 33 **Conditional Text Display**

## 35 **ICAO Search Example**

- 38 **Resolving Multiple Ident Matches**
- 40 **ICAO Search Airports - A Special Case**

## 41 **ICAO Transfer**

- 41 **ICAO Transfer Example – NearestAirport > WaypointAirport**

-  FS9
-  FSX

## 46 **Waypoint Airport Group**

- 46   WaypointAirportICAO
- 46   WaypointAirportIdent
- 46   WaypointAirportKind
- 46   WaypointAirportLongestRunwayDirection
- 46   WaypointAirportType
- 47   WaypointAirportName
- 47   WaypointAirportCity
- 47   WaypointAirportRegion
- 47   WaypointAirportLatitude
- 47   WaypointAirportLongitude
- 48   WaypointAirportElevation
- 48   WaypointAirportFuel1
- 48   WaypointAirportFuel2
- 48   WaypointAirportBestApproachEnum
- 48   WaypointAirportBestApproach

48			WaypointAirportRadarCoverage
48			WaypointAirportAirspace
49			WaypointAirportTowered
49			WaypointAirportCurrentFrequency
49			WaypointAirportFrequenciesNumber
49			WaypointAirportFrequencyName
49			WaypointAirportFrequencyLimit
50			WaypointAirportFrequencyValue
50			WaypointAirportFrequencyType
50			WaypointAirportCurrentRunway
50			WaypointAirportRunwaysNumber
50			WaypointAirportRunwayLatitude
50			WaypointAirportRunwayLongitude
51			WaypointAirportRunwayElevation
51			WaypointAirportRunwayDirection
52			WaypointAirportRunwayDesignation
52			WaypointAirportRunwayLength
52			WaypointAirportRunwayWidth
53			WaypointAirportRunwaySurface
53			WaypointAirportRunwayLighting
53			WaypointAirportCurrentApproach
53			WaypointAirportApproachesNumber
53			WaypointAirportApproachName
54			WaypointAirportApproachGps
54			WaypointAirportApproachTransitionsNumber
54			WaypointAirportApproachCurrentTransition
54			WaypointAirportApproachTransitionName
54			WaypointAirportApproachTransitionLatitude
54			WaypointAirportApproachTransitionLongitude
54			WaypointAirportApproachTransitionSize

55 **FSX-only Variables**

55 **Approach Variables** (not reliable)

- 55  WaypointAirportSelectedApproach
- 55  WaypointAirportApproachSelectedTransition
- 55  WaypointAirportApproachNumberLegs
- 55  WaypointAirportApproachCurrentLeg
- 56  WaypointAirportApproachCurrentLegIcao
- 56  WaypointAirportApproachCurrentLegType
- 56  WaypointAirportApproachCurrentLegBearing
- 56  WaypointAirportApproachCurrentLegDistance
- 56  WaypointAirportApproachCurrentLegIsMinutes

57 **Frequency Variables** (redundant with FS9)

- 57  WaypointAirportSelectedFrequencyIndex
- 57  WaypointAirportSelectedFrequencyValue

58 **Waypoint Intersection Group**

- 59   WaypointIntersectionIcao
- 59   WaypointIntersectionIdent
- 59   WaypointIntersectionType
- 59   WaypointIntersectionCity
- 59   WaypointIntersectionRegion
- 59   WaypointIntersectionLatitude
- 59   WaypointIntersectionLongitude
- 60   WaypointIntersectionNearestVorIdent
- 60   WaypointIntersectionNearestVorType
- 60   WaypointIntersectionNearestVorMagneticRadial
- 60   WaypointIntersectionNearestVorTrueRadial
- 60   WaypointIntersectionNearestVorDistance

61 **Waypoint NDB Group**

- 62   WaypointNdbICAO
- 62   WaypointNdbIdent
- 62   WaypointNdbType
- 63   WaypointNdbName
- 63   WaypointNdbCity
- 63   WaypointNdbRegion
- 63   WaypointNdbLatitude
- 63   WaypointNdbLongitude
- 63   WaypointNdbElevation
- 63   WaypointNdbFrequency
- 63   WaypointNdbWeatherBroadcast
- 64   WaypointNdbMagneticVariation

65 **FSX-only Variables**

- 65  WaypointNdbNearestAirportId
- 65  WaypointNdbNearestAirportLongestRunwayDirection
- 65  WaypointNdbNearestAirportKind
- 65  WaypointNdbNearestAirportBearing
- 65  WaypointNdbNearestAirportDistance

66 **Waypoint VOR Group**

- 67   WaypointVorICAO
- 67   WaypointVorIdent
- 67   WaypointVorType
- 68   WaypointVorClass
- 68   WaypointVorName
- 68   WaypointVorCity
- 68   WaypointVorRegion
- 68   WaypointVorLatitude
- 68   WaypointVorLongitude
- 68   WaypointVorElevation

- 69   WaypointVorFrequency
- 69   WaypointVorWeatherBroadcast
- 69   WaypointVorMagneticVariation
- 70 **FSX-only Variables**
- 70  WaypointVorNearestAirportId
- 70  WaypointVorNearestAirportLongestRunwayDirection
- 70  WaypointVorNearestAirportKind
- 70  WaypointVorNearestAirportBearing
- 70  WaypointVorNearestAirportDistance

71 **Nearest Airport Group**

- 71   NearestAirportCurrentLatitude
- 71   NearestAirportCurrentLongitude
- 71   NearestAirportMaximumItems
- 71   NearestAirportMaximumDistance
- 71   NearestAirportItemsNumber
- 71   NearestAirportCurrentLine
- 71   NearestAirportCurrentICAO
- 72   NearestAirportCurrentIdent
- 72   NearestAirportCurrentAirportKind
- 72   NearestAirportCurrentLongestAirportDirection
- 72   NearestAirportCurrentDistance
- 72   NearestAirportCurrentTrueBearing
- 72   NearestAirportCurrentBestApproachEnum
- 73   NearestAirportCurrentBestApproach
- 73   NearestAirportCurrentComFrequencyName
- 73   NearestAirportCurrentComFrequencyValue
- 74   NearestAirportCurrentLongestRunwayLength

74	<b>FSX-only Variables</b>
74	<b>Nearest Airport Data</b>
74	 NearestAirportSelected
74	 NearestAirportSelectedAirportLatitude
74	 NearestAirportSelectedAirportLongitude
74	 NearestAirportSelectedLatitude
74	 NearestAirportSelectedLongitude
74	 NearestAirportSelectedAirportName
74	 NearestAirportSelectedAirportCity
74	 NearestAirportSelectedAirportElevation
75	<b>Nearest Airport Frequency Data</b>
75	 NearestAirportCurrentFrequency
75	 NearestAirportSelectedFrequencyIndex
75	 NearestAirportSelectedNumberFrequencies
75	 NearestAirportCurrentFrequencyName
76	 NearestAirportSelectedFrequencyValue
76	<b>Nearest Airport Runway Data</b>
76	 NearestAirportSelectedRunway
76	 NearestAirportSelectedAirportRunwaysNumber
76	 NearestAirportSelectedRunwayDesignation
76	 NearestAirportSelectedRunwayLength
76	 NearestAirportSelectedRunwayWidth
76	 NearestAirportSelectedRunwaySurface
77	<b>Nearest Airport Approach Data</b>
77	 NearestAirportCurrentApproach
77	 NearestAirportSelectedApproachIndex
77	 NearestAirportSelectedNumberApproaches
77	 NearestAirportCurrentApproachName

## 78 **Nearest Intersection Group**

- 78   NearestIntersectionCurrentLatitude
- 78   NearestIntersectionCurrentLongitude
- 78   NearestIntersectionMaximumItems
- 78   NearestIntersectionMaximumDistance
- 78   NearestIntersectionCurrentFilter
- 79   NearestIntersectionAddIntersectionType
- 79   NearestIntersectionRemoveIntersectionType
- 80   NearestIntersectionSetDefaultFilter

80 **Examples:** NearestIntersection Add, Remove, and SetDefault

81 **Example 1:** NearestIntersectionAddIntersectionType

82 **Example 2:** NearestIntersectionRemoveIntersectionType

83 **Example 3:** NearestIntersectionSetDefaultFilter

- 84   NearestIntersectionItemsNumber
- 84   NearestIntersectionCurrentLine
- 84   NearestIntersectionCurrentICAO
- 84   NearestIntersectionCurrentIdent
- 84   NearestIntersectionCurrentType
- 84   NearestIntersectionCurrentDistance
- 84   NearestIntersectionCurrentTrueBearing

## 85 **FSX-only Variables**

- 85  NearestIntersectionSelectedIntersection
- 85  NearestIntersectionSelectedIntLatitude
- 85  NearestIntersectionSelectedIntLongitude
- 85  NearestIntersectionSelectedRefWorld
- 85  NearestIntersectionSelectedRefVorType
- 85  NearestIntersectionSelectedRefVorFrequency
- 85  NearestIntersectionSelectedRefVorTrueRadial
- 85  NearestIntersectionSelectedRefVorMagneticRadial
- 85  NearestIntersectionSelectedRefVorDistance

## 86 **Nearest VOR Group**

- 86   NearestVorCurrentLatitude
- 86   NearestVorCurrentLongitude
- 86   NearestVorMaximumItems
- 86   NearestVorMaximumDistance
- 86   NearestVorItemsNumber
- 86   NearestVorCurrentLine
- 86   NearestVorCurrentICAO
- 87   NearestVorCurrentIdent
- 87   NearestVorCurrentType
- 87   NearestVorCurrentFrequency
- 87   NearestVorCurrentDistance
- 87   NearestVorCurrentTrueBearing
- 87   NearestVorCurrentFilter
- 89   NearestVorAddVorType
- 90   NearestVorRemoveVorType
- 91   NearestVorSetDefaultFilter

## 92 **FSX-only Variables**

- 92  NearestVorSelectedVor
- 92  NearestVorSelectedVorLatitude
- 92  NearestVorSelectedVorLongitude
- 93  NearestVorSelectedVorName
- 93  NearestVorSelectedVorCity
- 93  NearestVorSelectedVorType
- 93  NearestVorSelectedVorFrequency
- 93  NearestVorSelectedVorMagneticVariation

## 94 **Nearest NDB Group**

- 94   NearestNdbCurrentLatitude
- 94   NearestNdbCurrentLongitude
- 94   NearestNdbMaximumItems

94			NearestNdbMaximumDistance
94			NearestNdbItemsNumber
94			NearestNdbCurrentLine
95			NearestNdbCurrentICAO
95			NearestNdbCurrentIdent
95			NearestNdbCurrentType
96			NearestNdbCurrentFrequency
96			NearestNdbCurrentDistance
96			NearestNdbCurrentTrueBearing
97			<b>FSX-only Variables</b>
97			NearestNdbSelectedNdb
97			NearestNdbSelectedNdbLatitude
97			NearestNdbSelectedNdbLongitude
97			NearestNdbSelectedNdbName
97			NearestNdbSelectedNdbCity
97			NearestNdbSelectedNdbType
97			NearestNdbSelectedNdbFrequency
98			<b>Nearest Airspace Group</b>
98			NearestAirspaceCurrentLatitude
98			NearestAirspaceCurrentLongitude
98			NearestAirspaceCurrentAltitude
99			NearestAirspaceTrueGroundTrack
99			NearestAirspaceGroundSpeed
99			NearestAirspaceNearDistance
99			NearestAirspaceNearAltitude
99			NearestAirspaceAheadTime
100			NearestAirspaceQuery
102			NearestAirspaceMaximumItems
102			NearestAirspaceMaximumDistance
102			NearestAirspaceItemsNumber

- 102   NearestAirspaceCurrentLine
- 103   NearestAirspaceCurrentName
- 103   NearestAirspaceCurrentType
- 103   NearestAirspaceCurrentFrequency
- 103   NearestAirspaceCurrentFrequencyName
- 104   NearestAirspaceCurrentMinAltitude
- 104   NearestAirspaceCurrentMaxAltitude
- 104   NearestAirspaceCurrentStatus
- 105 **Example Airspace Status and Messages**
- 109   NearestAirspaceCurrentNearDistance
- 109   NearestAirspaceCurrentAheadTime

## 110 **Message Group**

- 110   MessageItemsNumber
- 110   MessageCurrentLine
- 110   MessageCurrentType
- 110   NewMessagesNumber
- 110   NewMessagesConfirm

## 111 **ICAO Search Group**

- 111   IcaoSearchInitialIcao
- 111   IcaoSearchStartCursor
- 112   IcaoSearchStopCursor

### 112 **Data Entry Methods for ICAOSearch**

#### 112 **Keyboard Direct Entry**

#### 112 **Mouse Entry**

#### 114 **XML Script (code) Entry**

- 114   IcaoSearchAdvanceCursor
- 114   IcaoSearchAdvanceCharacter
- 115   IcaoSearchEnterChar
- 116   IcaoSearchBackupChar

- 118   IcaoSearchCursorPosition
- 118   IcaoSearchCurrentIdent
- 118   IcaoSearchCurrentIcao
- 118   IcaoSearchCurrentIcaoType
- 118   IcaoSearchCurrentIcaoRegion
- 119   IcaoSearchMatchedIcaosNumber
- 119   IcaoSearchMatchedIcao

120 **Name Search Group**

- 120   NameSearchInitialIcao
- 120   NameSearchInitialName
- 120   NameSearchStartCursor
- 120   NameSearchStopCursor
- 121   NameSearchAdvanceCursor
- 121   NameSearchAdvanceCharacter
- 122   NameSearchEnterChar

122 **Data Entry Methods for NameSearch**

122 **Keyboard Direct Entry**

122 **Mouse Entry**

123 **XML Script (code) Entry**

- 124   NameSearchBackupChar
- 125   NameSearchCursorPosition
- 125   NameSearchCurrentName
- 126   NameSearchCurrentMatch
- 126   NameSearchCurrentIcao
- 126   NameSearchCurrentIcaoType
- 126   NameSearchCurrentIcaoRegion

127		<b>Flight Plan Group</b>
127		<b>Flight Planning</b>
127		<b>Components of the Flight Plan</b>
127	 	FlightPlanTitle
127	 	FlightPlanDescription
127	 	FlightPlanFlightPlanType
128	 	FlightPlanRouteType
128	 	FlightPlanCruisingAltitude
128	 	FlightPlanDepartureLatitude
128	 	FlightPlanDepartureLongitude
129	 	FlightPlanDepartureAirportIdent
129	 	FlightPlanDepartureName
129	 	FlightPlanDepartureAltitude
129	 	FlightPlanDestinationLatitude
129	 	FlightPlanDestinationLongitude
130	 	FlightPlanDestinationAirportIdent
130	 	FlightPlanDestinationName
130	 	FlightPlanDestinationAltitude
130		<b>Comparison to Flight Plan .pln File</b>
131	 	FlightPlanAlternateAirportIdent
131	 	FlightPlanAlternateLatitude
131	 	FlightPlanAlternateLongitude
131	 	FlightPlanAlternateAltitude
131	 	FlightPlanAlternateName
132		<b>FlightPlanCruisingAltitude Discussion</b>
134	 	FlightPlanIsActiveFlightPlan
134	 	FlightPlanIsLoadedApproach
134	 	FlightPlanIsActiveApproach
134	 	FlightPlanIsActiveWaypoint
134	 	FlightPlanIsDirectTo
134	 	FlightPlanDirectToWaypoint

- 135   FlightPlanActiveWaypoint
- 135   FlightPlanActiveApproachWaypoint
- 135   FlightPlanIsActiveWaypointLocked
- 136   FlightPlanWaypointsNumber
- 136 **NewWaypoint Group: Creating and Editing a Flight Plan**
- 136 **Creating a Flight Plan With XML**
- 136 **Editing a Flight Plan**
- 137 **Entering New Waypoint Latitude and Longitude**
- 137   FlightPlanNewWaypointLatitude
- 137   FlightPlanNewWaypointLongitude
- 137   FlightPlanNewWaypointICAO
- 138   FlightPlanNewWaypointIdent
- 139 **Assigning a Waypoint Index and Adding the New Waypoint**
- 139   FlightPlanAddWaypoint
- 139   FlightPlanDeleteWaypoint
- 140 **Example 1:** FlightPlanAddWaypoint and FlightPlanDeleteWaypoint
- 141   FlightPlanDirectToDestination
- 143   FlightPlanCancelDirectTo
- 144 **Example 2:** FlightPlanDirectToDestination and CancelDirectTo
- 146 **Example 3:** ActiveWaypointLocked, AddWaypoint and ActiveWaypoint
- 149 **Example 4:** Changing the Active Waypoint
- 150 **NewApproach Group: Adding or Changing an Approach**
- 150   FlightPlanNewApproachAirport
- 150   FlightPlanNewApproachApproach
- 150   FlightPlanNewApproachTransition
- 151   FlightPlanNewApproachMissed
- 151   FlightPlanNewApproachAddInitialLeg
- 152   FlightPlanLoadApproach

- 153       **Example 5:** Adding or Changing an Approach
- 154       **Example 5.1** NewApproachAddInitialLeg = 0, LoadApproach = 1
- 156       **Example 5.2** NewApproachAddInitialLeg = 1, LoadApproach = 1
- 157       **Example 5.3** NewApproachAddInitialLeg = 2, LoadApproach = 1
- 158       **Example 5.4** NewApproachAddInitialLeg = 3, LoadApproach = 1
- 159       **Example 5.5** NewApproachAddInitialLeg = 0, LoadApproach = 2
- 160       **Example 5.6** NewApproachAddInitialLeg = 1, LoadApproach = 2
- 161       **Example 5.7** NewApproachAddInitialLeg = 2, LoadApproach = 2
- 162       **Example 5.8** NewApproachAddInitialLeg = 3, LoadApproach = 2
- 163       **Example 5.9** NewApproachAddInitialLeg = 0, LoadApproach = 3
- 165       **Example 5.10** NewApproachAddInitialLeg = 1, LoadApproach = 3

166       **En Route Navigation**

- 166   ✕ ✕ FlightPlanWaypointIndex
- 166   ✕ ✕ FlightPlanWaypointLatitude
- 166   ✕ ✕ FlightPlanWaypointLongitude
- 166   ✕ ✕ FlightPlanWaypointAltitude
- 166   ✕ ✕ FlightPlanWaypointICAO
- 166   ✕ ✕ FlightPlanWaypointIdent
- 167   ✕ ✕ FlightPlanWaypointAirwayIdent
- 167   ✕ ✕ FlightPlanWaypointType
- 167   ✕ ✕ FlightPlanWaypointMinAltitude
- 169   ⊘ ✕ FlightPlanWaypointFrequency
- 169   ✕ ✕ FlightPlanWaypointMagneticHeading
- 169   ✕ ✕ FlightPlanWaypointSpeedEstimate
- 169   ✕ ✕ FlightPlanWaypointDistance
- 169   ✕ ✕ FlightPlanWaypointDistanceTotal
- 170   ✕ ✕ FlightPlanWaypointDistanceRemaining
- 170   ✕ ✕ FlightPlanWaypointRemainingDistance
- 171   ✕ ✕ FlightPlanWaypointRemainingTotalDistance

171		<b>Turn Anticipation</b>
174	 	FlightPlanWaypointTimeZoneDeviation
174	 	FlightPlanWaypointETE
174	 	FlightPlanWaypointATE
174	 	FlightPlanWaypointEstimatedTimeRemaining
174	 	FlightPlanWaypointETA
175	 	FlightPlanWaypointFuelRemainedAtArrival
175	 	FlightPlanWaypointEstimatedFuelConsumption
175	 	FlightPlanWaypointActualFuelConsumption
176		<b>Instrument Approaches</b>
179	 	FlightPlanApproachWaypointType
179	 	FlightPlanApproachMode
179	 	FlightPlanApproachSegmentType
179		<b>Approach Segments and Sub-Segments</b>
180	 	FlightPlanApproachSegmentDistance
180	 	FlightPlanApproachSegmentLength
180	 	FlightPlanApproachIsWaypointRunway
180	 	FlightPlanApproachAirportIdent
181	 	FlightPlanApproachType
181	 	FlightPlanApproachIndex
181	 	FlightPlanApproachName
181	 	FlightPlanApproachTransitionIndex
182	 	FlightPlanApproachTransitionName
182	 	FlightPlanIsApproachFinal
182	 	FlightPlanIsApproachMissed
182	 	FlightPlanApproachWaypointsNumber
183	 	FlightPlanWaypointApproachIndex
183	 	FlightPlanWaypointApproachICAO
183	 	FlightPlanWaypointApproachName
183	 	FlightPlanWaypointApproachType
183	 	FlightPlanWaypointApproachMode

- 184   FlightPlanWaypointApproachLatitude
- 184   FlightPlanWaypointApproachLongitude
- 184   FlightPlanWaypointApproachAltitude
- 184   FlightPlanWaypointApproachCourse
- 184   FlightPlanWaypointApproachTarget
- 186   FlightPlanWaypointApproachLegDistance
- 187   FlightPlanWaypointApproachLegTotalDistance
- 187   FlightPlanWaypointApproachLegFromDistance
- 187   FlightPlanWaypointApproachRemainingDistance
- 187   FlightPlanWaypointApproachRemainingTotalDistance

188 **Miscellaneous**

188 **Dissecting the KICT ILS19R Approach Segments**

192 **Sub-Segment Length**

193 **Fly-By vs. Fly-Over Waypoints**

194 **Turn Anticipation vs. Amount of Turn**

195 **Facility Group**

- 196   FacilityICAO
- 196   FacilityCode
- 196   FacilityIdent
- 196   FacilityValid
- 197   FacilityName
- 197   FacilityCity
- 197   FacilityRegion
- 197   FacilityLatitude
- 197   FacilityLongitude
- 197   FacilityMagneticVariation

## 198 **GeoCalc Group**

198			GeoCalcLatitude1
198			GeoCalcLongitude1
198			GeoCalcLatitude2
198			GeoCalcLongitude2
198			GeoCalcAzimuth1
198			GeoCalcAzimuth2
199			GeoCalcLength
199			GeoCalcBearing
199			GeoCalcDistance
200			GeoCalcIsIntersect
200			GeoCalcIntersectionLatitude
200			GeoCalcIntersectionLongitude
200			GeoCalcExtrapolationLatitude
200			GeoCalcExtrapolationLongitude
201			GeoCalcCrossTrack

## 202 **Data Entry and Working with Strings**

### 203 **String Operators**

### 204 **ASCII Code**

### 204 **String Entry Methods**

204 Keyboard Direct Entry

206 Mouse Click Entry Using a Keypad Image

### 208 **Concatenation and String Storage**

208 5 characters maximum can be stored in a single L:Var

208 Shift Register

210 XMLVars – Custom L:Vars (Tom Aguilo)

213 String Storage Macros (Robbie McElrath)

213 The macros (12 in all)

214 Using the String Storage Macros

216	<b>Other</b>
216	Storage in internal registers
216	LOGGER XML > HDD > XML (Robbie McElrath)
216	String Storage v2.0.1 (Doug Dawson)
217	<b>&lt;ELEMENT&gt; Display Loops</b>
219	<b>Panel Reload Gauge</b>
220	<b>Bugs, Inops, and Issues</b>
221	<b>fs9gps Guidebook Updates</b>



# Introduction

This is an empirical guide for working with the FS9 and FSX gps modules. It's a collection of notes on utilization of the fs9gps module, definition and examples of its 386 variables (321 FS9 & FSX plus 65 that are FSX-only), and discussion of some xml coding techniques that are often used when working with the gps module.

**What's new in v.2.0?** Descriptions of the 65 FSX-only gps variables, an expanded section on Working with Strings and an example of a Reload Panel-Aircraft gauge which is an XML must-have utility have been added. In addition, many graphics are updated and, importantly, errors found in v.1.1 have been corrected (for example, Runway direction variables return True, not Magnetic).

What this Guidebook is not - It isn't a discussion of the stock gps\_500 xml gauge. None of the 80+ custom @g functions specific to that gauge are discussed. None of those are necessary to use the full capabilities of the gps module anyway. Nor is this a pilot's handbook on the operation of a gps unit. 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 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 Gauges community has some professional and experienced programmers that have always been willing to offer advice. Bill Learning heads that list. Still, a significant number of things having to do with fs9gps are not addressed by MSFT or anywhere in the forums, and that which is out there is scattered around and vanishing sometimes, so I attempted to fill in the 'white space' and pull it together into one document.

I relied a lot on some important tools such as BlackBox3, LOGGER, XMLVars, and many fit for purpose xml scripts to study the fs9gps ... test, test, test, and re-test. I mention the tools to acknowledge the contributions of Tom Aguilo (XMLVars) and my son, Robbie, the author of BlackBox and LOGGER and the professional programmer of the family.

Bob McElrath  
Bangkok, Thailand  
July, 2015 (Version 2.0.1)



© 2015 Robert McElrath

## Preliminaries

Some odds and ends:

- ❑ FS9 Version 9.01 Build 40901.01 was used in development of this guidebook. For evaluation of the FSX-only variables, FSX Deluxe Build 10.0.61637.0 (FSX-Xpack.20070926-1421).
- ❑ 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 Courier New font, (`A:PLANE LATITUDE, degrees`).
- ❑ Very often, I abbreviate gps variable names by omitting the Group Name. For example, in the Flight Plan Data Group, I refer to [FlightPlanWaypointApproachTarget](#) simply as [WaypointApproachTarget](#).
- ❑ FS9 xml syntax (FS9 XML schema) is used throughout, not FSX.
- ❑ @C macro is frequently used. `<Macro Name="C">C:fs9gps</Macro>`
- ❑ 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: BlackBox3, 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).

## FS9 vs. FSX

The FSX gps module is mostly the same as the FS9 version. FSX uses the same variable groups, same Regions, and has all of the FS9 variables. Both the FS9 and FSX versions file name is gps.dll. The FS9 version is typically found in the Modules folder in FS9, and in the root FSX folder in a typical FSX setup.

Methods such as ICAO Transfer and Cycle Skipping apply to both versions of the gps module.

However, some differences noted include:

- ❑ FSX adds 65 new variables. Some do not function, or function correctly, however.
- ❑ The FSX database is slightly different. Coordinates of some facilities are changed and new facilities (for example, new waypoints) have been added.
- ❑ The function of some variables is changed / fixed. Examples include:
  - FS9 locks the active waypoint after [AddWaypoint](#) is used. This does not occur in FSX (this is an example of a FS9 bug that was fixed).
  - [FlightPlanWaypointFrequency](#) returns incorrect data in FS9 but this is fixed in FSX.
- ❑ I suspect there are several examples of other, similar changes, but I have not gone through all of the variables in the FSX module to check.

As far as I know, the FSX gps module is used in at least the early versions of P3D, so I guess, but I don't know, that comments in this Guidebook apply to some versions of the P3D simulation as well. So far, I have not tested the gps module in P3D (any version). The reader should do their own testing to confirm.

# 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 L:LoadApprEnumEntered because [FlightPlanLoadApproach](#) cannot be displayed.

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

## UNITS

Like other FS variables (e.g., A:, P:, E:), gps variables presume units. Throughout this guidebook, however, I omit specifying the units for string, enum and bool variables, but include suggested units for distance, direction, time, speed, and frequency-type gps variables. The suggested units are not necessarily the default FS units (for example, the suggested distance unit is nmiles, but FS default distance unit is meters).

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 321 gps variables in the fs9gps FS9 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:
    - NearestAirport Group
    - NearestIntersection Group
    - NearestVor Group
    - NearestNdb Group
    - 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.



***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***

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  
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

***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***

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)  
 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)  
 VL Laos  
 VM Macau  
 VN Nepal  
 VO India (South Zone, Chennai Center)  
 VQ Bhutan  
 VR Maldives  
 VT Thailand  
 VV Vietnam  
 VY Myanmar

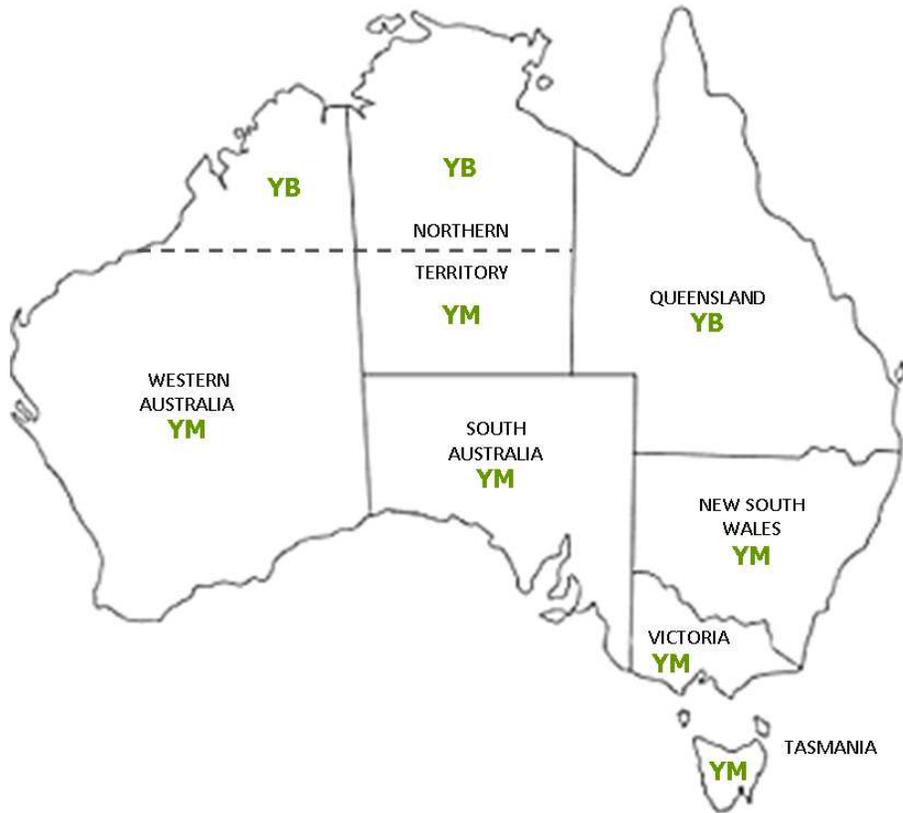
**W Maritime Southeast Asia (except the Philippines)**  
 WA Indonesia (also WI, WQ and WR)  
 WB Malaysia (East Malaysia), Brunei  
 WI Indonesia (also WA, WQ and WR)  
 WM Malaysia (Peninsular Malaysia)  
 WP Timor-Leste  
 WQ Indonesia (also WA, WI and WR)  
 WR Indonesia (also WA, WI and WQ)  
 WS Singapore

**Y Australia**  
 Y Australia (multiple)

**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 Australia



# ICAO

As defined by the GPS module

In the gps module, ICAO is defined as the 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.

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

## ICAO Examples

FACILITY TYPE	SLEN	Character Position											
		1	2	3	4	5	6	7	8	9	10	11	12
Airport	12	A							E	G	N	X	
Airport	12	A							W	3	6		
ILS	12	V			K	A	S	T	I	A	S	T	
VOR	12	V	E	G					M	I	D		
Intersection	12	W	K	3	K	8	8		A	B	U	Y	A
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			
Runway	12	R	K	3	K	I	C	T	R	W	1	9	R
		Type	Region		Owning Airport Ident				Facility Ident				

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

□ **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. Ident ≠ ICAO** Character positions 8 through 12. The one to five character long Ident of the facility. Rather than the full ICAO, Ident is the more common facility abbreviation, like KLAX for Los Angeles International Airport. However, Ident is also sometimes confused with the full 12 character "ICAO" even in Microsoft FS documentation (for example, Microsoft ESP; Panels and Gauges SDK; XML Gauge Maps; TextDetailLayerAirports).

Note that while the full 12 character fs9gps ICAOs are unique, Idents are not necessarily unique other than airport Idents; there are many occurrences in the database of VORs having the same 3 letter Ident. When working with the gps module, it is sometimes useful to remember that all *airport* Idents in the fs9gps database are unique, but Idents for other facilities are not necessarily unique.

## ICAO String Length

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 of the Ident if five characters are not used. In other words,

'A\_\_\_W36\_\_' is not necessary.

## GPS Database Search: Search > Index > Display

Most information needed from the fs9gps database must be obtained through a sequence of 1) Search (Extract), 2) Index, and then 3) Display. It is important to understand that the gps database search step can be computationally intensive, requiring time to complete. Usually, several gauge update cycle pass before the gps module finishes data retrieval and to accommodate this, the gps module must operate asynchronously with the other gauges of the panel set. Consequently, data cannot be retrieved from the database and then displayed or otherwise utilized in the same gauge update cycle that the database search was initiated because of the time required by the gps module to complete the search.

### 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.

A database Search is defined by specifying 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)

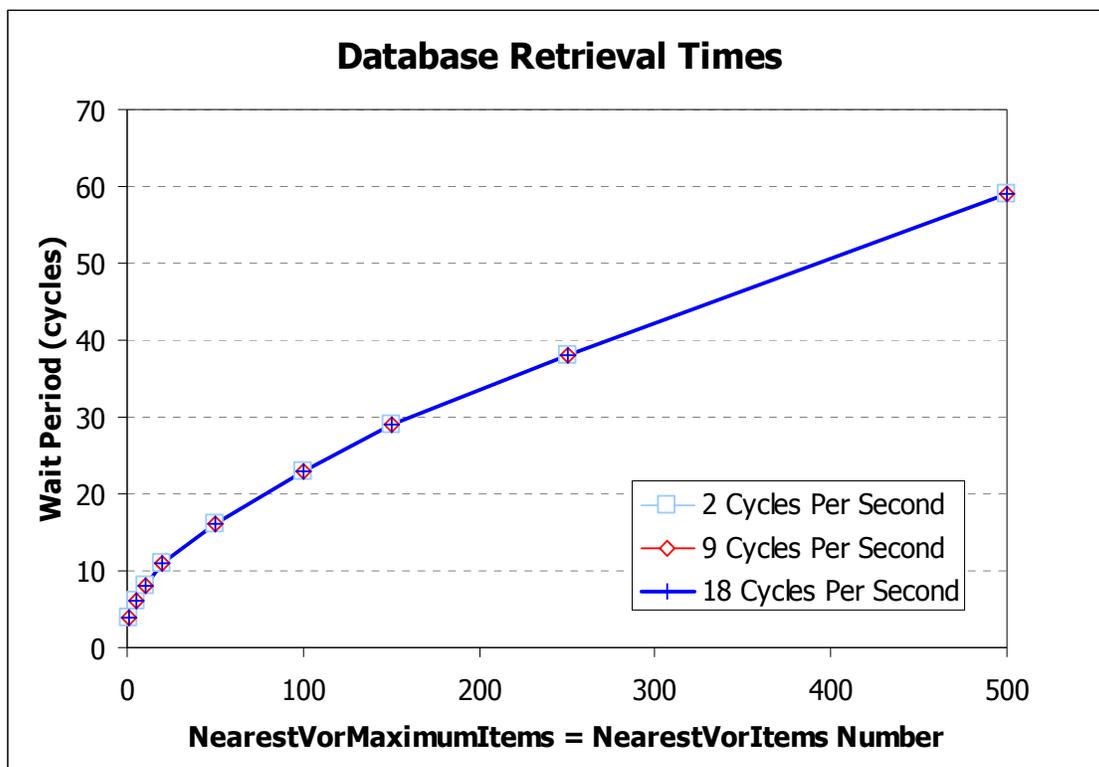
20 (>c:fs9gps:NearestVorMaximumItems, enum)
100 (>c:fs9gps:NearestVorMaximumDistance, nmiles)
```

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 low (e.g., lines 416-419, max items = 9, matching the real Garmin GNS 500 series GPS Units) 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 one 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.



**Multiple update cycle** database operations include (there may be more):

- 1) **Nearest** searches
- 2) **ICAO transfer**
- 3) 0 (>@c:FlightPlanIsActiveWaypointLocked) following a (>@c:FlightPlanAddWaypoint) (FS9 only)
- 4) **ICAO transfers** necessary for use of FSX **WaypointVor & NbdNearestAirport** variables

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 POINTER

Much of the information retrieved from the `gps` database 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 start at 0 for the first line).

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

### **Index Pointer**

WaypointAirportCurrentFrequency  
 WaypointAirportCurrentRunway  
 WaypointAirportCurrentApproach  
 WaypointAirportApproachCurrentTransition  
 NearestAirportCurrentLine  
 NearestIntersectionCurrentLine  
 NearestVorCurrentLine  
 NearestNdbCurrentLine  
 NearestAirspaceCurrentLine  
 FlightPlanWaypointIndex  
 IcaoSearchMatchedIcao  
 MessageCurrentLine  
 FlightPlanWaypointApproachIndex  
 ITrafficInfo:CurrentVehicle

### **Total Number**

WaypointAirportFrequenciesNumber  
 WaypointAirportRunwaysNumber  
 WaypointAirportApproachesNumber  
 WaypointAirportApproachTransitionsNumber  
 NearestAirportItemsNumber  
 NearestIntersectionItemsNumber  
 NearestVorItemsNumber  
 NearestNdbItemsNumber  
 NearestAirspaceItemsNumber  
 FlightPlanWaypointsNumber  
 IcaoSearchMatchedIcaosNumber  
 MessageItemsNumber  
 FlightPlanApproachWaypointsNumber  
 ITrafficInfo:ListSize

**Index Pointer** (FSX-Only Variables)

WaypointAirportApproachCurrentLeg  
 NearestAirportSelected  
 NearestAirportCurrentFrequency  
 NearestAirportSelectedFrequencyIndex  
 NearestAirportSelectedRunway  
 NearestAirportCurrentApproach  
 NearestAirportSelectedApproachIndex  
 NearestNdbSelectedNdb  
 NearestVorSelectedVor  
 NearestIntersectionSelectedIntersection  
 WaypointAirportSelectedFrequencyIndex

**Total Number** (FSX-Only Variables)

WaypointAirportApproachNumberLegs  
 NearestAirportItemsNumber  
 NearestAirportSelectedNumberFrequencies  
 NearestAirportSelectedNumberFrequencies  
 NearestAirportSelectedAirportRunwaysNumber  
 NearestAirportSelectedNumberApproaches  
 NONE  
 NearestNdbItemsNumber  
 NearestVorItemsNumber  
 NearestIntersectionItemsNumber  
 NONE

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  
 -1.6552 :Current Lon 100 :Max Dist 62 :Filter

----- NearestVorCurrent -----

Line	ICAO	Ident	Type	Freq	Dist	Brg
0	VEG FFA	FFA	3	113.40	12.4	335
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 BSO	BSO	3	110.00	22.0	71
6	VEG OX	OX	3	117.70	23.8	31
7	VEG GOS	GOS	3	115.55	30.6	322
8	VEG ODH	ODH	3	109.60	30.8	120
9	VEG BLC	BLC	3	116.20	32.0	108

## 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="300" Font="Courier New" FontSize="12"
    LineSpacing="12" BackgroundColor="white" 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.**"

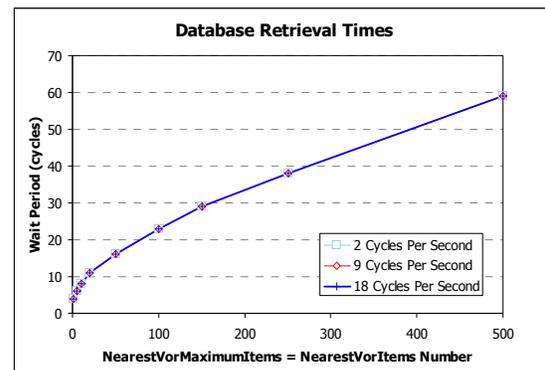
# Asynchronous Operation

## 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?

The question is not whether a particular fs9gps operation is multi-cycle, that is, whether it requires multiple gauge update cycles to complete. The important concern is whether or not execution of *subsequent* code that uses search data results must be carefully timed to not begin until search data are retrieved from the database.

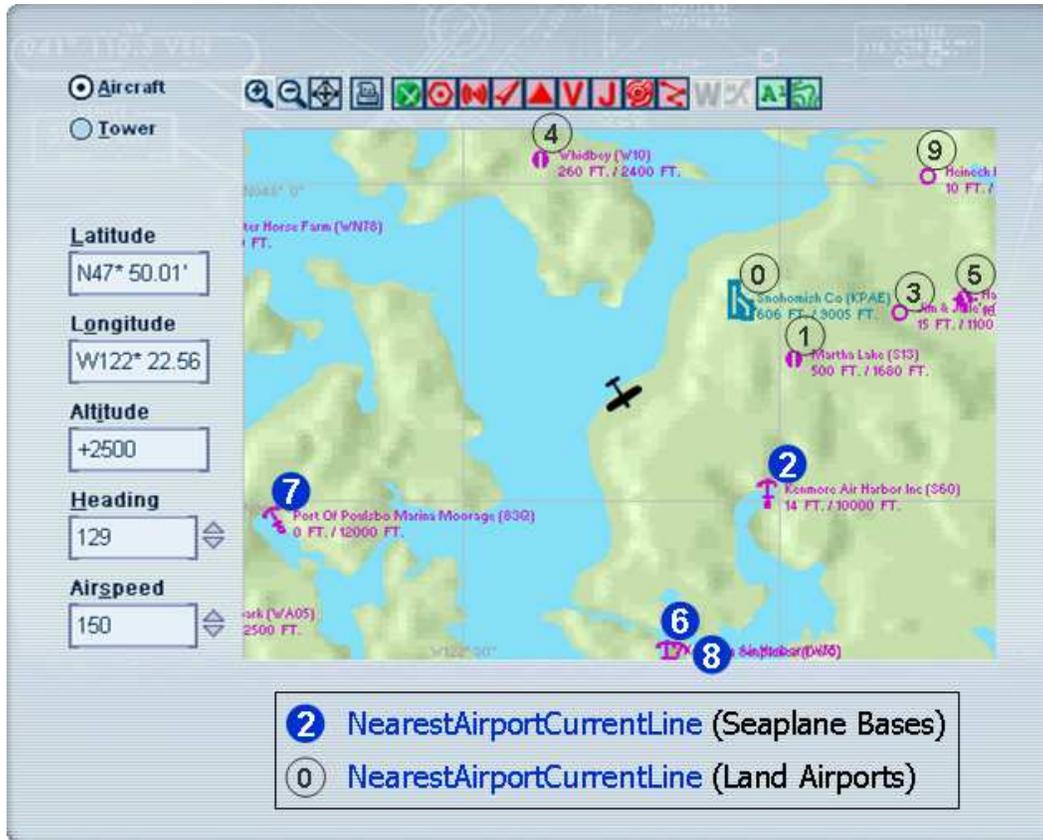
Consider a situation where one may want to find the nearest seaplane base. There are a couple of ways to approach this depending upon what you want to do afterward. For the purposes of this section, one way is to perform a [NearestAirport](#) search, then in <Update>, loop through the [NearestAirport](#) 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.

## Cycle Skip 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 Inc. Seaplane Base (S60).



### NEAREST AIRPORT SEARCH

47.8355 :Current Lat 100 :Max Items  
 -122.3760 :Current Lon 100 :Max Distance

Image truncated after 12th Airport  
 (FS9 database)

Kind = 3 = SEAPLANE BASE

Line	ICAO	Ident	Kind	Rwy°	Dist	Brg°	Best	Appr	Com	Freq	Length
0	A KPAE	KPAE	1	179	5.7	42	13	ILS	twr	120.20	9005
1	A S13	S13	1	180	5.8	74	0			0.00	1680
2	A S60	S60	3	180	6.8	136	0		CTF	122.70	10000
3	A 96WA	96WA	2	180	9.6	67	0			0.00	1100
4	A W10	W10	1	180	11.2	347	0		CTF	122.90	2400
5	A S43	S43	1	160	11.7	68	0		CTF	123.00	2660
6	A W55	W55	3	180	12.5	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	12.6	172	0		CTF	122.90	9500
9	A 76WA	76WA	2	91	14.1	44	0			0.00	2500
10	A W16	W16	1	92	15.5	82	0		CTF	122.90	2095
11	A WN78	WN78	2	21	17.2	296	0			0.00	2450

## 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:

```
1 <Update Frequency="18" Hidden="No">
2 <!-- NearestAirport Search RESET -->
3 (L:NrstAptSearchReset, bool) 1 ==
4   if{
5     -1 (>L:IndexPointer, enum)
6     1 (>L:LoopingThroughNearestAirportList, bool)
7     0 (>L:NrstSeaBaseDist, nmiles)
8     0 (>L:NrstSeaBaseTruBrg, degrees)
9
10
11     0 (>L:NrstAptSearchReset, bool)
12   }
13
14 <!-- Set Search parameters and Aircraft position reference point-->
15 100 (>@C:NearestAirportMaximumItems, enum)
16 100 (>@C:NearestAirportMaximumDistance, nmiles)
17 (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
18 (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)
19
20 <!-- Loop through the NearestAirport list searching for Seaplane Base, AirportKind = 3 -->
21
22
23
24 (L:LoopingThroughNearestAirportList, bool) 1 ==
25   if{
26     (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
27     (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
28     (@C:NearestAirportCurrentAirportKind) 3 ==
29     if{
30       (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
31       (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
32       0 (>L:LoopingThroughNearestAirportList, bool)
33     }
34   }
35 }
36 </Update>
```

- **Lines 3–12:** Search RESET. The gauge that this code snippet is from contains a Gauge Reset mouse click button. Clicking it performs the following:

```
1 (>L:NrstAptSearchReset, enum)
(A:Fuel weight per gallon, pounds per gallon) 0 ==
  if{ (>K:RELOAD_USER_AIRCRAFT) } // FSX
  els{ (>K:RELOAD_PANELS) } // FS9
```

Refer to the [Panel Reload Gauge](#) chapter for more information about this very useful Reload tool.

Then, in the subsequent gauge update cycle, lines 5 through 11 are executed.

- ❑ **Lines 24–35:** Loop through the [NearestAirport](#) search results list looking for the first occurrence of [NearestAirportCurrentAirportKind](#) = 3. When found, LVars `L:NrstSeaBaseDist` and `L:NrstSeaBaseTruBrng` 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:

#### Code That Uses Results of the Database Search Starts Too Soon

The problem is that 5 gauge update cycles are consumed before the gps database completes its search for 100 airports within 100 nmiles and the [NearestAirport](#) list is returned.

By the 5<sup>th</sup> gauge update cycle, 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.

The conclusion is that code that looks through the search results for Seaplane Bases should not start in the same gauge update cycle that the nearest airport search itself begins. Instead, it should begin execution *after* the database search produces the list of nearest airports. In other words, code that relies on results of a database search must be delayed until the search concludes.

The solution is to utilize a cycle skipping process to delay execution of the code.

## Cycle Skip Technique 1 - Cycle Counting

The next script example 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 the nearest airport list for a seaplane base.

```
1 <Update Frequency="18" Hidden="No">
2 <!-- NearestAirport Search RESET -->
3 (L:NrstAptSearchReset, bool) 1 ==
4   if{
5     -1 (>L:IndexPointer, enum)
6     1 (>L:LoopingThroughNearestAirportList, bool)
7     0 (>L:NrstSeaBaseDist, nmiles)
8     0 (>L:NrstSeaBaseTruBrg, degrees)
9     -1 (>L:NrtsAptSearchCycleSkipCounter, enum)
10    4 (>L:NrtsAptSearchCyclesToSkip, enum)
11    0 (>L:NrstAptSearchReset, bool)
12  }
13
14 <!-- Set Search parameters and Aircraft position reference point-->
15 100 (>@C:NearestAirportMaximumItems, enum)
16 100 (>@C:NearestAirportMaximumDistance, nmiles)
17 (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
18 (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)
19
20 <!-- Loop through the NearestAirport list searching for Seaplane Base, AirportKind = 3 -->
21 (L:NrtsAptSearchCycleSkipCounter, enum) ++ (>L:NrtsAptSearchCycleSkipCounter, enum)
22 (L:NrtsAptSearchCycleSkipCounter, enum) (L:NrtsAptSearchCyclesToSkip, enum) &gt;;
23   if{
24     (L:LoopingThroughNearestAirportList, bool) 1 ==
25     if{
26       (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
27       (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
28       (@C:NearestAirportCurrentAirportKind) 3 ==
29       if{
30         (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
31         (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
32         0 (>L:LoopingThroughNearestAirportList, bool)
33       }
34     }
35   }
36 </Update>
```

- ❑ **Lines 3-12:** The following cycle count parameters have been added:
  - 1 (>L:NrstArptSearchCycleSkipCounter, enum) and
  - 4 (>L:NrstArptSearchCyclesToSkip, enum)
- ❑ **Line 9:** The cycle skip counter. It is reset to 0 when the Search RESET mouse button is clicked.
- ❑ **Line 10:** The pre-selected number of update cycles to skip.

- ❑ **Line 21:** The cycle skip counter is incremented by 1 each update cycle.
- ❑ **Lines 21-22:** 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 24-34).

When `L:NrstArptSearchCyclesToSkip` is set too low, lines 24-34 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 [NearestAirport](#) results are available.

In this example, when `L:NrstArptSearchCyclesToSkip = 3` or less, the first seaplane base found is ident W55, which is incorrect. When `CyclesToSkip` is set to 4 or more, the first seaplane base found is ident S60, which is correct.

## Cycle Skip 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 22 checks if [NearestAirportItemsNumber](#), which is the total number of airports found in the nearest airport 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 24-34 executed. This time, the [NearestAirport](#) list is available before the loop begins and the nearest seaplane base, **S60**, is not bypassed.

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

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 absolutely necessary that the gps\_500 does that in this particular situation – to delay a *display* loop until the search concludes. If the user is simply *displaying* the Nearest list, the display script will just 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.

## Cycle Skip Example 2 – ICAO Transfers

### Cycle Skip Technique - Cycle Count (“Let fs9gps tell you” is not available)

Similar to Nearest searches, ICAO Transfers are a 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 script: a [NearestAirport](#) to [WaypointAirport](#) ICAO Transfer, cycle counting code following the ICAO Transfer, and a [WaypointAirport](#) Group write-to-LVar statement.

- ❑ **Line 33:** The ICAO Transfer. [NearestAirport](#) to [WaypointAirport](#) Group.
- ❑ **Line 34:** Write [WaypointAirportLatitude](#) to an LVar, **same** cycle as ICAO Xfer.
- ❑ **Line 35:** Write [WaypointAirportLongitude](#) to an LVar, **same** cycle as ICAO Xfer.
- ❑ **Lines 42 –51:**
  - If Line 36 `LoopingThroughNearestAirportList` is set to 2, then Lines 44 – 0 will be executed, otherwise they won't.
  - Starting in Line 44, the `ICAOXferCycleSkipCounter` is incremented by one each update cycle.
  - Only when it is equal to or greater than the prescribed `ICAOTransferCyclesToSkip` (Line 45),
  - are the [WaypointAirport](#) Group variables [Latitude](#) and [Longitude](#) written to LVars (Lines 47, 48).
  - Finally, Line 49 sets `LoopingThroughNearestAirportList` to zero, terminating the entire Nearest Search and ICAO Transfer process.
- ❑ **Lines 5 –13:** Resets cycle counters and LVar values.

Note that in the following <Update> section, LVAR write statements (Lines 36, 37) are included immediately following the ICAO Transfer (Line 35) to illustrate that the [WaypointAirport](#) Group variables including [Latitude](#) and [Longitude](#) are not accessible in the same cycle as the ICAO Transfer.

However, duplicate LVAR write statements (Lines 47, 48) are also included in a cycle count sequence (Lines 44 through 50) that follows the ICAO transfer to show that the [WaypointAirport](#) Group variables including [Latitude](#) and [Longitude](#) are ultimately accessible after a one gauge cycle delay.

```

1 <Update Frequency="18" Hidden="No">
2 <!-- NearestAirport ICAO Search RESET -->
3 (L:NrstAptSearchReset, bool) 1 ==
4   if{
5     -1 (>L:IndexPointer, enum)
6     1 (>L:LoopingThroughNearestAirportList, bool)
7     -1 (>L:ICAOXferCycleSkipCounter, enum)
8     0 (>L:ICAOXferCyclesToSkip, enum)
9     0 (>L:NrstArptCurDist, nmiles)
10    0 (>L:NrstArptCurTruBrg, degrees)
11    0 (>L:WptArptLat, degrees)
12    0 (>L:WptArptLon, degrees)
13    0 (>L:NrstAptSearchReset, bool)
14  }
15
16 <!-- Set Search parameters and Aircraft position reference point-->
17 10 (>@C:NearestAirportMaximumItems, enum)
18 100 (>@C:NearestAirportMaximumDistance, nmiles)
19 (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
20 (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)
21
22 <!-- Loop through the NearestAirport list searching for Seaplane Base, AirportKind = 3 -->
23 (L:LoopingThroughNearestAirportList, bool) 1 ==
24   if{
25     (@C:NearestAirportItemsNumber) 0 &gt;;
26     if{
27       (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
28       (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
29       (@C:NearestAirportCurrentAirportKind) 3 ==
30       if{
31         (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
32         (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
33         (@C:NearestAirportCurrentIcao) (>@C:WaypointAirportIcao) <!-- ICAO Xsfer -->
34         (@C:WaypointAirportLatitude, degrees) (>L:WptArptLat, degrees)
35         (@C:WaypointAirportLongitude, degrees) (>L:WptArptLon, degrees)
36         0 (>L:LoopingThroughNearestAirportList, bool)
37       }
38     }
39   }
40
41 <!-- After ICAO Xfer, delay accessing Waypoint Group by using a Cycle Count Technique -->
42 (L:LoopingThroughNearestAirportList, bool) 2 ==
43   if{
44     (L:ICAOXferCycleSkipCounter, enum) ++ (>L:ICAOXferCycleSkipCounter, enum)
45     (L:ICAOXferCycleSkipCounter, enum) (L:ICAOXferCyclesToSkip, enum) &gt;;=
46     if{
47       (@C:WaypointAirport Latitude, degrees) (>L:WptArptLat, degrees)
48       (@C:WaypointAirport Longitude, degrees) (>L:WptArptLon, degrees)
49       0 (>L:LoopingThroughNearestAirportList, bool)
50     }
51   }
52 </Update>

```

When Line 36 is set to zero, the loop terminates after the LVar write statements (Lines 34, 35) that follow the ICAO Transfer (Line 35). The [WaypointAirport](#) variables [Latitude](#) and [Longitude](#) are not yet accessible, so the LVars remain 0.0000, as shown below.

Similarly, when Line 36 is set to 2, but with `ICAOXferCyclesToSkip = 0` (Line 8), 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 ----- NearestAirportCurrent -----
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  9004
1       A      S13    S13    1  180   5.8  73  0
2       A      WA61   WA61    2   50   6.5 251  0
3       A      S60    S60    3  180   6.7 135  0
4       A      96WA   96WA    2  180   9.7  66  0
5       A      WA22   WA22    2  189  11.3 304  0
6       A      W10    W10    1  180  11.4 347  0      CTF  122.90  2400
7       A      S43    S43    1  166  11.8  68  0      CTF  123.00  2672
8       A      W55    W55    3  180  12.4 173  0
9       A      83Q    83Q    3  150  12.5 241  0
                                     0.00 12000

NEAREST AIRPORT --> L:Vars      WAYPOINT AIRPORT --> L:Vars
(L:NrstArptCurDist, nmiles):    6.7  (L:WptArptLat, degrees):    0.0000
(L:NrstArptCurTruBrg, degrees): 135  (L:WptArptLon, degrees):    0.0000

```

However, if Line 36 is set to 2 and Line 8, the `CyclesToSkip` variable, is edited to read:

1 (>L:ICAOXferCyclesToSkip, enum)

then the write statements in Lines 47 & 48 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 ----- NearestAirportCurrent -----
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  9004
1       A      S13    S13    1  180   5.8  73  0
2       A      WA61   WA61    2   50   6.5 251  0
3       A      S60    S60    3  180   6.7 135  0
4       A      96WA   96WA    2  180   9.7  66  0
5       A      WA22   WA22    2  189  11.3 304  0
6       A      W10    W10    1  180  11.4 347  0      CTF  122.90  2400
7       A      S43    S43    1  166  11.8  68  0      CTF  123.00  2672
8       A      W55    W55    3  180  12.4 173  0
9       A      83Q    83Q    3  150  12.5 241  0
                                     0.00 12000

NEAREST AIRPORT --> L:Vars      WAYPOINT AIRPORT --> L:Vars
(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, but to be safe, skip 4 cycles.

To get a better sense of the sequence of events, slow the update frequency rate (Line 1) down to 2, and watch the display after Gauge Reset (lines 5 – 13) is executed. 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.

### **Cycle skipping not critical if data only displayed**

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 as shown below. 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.

\* 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.

```

1 <Update Frequency="18" Hidden="No">
2 <!-- NearestAirport ICAO Search RESET -->
3 (L:NrstAptSearchReset, bool) 1 ==
4   if{
5     -1 (>L:IndexPointer, enum)
6     1 (>L:LoopingThroughNearestAirportList, bool)
7
8
9     0 (>L:NrstArptCurDist, nmiles)
10    0 (>L:NrstArptCurTruBrg, degrees)
11    0 (>L:WptArptLat, degrees)
12    0 (>L:WptArptLon, degrees)
13    0 (>L:NrstAptSearchReset, bool)
14  }
15
16 <!-- Set Search parameters and Aircraft position reference point-->
17 10 (>@C:NearestAirportMaximumItems, enum)
18 100 (>@C:NearestAirportMaximumDistance, nmiles)
19 (A:PLANE LATITUDE, degrees) (>@C:NearestAirportCurrentLatitude, degrees)
20 (A:PLANE LONGITUDE, degrees) (>@C:NearestAirportCurrentLongitude, degrees)
21
22 <!-- Loop through the NearestAirport list searching for Seaplane Base, AirportKind = 3 -->
23 (L:LoopingThroughNearestAirportList, bool) 1 ==
24   if{
25     (@C:NearestAirportItemsNumber) 0 !=
26     if{
27       (L:IndexPointer, enum) ++ (>L:IndexPointer, enum)
28       (L:IndexPointer, enum) (>@C:NearestAirportCurrentLine)
29       (@C:NearestAirportCurrentAirportKind) 3 ==
30       if{
31         (@C:NearestAirportCurrentDistance, nmiles) (>L:NrstArptCurDist, nmiles)
32         (@C:NearestAirportCurrentTrueBearing, degrees) (>L:NrstArptCurTruBrg, degrees)
33         (@C:NearestAirportCurrentIcao) (>@C:WaypointAirportIcao) <!-- ICAO Xsfer -->
34
35
36         0 (>L:LoopingThroughNearestAirportList, bool)
37       }
38     }
39   }
40
41 <!-- Write WaypointAirport Latitude and Longitude to L:Vars each gauge update cycle -->
42 (@C:WaypointAirport Latitude, degrees) (>L:WptArptLat, degrees)
43 (@C:WaypointAirport Longitude, degrees) (>L:WptArptLon, degrees)
44
45 </Update>

```

## 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 APT LOOP DISPLAY">
63   <Position X="10" Y="5"/>
64   <FormattedText X="800" Y="700" Font="Courier New" FontSize="12" LineSpacing="12"
65   Color="#100000" BackgroundColor="white" Bright="Yes">
66     <Color Value="blue"/>
67     <Color Value="green"/>
68     <String>
69       \{clr2}NEAREST AIRPORT SEARCH\n
70       \{clr3}%{(C:fs9gps:NearestAirportCurrentLatitude, degrees)}%!9.4f! :Current Lat\n{nr}
71       %((C:fs9gps:NearestAirportMaximumItems, enum))%!5d! :Max Items\n{nr}
72       %((@c:NearestAirportItemsNumber))%!4d! :Items Num\n
73       %((C:fs9gps:NearestAirportCurrentLongitude, degrees)}%!9.4f! :Current Lon\n{nr}
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           % (11 (>@c:NearestAirportCurrentLine))
83
84
85           \{clr}%{(@c:NearestAirportCurrentLine)}%!-5d!\{nr}
86           %{(@c:NearestAirportCurrentICAO)}%!16s!\{nr}
87           %{(@c:NearestAirportCurrentIdent)}%!6s!\{nr}
88           %{(@c:NearestAirportCurrentAirportKind)}%!5d!\{nr}
89           %{(@c:NearestAirportCurrentLongestAirportDirection, degrees)}%!5d!\{nr}
90           %{(@c:NearestAirportCurrentDistance, nmiles)}%!6.1f!\{nr}
91           %{(@c:NearestAirportCurrentTrueBearing, degrees)}%!5d!\{nr}
92           %{(@c:NearestAirportCurrentBestApproachEnum)}%!4d!\{nr}
93           %{(@c:NearestAirportCurrentBestApproach)}%!6s!\{nr}
94           %{(@c:NearestAirportCurrentComFrequencyName)}%!5s!\{nr}
95           %{(@c:NearestAirportCurrentComFrequencyValue, mhz)}%!8.2f!\{nr}
96           %{(@c:NearestAirportCurrentLongestRunwayLength, feet)}%!7d!\n
97
98           % (11 ++ s1 12 &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:

```

62 <Element Name="NEAREST APT LOOP DISPLAY">
63   <Position X="10" Y="5"/>
64   <FormattedText X="800" Y="700" Font="Courier New" FontSize="12" LineSpacing="12"
65   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\n{nr}
71       %((C:fs9gps:NearestAirportMaximumItems, enum))%!5d! :Max Items\n{nr}
72       %((@c:NearestAirportItemsNumber))%!4d! :Items Num\n
73       %((C:fs9gps:NearestAirportCurrentLongitude, degrees)}%!9.4f! :Current Lon\n{nr}
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
78       %((@c:NearestAirportItemsNumber) s2 0 !=)
79       %if)
80       %0 sp1)
81       %loop)
82       %11 (>@c:NearestAirportCurrentLine))
83       %((@c:NearestAirportCurrentAirportKind) 3 ==)
84       %if)\{nr}
85       \{clr}\{(@c:NearestAirportCurrentLine)}%!-5d!\{nr}
86       %((@c:NearestAirportCurrentICAO))%!16s!\{nr}
87       %((@c:NearestAirportCurrentIdent))%!6s!\{nr}
88       %((@c:NearestAirportCurrentAirportKind))%!5d!\{nr}
89       %((@c:NearestAirportCurrentLongestAirportDirection, degrees))%!5d!\{nr}
90       %((@c:NearestAirportCurrentDistance, nmiles))%!6.1f!\{nr}
91       %((@c:NearestAirportCurrentTrueBearing, degrees))%!5d!\{nr}
92       %((@c:NearestAirportCurrentBestApproachEnum))%!4d!\{nr}
93       %((@c:NearestAirportCurrentBestApproach))%!6s!\{nr}
94       %((@c:NearestAirportCurrentComFrequencyName))%!5s!\{nr}
95       %((@c:NearestAirportCurrentComFrequencyValue, mhz))%!8.2f!\{nr}
96       %((@c:NearestAirportCurrentLongestRunwayLength, feet))%!7d!\n
97       %end)
98       %11 ++ s1 12 &lt;t;)
99       %next)
100      %end)
101    </String>
102  </FormattedText>
103 </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 Line	ICAO	111	-----	NearestAirportCurrent	-----	-----	-----	-----	-----	-----
Line	123456789012	Ident	Kind	Rwy*	Dist	Brg	Best	Appr	Com	Freq Length
3	A	S60	S60	3 180	6.7	135	0			0.00 10000
8	A	W55	W55	3 180	12.4	173	0			0.00 5000
9	A	83Q	83Q	3 150	12.5	241	0			0.00 12000
10	A	OW0	OW0	3 201	12.5	172	0			0.00 9500
25	A	W36	W36	3 140	21.0	162	0			0.00 5000
46	A	2WA2	2WA2	3 200	27.4	186	0			0.00 15000
50	A	WN19	WN19	3 110	30.9	196	0			0.00 6000
75	A	21H	21H	3 156	40.9	343	0			0.00 5000
80	A	W37	W37	3 40	42.2	190	0			0.00 5500
97	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](#)). [VCY\\_ \\_ \\_ \\_1CD\\_ \\_](#) is the associated ICAO.

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. Icao Search variables

```
(C:fs9gps:IcaoSearchMatchedIcaosNumber)  
(C:fs9gps:IcaoSearchMatchedIcao)  
(C:fs9gps:IcaoSearchCurrentIcao)
```

yield the following:

```
MatchedIcaosNumber: 1
ICAO                1 1
Index 1 2 3 4 5 6 7 8 9 0 1 2
0 V                1 C D
```

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                1 1
Index 1 2 3 4 5 6 7 8 9 0 1 2
0 V      S O C A C A
```

"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' :

**MatchedIcaoNumber: 3**

	ICAO										1	1
Index	1	2	3	4	5	6	7	8	9	0	1	2
0	V			E	G	D	C	C	V			
1				L	F	K	C	C	V			
2	V	Y	B							C	V	

**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' :

**MatchedIcaoNumber: 2**

	ICAO										1	1
Index	1	2	3	4	5	6	7	8	9	0	1	2
0	V	H	E							C	V	O
1	V	K	1							C	V	O

**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 will access the second VOR:

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

```
3 IcaoSearchCursorPosition
CVO IcaoSearchCurrentIdent
VK1 CVO IcaoSearchCurrentIcao
V IcaoSearchCurrentIcaoType
K1 IcaoSearchCurrentIcaoRegion
2 IcaoSearchMatchedIcaosNumber
1 IcaoSearchMatchedIcao
```

**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)
```

```
VK1 CVO WaypointVorICAO
CVO WaypointVorIdent
CORVALLIS WaypointVorName
44.4996 WaypointVorLatitude
-123.2937 WaypointVorLongitude
3 IcaoSearchCursorPosition
CVO IcaoSearchCurrentIdent
VK1 CVO IcaoSearchCurrentIcao
V IcaoSearchCurrentIcaoType
K1 IcaoSearchCurrentIcaoRegion
2 IcaoSearchMatchedIcaosNumber
1 IcaoSearchMatchedIcao
```

Now, [WaypointVorIcao](#) = [IcaoSearchCurrentIcao](#) = [VK1\\_ \\_ \\_CVO\\_ \\_](#), and variables [WaypointVorLatitude](#) and [WaypointVorLongitude](#) return the desired Lat and Lon of the CORVALLIS VOR.

## RESOLVING MULTIPLE IDENT MATCHES

Because Idents are not unique, ICAO Searches sometimes 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 [ICAO Search](#) 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 RESULTS TO FND NEAREST FACILITY -->
10 (@c:IcaoSearchMatchedIcaosNumber) 0 &gt; (L:ResolveMultipleIdents, bool) 1 == and
11 if{
12     (L:ResolveMultipleIdentsInit, 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:ResolveMultipleIdentsInit, bool)
22     }
23
24     (L:NumberOfFacilitiesChecked, enum) (>@c:IcaoSearchMatchedIcao)
25     (@c:IcaoSearchCurrentIcao) (>@c:FacilityIcao) <!-- ICAO Xfer -->
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:FacilityCurrentDistance, nmiles)
33         (L:FacilityNearestDistance, nmiles) (L:FacilityCurrentDistance, nmiles) &gt;;
34         if{
35             (L:FacilityCurrentDistance, 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) 1 + (@c:IcaoSearchMatchedIcaosNumber) &gt;;
41         if{
42             0 (>L:ResolveMultipleIdents, bool)
43         }
44     }
45 }

```

- ❑ **Line 10.** The Loop is executed if there are multiple ICAOs returned by [ICAO Search](#) 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
```

Use of this shortcut to avoid an ICAO Search is a special case that is safe for Airports only.

## 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. It's also important to understand when working with the FSX gps module, although some of the need for the ICAO Transfer has been alleviated because, with FSX, the Nearest Groups (the 'first' group) are populated with several more variables from the Waypoint Groups (the 'second' group) to begin with.

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
```

```
----- NearestAirportCurrent -----
      ICAO      111
Line 123456789012 Ident Kind Rwy* Dist  Brg Best Appr Com  Freq Length
0   A      KDPA      KDPA   1  14   4.6 338 13  ILS  twr  120.9  7573
1   A      LL10      LL10   1 179   6.1 177  0
2   A      1C5       1C5   1 178   9.2 157  8  GPS  CTF  122.9  3363
3   A      85LL      85LL   2 359   9.4 194  0
4   A      06C       06C   1 109  10.4  28  0
5   A      IS23      IS23   3 179  11.0 341  0
6   A      LL22      LL22   1  89  11.3 123  0
7   A      KARR      KARR   1  89  12.5 252 13  ILS  twr  120.6  6491
8   A      2IS0      2IS0   1 179  13.4 142  0
9   A      LL51      LL51   2 359  14.2 192  0
```

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, all of 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 can 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

----- WaypointAirportFrequency -----
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:fs9gps:NearestAirportMaximumItems, enum)
10 75 (>C:fs9gps:NearestAirportMaximumDistance, nmiles)
11
12 <!-- SET REFERENCE POINT VARIABLES (AIRCRAFT POSITION) -->
13 (A:PLANE LATITUDE, Radians) (>C:fs9gps:NearestAirportCurrentLatitude, Radians)
14 (A:PLANE LONGITUDE, Radians) (>C:fs9gps:NearestAirportCurrentLongitude, Radians)
15
16 <!-- ICAO TRANSFER: NearestAirport to WaypointAirport -->
17 0 (>C:fs9gps:NearestAirportCurrentLine)
18 (@c:NearestAirportCurrentICAO) (@c:WaypointAirportICAO) <!-- ICAO Xfer -->
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>
```

```

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

```

```

115 <Element Name="WAYPOINT APT FREQUENCIES LOOP DISPLAY">
116   <Position X="10" Y="190"/>
117   <FormattedText X="800" Y="700" Font="Courier New" FontSize="12" LineSpacing="12"
118   Color="#100000" BackgroundColor="White" Bright="Yes" >
119     <Color Value="blue"/>
120     <Color Value="green"/>
121     <Color Value="red"/>
122     <String>
123       \n
124       \{clr2}WAYPOINT AIRPORT FREQUENCIES\n
125       \{clr4}ICAO      111\n
126       \{clr3}\{(@c:WaypointAirportFrequenciesNumber)}\!6d! :Frequencies Num\n
127       \{clr4}123456789012\n
128       \{clr3}\{(@c:WaypointAirportICAO)}\!12s! :Waypoint Airport ICAO
129       \n\n\{clr2}
130       ---- WaypointAirportFrequency ----\n
131       Freq      Name Limit Value Type\n
132       \{(@c:WaypointAirportFrequenciesNumber) s2 0 !=)
133       \{if}
134       \{0 sp1}
135       \{loop}
136       \{l1 (>@c:WaypointAirportCurrentFrequency)}
137       \{clr}\{(@c:WaypointAirportCurrentFrequency)}\!-3d!
138       \{(@c:WaypointAirportFrequencyName)}\!13s!
139       \{(@c:WaypointAirportFrequencyLimit)}\!5d!
140       \{(@c:WaypointAirportFrequencyValue, mHz)}\!9.2f!
141       \{(@c:WaypointAirportFrequencyType)}\!6d!\n
142       \{l1 ++ s1 12 &lt;);}
143       \{next}
144     \{end}
145   </String>
146 </FormattedText>
147 </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. Airport 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 (Kind).

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

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

### ❑ `WaypointAirportLongestRunwayDirection` (degrees) [Get]

Direction (True) of the longest runway. Only one direction of the runway pair is returned. About ~90% of the time, FS returns the direction of the eastward (0 to 180 degrees) facing runway. Why the 10% exceptions, I don't know.

### ❑ `WaypointAirportType` (enum) [Get]

A number representing Airport Type.

## WaypointAirportType

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

<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.

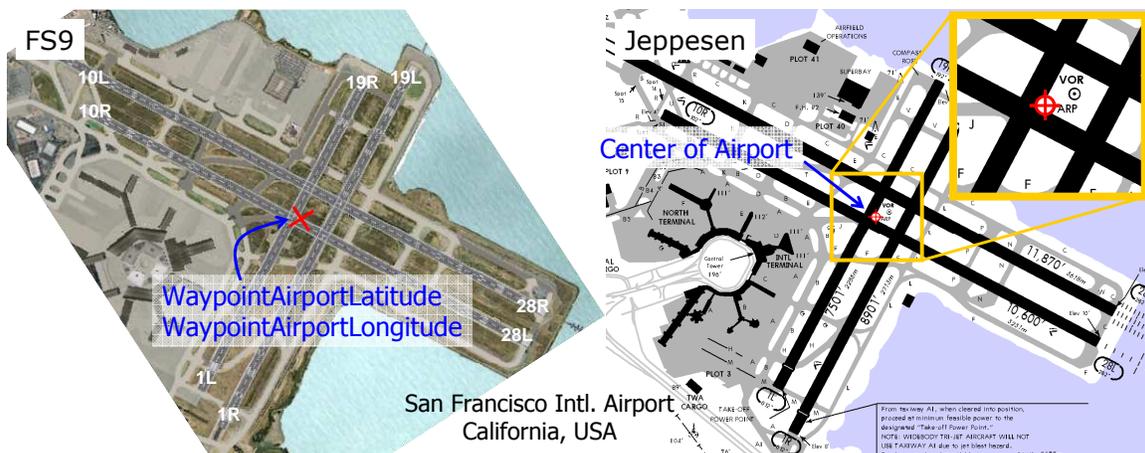
### ⊘ WaypointAirportRegion (string) [Get]

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, at [WaypointAirportLatitude](#), [Longitude](#).

❑ **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. The higher the number, the more precise the approach.

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

<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]**

As far as I can tell, [AirportRadarCoverage](#) is not functional in FS9 or FSX

⊗ **WaypointAirportAirspace (string) [Get]**

As far as I can tell, [AirportAirspace](#) is not functional in FS9 or FSX

❑ **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. Communication frequency names include:

- Approach
- ATIS, ASOS, AWOS
- CTAF
- Unicom
- Multicom
- Clearance
- Clearance Pre-Taxi
- Ground
- Tower
- Departure
- FSS
- Remote Clearance Delivery

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]**

Communication frequency = 1. Navigation frequency (ILS or LOC) = 2.

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

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

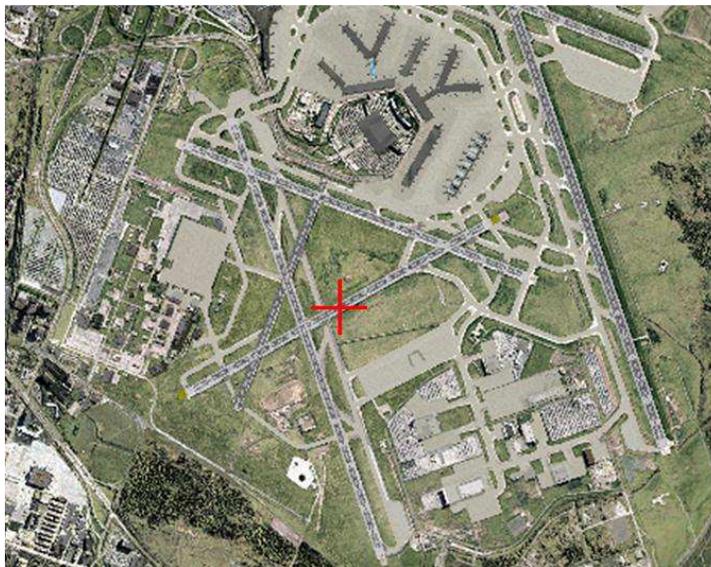
❑ **WaypointAirportRunwaysNumber (enum) [Get]**

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]**

Direction (True) of the runway, not the magnetic direction universally associated with runway direction designations. Only the first direction (true) of the pair of directions for each runway is returned. In the example pics below, [WaypointAirportRunwayDirection](#) of Runway 01-19 is 345.2 degrees.

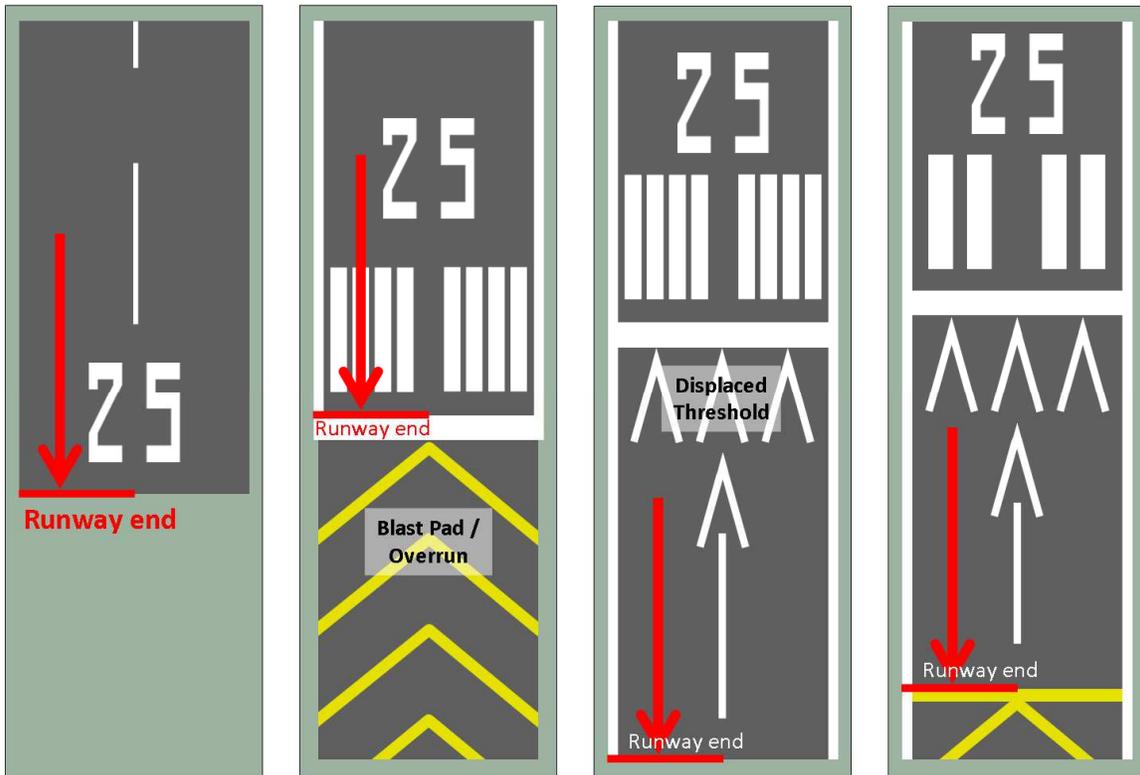


❑ **WaypointAirportRunwayDesignation (string) [Get]**

The name of the runway pair. For example, "04L-22R".

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

Runway length, measured from runway ends as shown below. In the fs9gps module, displaced thresholds are included in runway measured length.



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

Width of the runway. On runways where the sides are marked with white stripes, [RunwayWidth](#) is measured outside to outside the stripes as depicted below.



### ❑ **WaypointAirportRunwaySurface (enum) [Get]**

A number representing runway surface type.

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

<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.

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

<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 procedure 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.

## FSX-ONLY VARIABLES – WaypointAirport Group (these add nothing to FS9)

As far as I can determine, the FSX-only [WaypointAirport](#) variables either do not function properly or add nothing to the existing FS9 gps.dll functionality.

### WaypointAirportApproach Variables (these do not function reliably)

The [WaypointAirportApproach](#) variables (FSX-only) appear to access and synthesize some [FlightPlanWaypointApproach](#) waypoints from the [WaypointAirport](#) group. In my opinion, these variables do not function reliably. Furthermore, I don't understand why they would be useful even if they worked correctly.

From what I can tell, the variables return a list of two to four approach waypoints where the first, and sometimes the second, is associated with the final approach leg, or the intermediate approach plus final approach legs. The last waypoint returned is associated with a missed approach fix or procedure. I cannot discern the reason why two, three or four waypoints are returned, and I cannot always correlate between the FS9 [FlightPlanWaypointApproach](#) variables, which appear to function correctly, and these FSX-only variables.

Finally, I note that [WaypointAirportSelectedApproach](#) is critical to this set of variables, yet it is not even listed in the SDK. It occurred to me that such a variable might be needed, but I had to inspect the FSX gps.dll module to find its name.

#### ✘ [WaypointAirportSelectedApproach](#) (enum) [Get, Set]

The loaded approach. It is analogous to [WaypointAirportCurrentApproach](#), but must be used rather than [WaypointAirportCurrentApproach](#) for this set of FSX-only variables.

#### ⊗ [WaypointAirportApproachSelectedTransition](#) (enum) [Get, Set]

Does not seem to function at all.

#### ✘ [WaypointAirportApproachNumberLegs](#) (enum) [Get]

The number of waypoints returned, analogous to [FlightPlanApproachWaypointsNumber](#), but obviously not the same value.

#### ✘ [WaypointAirportApproachCurrentLeg](#) (enum) [Get, Set]

The index pointer, analogous to [FlightPlanWaypointApproachIndex](#).

**✘ WaypointAirportApproachCurrentLegIcao (string) [Get]**

The Ident of the waypoint, not the ICAO.

**⊖ WaypointAirportApproachCurrentLegType (string?) [Get]**

Does not appear to function. Always returns a zero. Additionally, 'Type' would typically be a number, not a string.

**✘ WaypointAirportApproachCurrentLegBearing (degrees) [Get]**

Magnetic course of the leg associated with the waypoint.

**✘ WaypointAirportApproachCurrentLegDistance (nmiles) [Get]**

Length of the leg associated with the waypoint. Often, this cannot be correlated with and analogous leg from [FlightPlanWaypointApproach](#) group.

**✘ WaypointAirportApproachCurrentLegIsMinutes (bool) [Get]**

A flag associated with a timed missed approach hold procedure.

FS9 & FSX  
FlightPlanWaypointApproach

```

KICT :FlightPlanApproachAirportIdent
13 :FltPlnApprType 9 :FltPlnApprWptsNum
ILS 19R :FltPlnApprName
ICT :FPTransName

----- FlightPlanWaypointApproach -----
ICAO 111
Idx ICAO Name Type Mode Course Dist
0 VK3 ICT ICT 1 1 -1.0 0.0
1 WK3KICTHOVER HOVER 1 1 93.1 8.8
2 WK3KICTHOVER HOVER 3 1 328.0 21.3
3 WK3KICTCF19R CF19R 1 2 197.0 8.9
4 WK3KICTHOVER HOVER 1 2 193.0 6.3
5 RK3KICTRW19R RW19R 1 2 193.0 4.8
6 9 3 193.0 5.3
7 VK3 ICT ICT 1 3 344.7 13.7
8 VK3 ICT ICT 6 3 180.0 14.3
                    
```

✈ FSX-only  
WaypointAirportApproach

```

KICT :WaypointAirportIdent
3 :Cur Appr ILS 19R :Name
3 :Sel Appr FSX
2 :Cur Trans HOVER :Name
0 :Sel Trans FSX 3 :Num Legs FSX

WaypointAirportApproachCurrent FSX
Leg Leg Leg Leg Leg Is
Leg ICAO Type Brg Dist Min
0 HOVER 0 193.0 6.3 0
1 RW19R 0 193.0 4.8 0
2 ICT 0 180.0 0.0 1
                    
```

Leg 0 and 1: Intermediate and Final Approach legs

Leg 2: Missed approach Hold

### **WaypointAirportSelectedFrequency (redundant with existing FS9 variables)**

The [WaypointAirportSelectedFrequency](#) variables function, but are redundant with [WaypointAirportFrequency](#) variables discussed before. That is, they return the same list of airport frequencies.

I also note that these two variables are not listed in the SDK, but can be found when the FSX gps.dll module is inspected for variable names.

#### **☒ [WaypointAirportSelectedFrequencyIndex](#) (enum) [Get, Set]**

Index pointer for the airport frequency list, same as [WaypointAirportCurrentFrequency](#).

#### **☒ [WaypointAirportSelectedFrequencyValue](#) (MHz) [Get]**

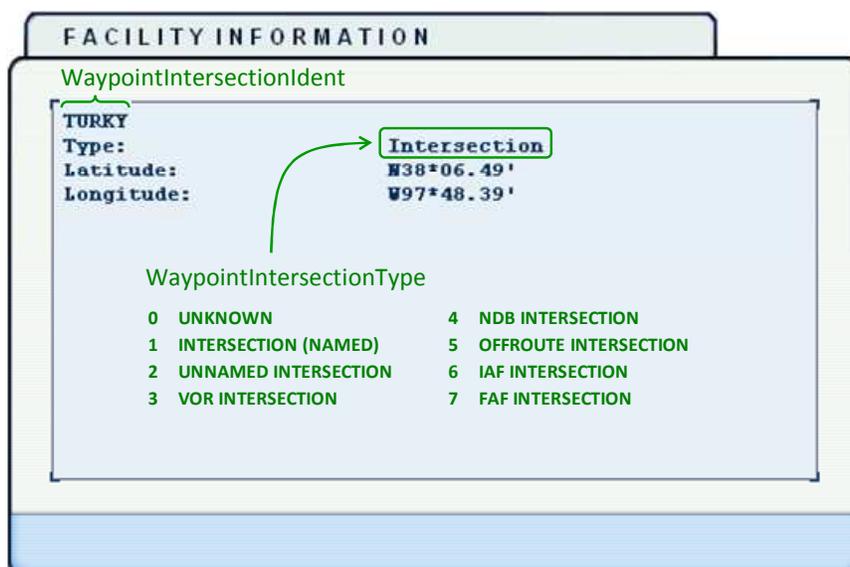
Radio frequency value. Same as [WaypointAirportFrequencyValue](#).

## 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 [WaypointIntersection](#) variables for the same Intersection.

<b>WK3KHUTTURKY</b>	WaypointIntersectionICAO
TURKY	WaypointIntersectionIdent
	WaypointIntersectionCity
K3	WaypointIntersectionRegion
38.1081	WaypointIntersectionLatitude
-97.8066	WaypointIntersectionLongitude
1	WaypointIntersectionType
GNP	WaypointIntersectionNearestVorIdent
2	WaypointIntersectionNearestVorType
327	WaypointIntersectionNearestVorTrueRadial
321	WaypointIntersectionNearestVorMagneticRadial
158	WaypointIntersectionNearestVorDistance

❑ **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.

#	Intersection Type	#	Intersection Type
0	UNKNOWN	4	NDB
1	NAMED	5	OFFROUTE
2	UNNAMED	6	IAF
3	VOR	7	FAF

<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]**

Returns a blank string. [WaypointIntersectionCity](#) is apparently not populated, or at least, not functional in the fs9gps data base.

❑ **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 the VOR nearest the intersection.

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

An enum representing the VOR Type of the nearest VOR to the intersection.

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

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

**WaypointIntersectionNearestVor ...**

❑ **MagneticRadial (degrees) [Get]**

Direction (magnetic) from the nearest VOR to the intersection. This is the VOR radial that the intersection is on as shown in the figure on the right.

❑ **TrueRadial (degrees) [Get]**

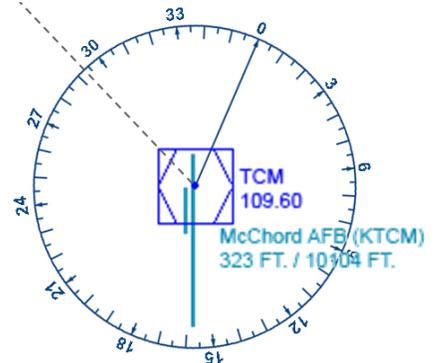
Bearing (true) from the nearest VOR to the intersection. Degrees in the example shown.

❑ **Distance (NMiles) [Get]**

Distance from the nearest VOR to the intersection. 6.2 nmiles in the example shown.

△ ARVAD

```
ARVAD :WaypointIntersectionIdent
TCM :WaypointIntersectionNearestVorIdent
294 :WaypointIntersectionNearestVorMagneticRadial
316 :WaypointIntersectionNearestVorTrueRadial
6.2 :WaypointIntersectionNearestVorDistance
```



# 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.

**FACILITY INFORMATION**

**WaypointNdbName**      **WaypointNdbIdent**

**LAKE LAWN (DELAVAN)**      **((LVV))**

Type:      **NDB**

Class:      **MH**

Frequency:      **404.0 kHz** ← **WaypointNdbFrequency**

Morse:      . . . . .

**WaypointNdbType = 2**

Type / Class of NDB, Transmission Power, real life Effective Range  
(which may or may not match how it is modeled in FS9)

<b>0</b> - Unknown	<b>3 - H</b> 50 to 1,999 watts 50 - 75 NM. Enroute Airway Beacon, common in Canada and Caribbean
<b>1 - Compass locator</b> below 25 watts 15 - 25 NM. Most common in US. Outer Marker, for example	<b>4 - HH</b> 2,000+ watts 75 - 125 NM. High powered Beacon found along coasts
<b>2 - MH</b> below 50 watts 25 - 50 NM. DB Approach Facility found at or near airports where it is the primary approach aid	

The following is a snapshot of all [WaypointNdb](#) variables for the LVV NDB.

<b>NK5</b>	<b>LVV</b>	WaypointNdbICAO
LVV		WaypointNdbIdent
2		WaypointNdbType
LAKE LAWN (DELEVAN)		WaypointNdbName
		WaypointNdbCity
K5		WaypointNdbRegion
42.6988		WaypointNdbLatitude
-88.5932		WaypointNdbLongitude
948		WaypointNdbElevation
404		WaypointNdbFrequency
0		WaypointNdbWeatherBroadcast
2		WaypointNdbMagneticVariation

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

The ICAO identifier 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 lists NDB Type and Class, and real life Transmission Power and Effective Range (which may not match how it is modeled in Flight Simulator. I’m not sure):

**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]**

Returns a blank string. [WaypointNdbCity](#) is apparently not populated, or at least, not functional in the fs9gps data base.

#### ❑ **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

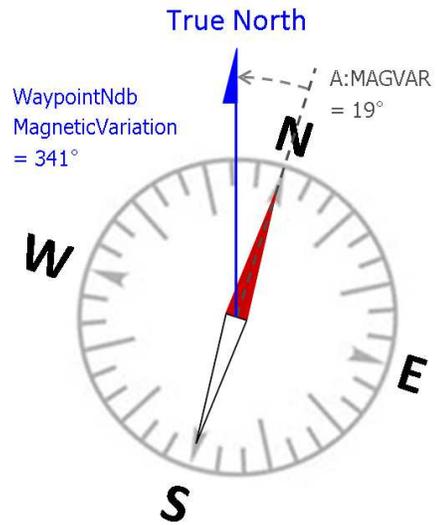
However, having scanned most NDBs in the fs9gps database, so far I have found all NDBs have [WaypointNdbWeatherBroadcast](#) = 0. Consequently, this variable may not represent an active feature in FS9.

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

**WaypointNdbMagneticVariation** is the compass direction of true north. An integer is always returned.

In this example, A:GPS MAGVAR and A:MGVAR would equal 19° (19E).

To derive the magnetic course from a gps var that returns true bearing (which is mostly the case) subtract A:MAGVAR from the true bearing.



## FSX-ONLY VARIABLES – WaypointNDB Group

Similar to the other variables in this Group, these FSX-only variables require an ICAO Transfer of an NDB ICAO into [WaypointNdbICAO](#) before they return values.

These variables belong to the WaypointNdb Group and are not related to, nor function with, NearestNdb Search.

### **WaypointNdbNearestAirportId (string) [Get]**

The three to four character Ident of the airport closest to the NDB identified by the ICAO of the NDB.

### **WaypointNdbNearestAirportLongestRunwayDirection (degrees) [Get]**

Direction (True) of the longest runway at the airport closest to the NDB.

### **WaypointNdbNearestAirportKind (enum) [Get]**

A number representing Airport Class (Kind).

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

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

### **WaypointNdbNearestAirportBearing (degrees) [Get]**

The bearing (True) to the nearest airport. Presumed this is referenced to the center of the airport facility, [WaypointAirportLatitude](#) and [Longitude](#).

### **WaypointNdbNearestAirportDistance (nmiles) [Get]**

Distance to the nearest airport. Presumed this is referenced to the center of the airport facility, [WaypointAirportLatitude](#) and [Longitude](#).

# 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 screenshot shows a 'FACILITY INFORMATION' window for a VOR facility. The facility name is 'WARITA (TOKYO)' and the ICAO identifier is 'WRE'. The facility type is 'VOR/DME' and the class is 'High altitude'. The frequency is '117.90 MHz'. The Morse code is shown as a series of dashes. Below the facility information, there are three columns of codes and their corresponding names: WaypointVorType, WaypointVorClass, and WaypointVorFrequency. Arrows point from the facility information to the corresponding code columns.

WaypointVorName	WaypointVorIdent
WARITA (TOKYO)	(WRE)
Type:	VOR/DME
Class:	High altitude
Frequency:	117.90 MHz ← WaypointVorFrequency
Morse:	- - - - -

WaypointVorType = 2		WaypointVorClass = 3 (also known as VOR Kind)
0 = UNKNOWN	4 = TACAN	0 = UNKNOWN
1 = VOR	5 = VORTAC	1 = TERMINAL
2 = VOR_DME	6 = ILS	2 = LOW_ALT
3 = DME	7 = VOT	3 = HIGH_ALT
		4 = ILS
		5 = VOT

The following is a snapshot of all [WaypointVor](#) variables for the NRE VOR.

```

VRJ      NRE      WaypointVorICAO
NRE      WaypointVorIdent
2        WaypointVorType
3        WaypointVorClass
NARITA (TOKYO) WaypointVorName
          WaypointVorCity
RJ       WaypointVorRegion
35.7823  WaypointVorLatitude
140.3625 WaypointVorLongitude
154      WaypointVorElevation
117.9    WaypointVorFrequency
0        WaypointVorWeatherBroadcast
7        WaypointVorMagneticVariation

```

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

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

<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]**

Returns a blank string. [WaypointVorCity](#) is apparently not populated, or at least, not functional in the fs9gps data base.

❑ **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 (MHz) [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

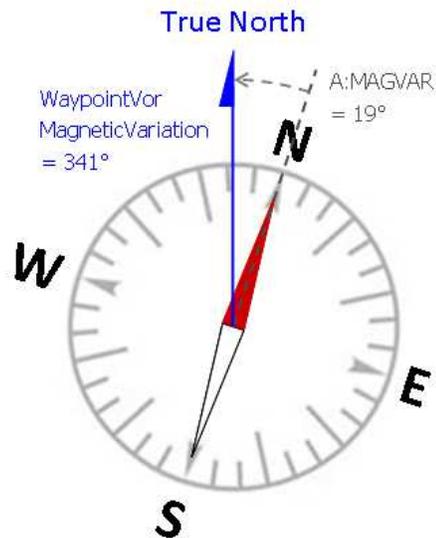
However, having scanned most VORs in the fs9gps database, so far I have found all VORs have [WaypointVorWeatherBroadcast](#) = 0. Consequently, this variable may not represent an active feature in fs9gps.

### ❑ **WaypointVorMagneticVariation (degrees) [Get]**

[WaypointVorMagneticVariation](#) is the compass direction of true north.

In the example shown (SEA VORTAC, Seattle Washington, USA. Magnetic Variation = 19E), to derive magnetic variation (similar to A:MAGVAR and A:GPS MAGVAR), subtract [WaypointVorMagneticVariation](#) from 360°.

In FSX, A:MAGVAR is actually 17.3°, curiously reflecting a different mag variation in the gps module vs. the A:MAGVAR system variable. Don't know why.



## FSX-ONLY VARIABLES – WaypointVOR Group

Similar to the other variables in this Group, these FSX-only variables require an ICAO Transfer of a VOR ICAO into [WaypointVORICAO](#) before they return values.

These variables belong to the WaypointVOR Group and are not related to, nor function with, NearestVOR Search.

### ✘ **WaypointVORNearestAirportId (string) [Get]**

The three to four character Ident of the airport closest to the VOR referenced by the ICAO.

### ✘ **WaypointVORNearestAirportLongestRunwayDirection (degrees) [Get]**

Direction (True) of the longest runway. Only one direction of the runway pair is returned. About ~90% of the time, FS returns the direction of the eastward (0 to 180 degrees) facing runway. Why the 10% exceptions, I don't know.

### ✘ **WaypointVORNearestAirportKind (enum) [Get]**

A number representing Airport Class (Kind).

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

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

### ✘ **WaypointVORNearestAirportBearing (degrees) [Get]**

The bearing (True) to the nearest airport. Presumed this is referenced to the center of the airport facility, [WaypointAirportLatitude](#) and [Longitude](#).

### ✘ **WaypointVORNearestAirportDistance (nmiles) [Get]**

Distance to the nearest airport. Presumed this is referenced to the center of the airport facility, [WaypointAirportLatitude](#) and [Longitude](#).

## 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 (nmiles) [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.

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

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

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

Direction (True) of the longest runway. Only one direction of the runway pair is returned. About ~90% of the time, FS returns the direction of the eastward (0 to 180 degrees) facing runway. Why the 10% exceptions, I don't know.

❑ **NearestAirportCurrentDistance (nmiles) [Get]**

The distance of each airport in the [NearestAirport](#) search from the reference point.

❑ **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.

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

<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 = Unicom
- mul = Multicom

Ground, Clearance, Clearance Pre-Taxi, 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

```
----- NearestAirportCurrent -----  
ICAO 111  
Line 123456789012 Ident Kind Rwy* Dist Brg Best Appr Com Freq Length  
0 A CKJ2 CKJ2 2 180 13.1 293 0  
1 A CKK7 CKK7 1 179 17.9 65 8 GPS uni 122.7 3109  
2 A CJB3 CJB3 1 149 20.2 57 0 mul 122.7 3000  
3 A CJL6 CJL6 1 180 22.9 226 0 mul 123.2 3248  
4 A CKA4 CKA4 2 124 24.5 102 0 0.0 2875  
5 A KPMB KPMB 1 160 25.9 191 7 VOR CTF 122.8 3797  
6 A NA67 NA67 2 177 26.8 211 0 0.0 2000  
7 A CJL5 CJL5 2 96 29.7 11 0 mul 123.2 3000  
8 A CJT8 CJT8 2 90 30.0 287 0 0.0 3000  
9 A 8NA6 8NA6 2 87 30.5 214 0 0.0 3800  
10 A CKJ7 CKJ7 2 137 30.5 314 0 mul 123.4 3000  
11 A 5ND3 5ND3 2 156 31.7 195 0 0.0 1600  
12 A CYWG CYWG 1 188 32.9 352 13 ILS twr 118.3 10989  
13 A CKZ7 CKZ7 1 90 33.7 250 0 uni 122.8 2900  
14 A CKD7 CKD7 2 177 34.3 274 0 0.0 3500
```

**NearestAirportCurrentLongestRunwayLength (feet) [Get]**

Length of the longest runway at the airport.

**FSX-ONLY VARIABLES – NearestAirport Group**

These FSX-only variables add to the list of airport variables that are retrievable through a NearestAirport search.

**NearestAirport Airport Variables**

**NearestAirportSelected (enum) [Get, Set]**

Index for the FSX NearestAirportSelectedAirport variables.  
Analogous to [NearestAirportCurrentLine](#)

**NearestAirportSelectedAirportLatitude**

**NearestAirportSelectedAirportLongitude (degrees, radians) [Get]**

Coordinates of the nearest selected airport. Same values as [WaypointAirportLatitude](#) and [Longitude](#).

**NearestAirportSelectedLatitude**

**NearestAirportSelectedLongitude (degrees, radians) [Get]**

Redundant. Identical to [NearestAirportSelectedAirportLatitude](#) and [Longitude](#).

**NearestAirportSelectedAirportName (string) [Get]**

Name of the nearest selected airport. Same as [WaypointAirportName](#).

**NearestAirportSelectedAirportCity (string) [Get]**

City name of the nearest selected airport. Same as [WaypointAirportCity](#).

**NearestAirportSelectedAirportElevation (feet) [Get]**

Airport elevation (asl) of the nearest selected airport.  
Same as [WaypointAirportElevation](#).

## NearestAirport Frequency Variables

These variables list certain frequency information for a selected nearest airport. A specific nearest airport must be first selected by using its index, for example:

```
4 (>@c:NearestAirportSelected)
```

Following that, an indexed list of frequencies associated with that nearest airport can be retrieved. In this case, the 5<sup>th</sup> airport in the nearest airport list.

### ✘ NearestAirportCurrentFrequency (enum) [Get, Set]

Index for [NearestAirportCurrentFrequencyName](#).

### ✘ NearestAirportSelectedFrequencyIndex (enum) [Get, Set]

Index for [NearestAirportSelectedFrequencyValue](#).

Note the two different index pointers that must be used to obtain frequency data in the nearest airport search. Both are analogous to [WaypointAirportCurrentFrequency](#).

### ✘ NearestAirportSelectedNumberFrequencies (enum) [Get]

Number of frequencies at the selected nearest airport.  
Same as [WaypointAirportFrequenciesNumber](#).

### ✘ NearestAirportCurrentFrequencyName (string) [Get]

Names of the airport frequencies. Communication frequency names include:

- Approach
- ATIS, ASOS, AWOS
- CTAF
- Unicom
- Multicom
- Clearance
- Clearance Pre-Taxi
- Ground
- Tower
- Departure
- FSS
- Remote Clearance Delivery

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

Same as [WaypointAirportFrequencyName](#).

This variable is not the same as [NearestAirportCurrentComFrequencyName](#) which is the name of the airport traffic control only, if one is present at the selected airport.

✘ **NearestAirportSelectedFrequencyValue (MHz) [Get]**

Radio frequency value, usually expressed an MHz.  
Same as [WaypointAirportFrequencyValue](#).

### **NearestAirport Runway Variables**

These variables list certain runway information for a selected nearest airport. A specific nearest airport must be first selected by using its index, for example:

```
4 (>@c:NearestAirportSelected)
```

Following that, an indexed list of runways associated with that nearest airport can be retrieved. In this case, the 5<sup>th</sup> airport in the nearest airport list.

✘ **NearestAirportSelectedRunway (enum) [Get, Set]**

Index pointer. Analogous to [WaypointAirportCurrentRunway](#).

✘ **NearestAirportSelectedAirportRunwaysNumber (enum) [Get]**

Number of runways at the selected nearest airport.  
Same as [WaypointAirportRunwaysNumber](#).

✘ **NearestAirportSelectedRunwayDesignation (string) [Get]**

The designation, or name, of the selected runway. For example, "04L-22-R". Same as [WaypointAirportRunwayDesignation](#).

✘ **NearestAirportSelectedRunwayLength (feet) [Get]**

Length of the selected runway. Same as [WaypointAirportRunwayLength](#).

✘ **NearestAirportSelectedRunwayWidth (feet) [Get]**

Width of the selected runway. Same as [WaypointAirportRunwayWidth](#).

✘ **NearestAirportSelectedRunwaySurface (R.S.T. enum) [Get]**

A number representing runway surface type.  
Same as [WaypointAirportRunwaySurface](#).

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

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

### NearestAirport Approach Variables

These variables list the approach names for the selected nearest airport. A specific nearest airport must be first selected by using its index, for example:

```
4 (>@c:NearestAirportSelected)
```

Following that, an indexed list of approaches associated with that nearest airport can be retrieved. In this case, the 5<sup>th</sup> airport in the nearest airport list. All approaches for all runways are listed.

#### **NearestAirportCurrentApproach (enum) [Get, Set]**

Index pointer. Analogous to [WaypointAirportCurrentApproach](#).

#### **NearestAirportSelectedApproachIndex (enum) [Get, Set]**

This appears to be an index pointer, and its value can be Set. What it's an index pointer to, I don't know.

#### **NearestAirportSelectedNumberApproaches (enum) [Get]**

The number of approach procedures for the selected nearest airport. Same as [WaypointAirportApproachesNumber](#).

#### **NearestAirportCurrentApproachName (string) [Get]**

The name of the selected approach, such as "ILS 22R", "NDB 27R", "RNAV 09L". Same as [WaypointAirportApproachName](#).

## 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**
- ❑ **NearestIntersectionCurrentLongitude (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.

- ❑ **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 [NearestIntersection](#) 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	Intersection Type	Bit	Intersection Type
0	UNKNOWN	4	NDB
1	NAMED	5	OFFROUTE
2	UNNAMED	6	IAF
3	VOR	7	FAF

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

The default fs9gps [NearestIntersectionCurrentFilter](#) value is 230. The binary equivalent of decimal 230 is 1 1 1 0 0 1 1 0.

### INTERSECTION TYPES

128	64	32	16	8	4	2	1	- Decimal value
		OFFROUTE			UNNAMED	NAMED	UNKNOWN	- INTERSECTION TYPE
FAF	IAF		NDB	VOR				
7	6	5	4	3	2	1	0	- Bit number (0 thru 7)
<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	- <b>Bit selections</b>

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 start, this is an example search result using the default filter setting (that is, no filter or intersection type is set):

```
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      WK3KCNFLOUR  FLOUR   1    0.0  187
1      WK3KNUMACEZ  MACEZ   1    0.3  331
2      WK3  BABGE   BABGE   1    3.4  162
3      WK3KNUMYYER  MYYER   1    5.0  187
4      WK3  WEITT   WEITT   1    6.3  323
5      WK3K88 HEKBO  HEKBO   1    8.1   26
6      WK3KPPFAGNEW  AGNEW   1    9.5  186
7      WK3K88 25THR  25THR   2   10.5   23
8      WK3KPPFF17   FF17    2   14.5  184
9      WK3KPPFDENUM  DENUM   1   16.0  182
```

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 used for cross-country navigation purposes and often part of 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.

### EXAMPLE 1: NearestIntersectionAddIntersectionType

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

```
3 (>C:fs9gps:NearestIntersectionAddIntersectionType)
```

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     WK3KNUFLOUR  FLOUR   1    0.0  187
1     WK3KNUMACEZ  MACEZ   1    0.3  331
2     WK3  BABGE   BABGE   1    3.4  162
3     WK3KNUMYYER  MYYER   1    5.0  187
4     VK3  CNU     CNU     3    5.5  247
5     WK3  WEITT   WEITT   1    6.3  323
6     WK3K88 HEKBO  HEKBO   1    8.1   26
7     WK3KPPFAGNEW  AGNEW   1    9.5  186
8     WK3K88 25THR  25THR   2   10.5  23
9     WK3KPPFFF17  FF17    2   14.5  184
```

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

## EXAMPLE 2: NearestIntersectionRemoveIntersectionType

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

```
3 (>C:fs9gps:NearestIntersectionAddIntersectionType)
1 (>C:fs9gps:NearestIntersectionRemoveIntersectionType)
```

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      WK3K88  25THR    25THR  2   10.5  23
2      WK3KPPFF  FF17     FF17   2   14.5  184
3      WK3KPPFMA  MA118    MA118  2   19.7  183
4      WK3KPPF22  22THR    22THR  2   22.6  182
5      WK3KPPFFF  FF35     FF35   2   25.4  182
6      WK3KJLNJL  JLN31    JLN31  2   27.6  106
7      VK3      OSW      OSW    3   33.2  156
8      WK3KUKLFF  FF36     FF36   2   34.7  339
9      WK3KPTSFF  FF166    FF166  2   36.7  104
```

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:

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

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     WK3KNUFLOUR  FLOUR  1    0.0  187
1     WK3KNUMACEZ  MACEZ  1    0.3  331
2     WK3  BABGE   BABGE  1    3.4  162
3     WK3KNUMYYER  MYYER  1    5.0  187
4     WK3  WEITT   WEITT  1    6.3  323
5     WK3K88 HEKBO  HEKBO  1    8.1  26
6     WK3KPPFAGNEW  AGNEW  1    9.5  186
7     WK3K88 25THR  25THR  2   10.5  23
8     WK3KPPFFF17  FF17   2   14.5  184
9     WK3KPPFDENUM  DENUM  1   16.0  182
```

❑ **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.

## FSX-ONLY VARIABLES – NearestIntersection Group

### ☒ **NearestIntersectionSelectedIntersection (enum) [Get, Set]**

The Index Pointer. Analogous to [NearestIntersectionCurrentLine](#). The variable that counts the number of nearest intersections is [NearestIntersectionItemsNumber](#) (above)

### ☒ **NearestIntersectionSelectedIntLatitude**

### ☒ **NearestIntersectionSelectedIntLongitude (degrees) [Get]**

Latitude and Longitude of the Selected Intersection. These are the same coordinates as [WaypointIntersectionLatitude](#) and [Longitude](#).

### ☒ **NearestIntersectionSelectedRefVorId (string) [Get]**

### ☒ **NearestIntersectionSelectedRefVorType (VOR Type enum) [Get]**

### ☒ **NearestIntersectionSelectedRefVorFrequency (MHz) [Get]**

### ☒ **NearestIntersectionSelectedRefVorTrueRadial (degrees) [Get]**

### ☒ **NearestIntersectionSelectedRefVorMagneticRadial (degrees) [Get]**

### ☒ **NearestIntersectionSelectedRefVorDistance (nmiles) [Get]**

As far as I can tell, the [SelectedRefVor](#) variables are not functional.

There is reason to suspect that [RefVor](#) refers to, or should refer to, the VOR closest to the intersection\* consistent with the existing [WaypointIntersectionNearestVor](#) variables. If true, the [NearestIntersectionSelectedRefVor](#) variables should return values when indexed using [NearestIntersectionCurrentLine](#), but they do not.

\* the intersection defined by [WaypointIntersectionIcao](#)

## 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) [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](#).

- ❑ **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 12 character 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.

❑ **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	VOR Type	Bit	VOR Type
0	UNKNOWN	4	TACAN
1	VOR	5	VORTAC
2	VOR_DME	6	ILS
3	DME	7	VOT

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

The assumption is that the VOR types for FS9 are:

Bit	VOR Type	Bit	VOR Type
0	UNKNOWN	3	DME
1	VOR	4	TACAN
2	VOR_DME	5	VORTAC

The default fs9gps [NearestVorCurrentFilter](#) value is 62. The binary equivalent is 1 1 1 1 1 0.

**VOR TYPES**

32	16	8	4	2	1	- Decimal value
VORTAC	TACAN	DME	VOR_DME	VOR	UNKNOWN	- VOR TYPE
5	4	3	2	1	0	- Bit number (0 thru 5)
<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	- <b>Bit selections</b>

which includes all VOR types except UNKNOWN.

To include only VOR types 1, 2, and 3 in the [NearestVor](#) search, the binary would be 0 0 1 1 1 0.

**VOR TYPES**

32	16	8	4	2	1	- Decimal value
VORTAC	TACAN	DME	VOR_DME	VOR	UNKNOWN	- VOR TYPE
5	4	3	2	1	0	- Bit number (0 thru 5)
<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	- <b>Bit selections</b>

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.

❑ **NearestVorAddVorType (enum) [Set]**

[NearestVorAddVorType](#) is an enum (not a binary) adds a VOR type to the [NearestVor](#) search. If the following is used

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

then only VOR types 1 and 3 (decimal 10 = binary 0 0 1 0 1 0) 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
 -88.9090 :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 a [NearestVorAddVorType](#) instruction is added,

```
10 (>C:fs9gps:NearestVorCurrentFilter)
 2 (>C:fs9gps:NearestVorAddVorType)
```

the [NearestVor](#) search changes:

## NEAREST VOR SEARCH

42.9638 :Current Lat 20 :Max Items 20 :Items Num  
-88.9090 :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 0 0 1 1 1 0, 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:

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

and the search result becomes:

## NEAREST VOR SEARCH

42.9638 :Current Lat 20 :Max Items 11 :Items Num  
-88.9090 :Current Lon 300 :Max Dist 58 :Filter

```
----- NearestVorCurrent -----
```

Line	ICAO	111	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 1 1 1 0 1 0 (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 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:

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

sets the [CurrentFilter](#) to 14 (binary 0 0 1 1 1 0) and VOR types 1, 2, and 3 are included in the [NearestVor](#) search.

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

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

removes VOR Type 2, [CurrentFilter](#) becomes 10 (binary 0 0 1 0 1 0), and VOR types 1 and 3 are included in the [NearestVor](#) search.

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

resets [CurrentFilter](#) to 62 (binary 1 1 1 1 1 0) 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.

## **FSX-ONLY VARIABLES – NearestVor Group**

These FSX-only variables add to, or augment the available FS9 NearestVor search variables except for [SelectedVorType](#) and [SelectedVorFrequency](#) which are duplicates of the FS9 counterparts.

### **NearestVorSelectedVor (enum) [Get, Set]**

The Index pointer. Analogous to [NearestVorCurrentLine](#).

### **NearestVorSelectedVorLatitude**

### **NearestVorSelectedVorLongitude (degrees) [Get]**

Latitude and Longitude of the selected VOR. Same coordinates as [WaypointVorLatitude](#) and [Longitude](#).

✘ **NearestVorSelectedVorName (string) [Get]**

Name of the selected VOR. Same string as [WaypointVorName](#).

⊘ **NearestVorSelectedVorCity (string) [Get]**

Returns a blank string. [WaypointVorCity](#) appears to not be populated in the gps database.

✘ **NearestVorSelectedVorType (VOR Type enum) [Get]**

The type of each VOR. Same as [NearestVorCurrentType](#) described above.

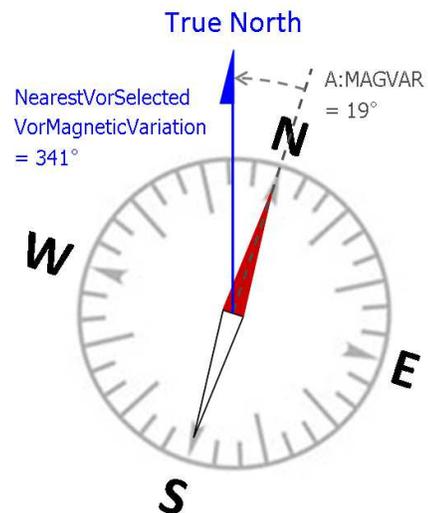
✘ **NearestVorSelectedVorFrequency (MHz) [Get]**

The frequency of each VOR. Same as [NearestVorCurrentFrequency](#) described above.

✘ **NearestVorSelectedVorMagneticVariation (degrees) [Get]**

[NearestVorSelectedVorMagneticVariation](#) is the compass direction of true north.

In this example, A:MAGVAR and A:GPS MAGVAR would equal +15° (15E).



## 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**
- ❑ **NearestNdbCurrentLongitude (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.

- ❑ **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 [NearestNdb](#) 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 fs9gps):

- 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    11 :Items Num
-0.0070 :Curr Lon    50 :Max Distance

----- NearestNdbCurrent -----
      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     NEGETCCIT  CIT    2    850  41.7  209
8     NEG      EME    EME    2    354  43.5  278
9     NEG      NWI    NWI    1    343  47.4   94
10    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.

#### **NearestNdbCurrentFrequency (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.

## FSX-ONLY VARIABLES – NearestNdb Group

These FSX-only variables add to, or augment the available FS9 NearestNdb search variables except for [SelectedNdbType](#) and [SelectedNdbFrequency](#) which are duplicates of the FS9 counterparts.

### **NearestNdbSelectedNdb (enum) [Get, Set]**

The Index pointer. Analogous to [NearestNdbCurrentLine](#).

### **NearestNdbSelectedNdbLatitude**

### **NearestNdbSelectedNdbLongitude (degrees) [Get]**

Latitude and Longitude of the selected NDB. Same coordinates as [WaypointNdbLatitude](#) and [Longitude](#).

### **NearestNdbSelectedNdbName (string) [Get]**

Name of the selected NDB. Same string as [WaypointNdbName](#).

### **NearestNdbSelectedNdbCity (string) [Get]**

Returns a blank string. [WaypointNdbCity](#) appears to not be populated in the gps database.

### **NearestNdbSelectedNdbType (NDB Type enum) [Get]**

The type of each NDB. Same as [NearestNdbCurrentType](#) described above.

### **NearestNdbSelectedNdbFrequency (KHz) [Get]**

The frequency of each NDB. Same as [NearestNdbCurrentFrequency](#) described above.

## 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 VOR 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**
- ❑ **NearestAirspaceCurrentLongitude (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, degrees)
(>@c:NearestAirspaceCurrentLatitude, degrees)
```

```
(A:PLANE LONGITUDE, degrees)
(>@c:NearestAirspaceCurrentLongitude, degrees)
```

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 1/60<sup>th</sup> 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:PLANE ALTITUDE, feet)
(>@c:NearestAirspaceCurrentAltitude, feet)
```

#### ❑ **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 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, a parameter named **kDisplayedAirspaces** is assigned the hex value **0xEFC038**. Converting this hex number to binary yields **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:

Bit	Airspace Type	Bit	Airspace Type	Bit	Airspace Type
0	NONE	8	CLASS_G	16	PROHIBITED
1	CENTER	9	TOWER	17	WARNING
2	CLASS_A	10	CLEARANCE	18	ALERT
3	CLASS_B	11	GROUND	19	DANGER
4	CLASS_C	12	DEPARTURE	20	NATIONAL_PARK
5	CLASS_D	13	APPROACH	21	MODE_C
6	CLASS_E	14	MOA	22	RADAR2
7	CLASS_F	15	RESTRICTED	23	TRAINING

[http://msdn.microsoft.com/en-us/library/cc526954.aspx#FAC\\_BV\\_TYPE](http://msdn.microsoft.com/en-us/library/cc526954.aspx#FAC_BV_TYPE)

This binary number displayed in a bit table:

**AIRSPACE BIT TABLE**

8,388,608	4,194,304	2,097,152	1,048,576	524,288	262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1	- Decimal value
TRAINING	RADAR	MODE C	NATIONAL PARK	DANGER	ALERT	WARNING	PROHIBITED	RESTRICTED	MOA	APPROACH	DEPARTURE	GROUND	CLEARANCE	TOWER	CLASS G	CLASS F	CLASS E	CLASS D	CLASS C	CLASS B	CLASS A	CENTER	NONE	- AIRSPACE TYPE
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	- Bit number (0 thru 23)
1	1	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	- Bit selections

Reading 1110 1111 1100 0000 0011 1000 from *right to left* as shown in the bit table above (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 list, 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:

**AIRSPACE BIT TABLE**

	8,388,608	4,194,304	2,097,152	1,048,576	524,288	262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1	- Decimal value
TRAINING	RADAR	MODE C	NATIONAL PARK	DANGER	ALERT	WARNING	PROHIBITED	RESTRICTED	MOA	APPROACH	DEPARTURE	GROUND	CLEARANCE	TOWER	CLASS G	CLASS F	CLASS E	CLASS D	CLASS C	CLASS B	CLASS A	CENTER	NONE	- AIRSPACE TYPE	
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	- Bit number (0 thru 23)	
0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	- Bit selections

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:

**AIRSPACE BIT TABLE**

	8,388,608	4,194,304	2,097,152	1,048,576	524,288	262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1	- Decimal value
TRAINING	RADAR	MODE C	NATIONAL PARK	DANGER	ALERT	WARNING	PROHIBITED	RESTRICTED	MOA	APPROACH	DEPARTURE	GROUND	CLEARANCE	TOWER	CLASS G	CLASS F	CLASS E	CLASS D	CLASS C	CLASS B	CLASS A	CENTER	NONE	- AIRSPACE TYPE	
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	- Bit number (0 thru 23)	
1	1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	- Bit selections

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

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

It is also acceptable to enter the decimal equivalent instead, although 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, VORs, 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]**

[AirspaceCurrentType](#) is 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 \(enum: 0, 1, 2, 3, or 4\) \[Get\]](#)**

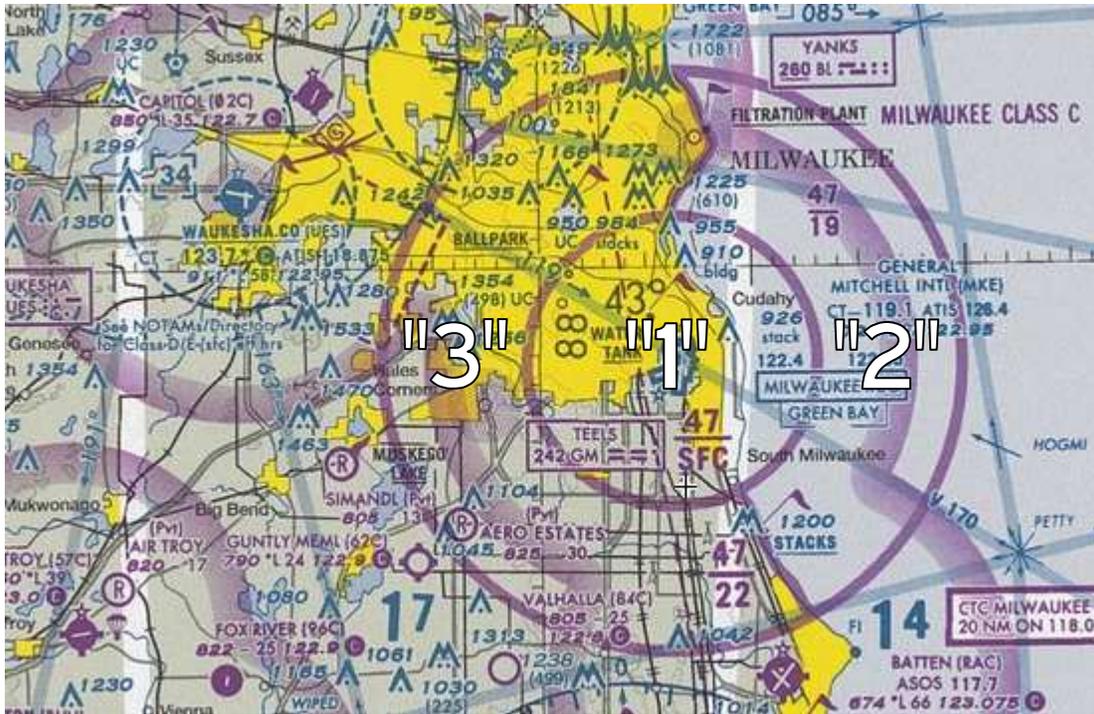
[NearestAirspaceCurrentStatus](#) is a number that reflects an aircraft's positional status relative to nearby airspaces. It is determined from the "closeness" of the aircraft to airspace sectors in either x, y, z space or time. [CurrentStatus](#), in turn, controls [MessageItemsNumber](#) and [MessageCurrentType](#) 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.

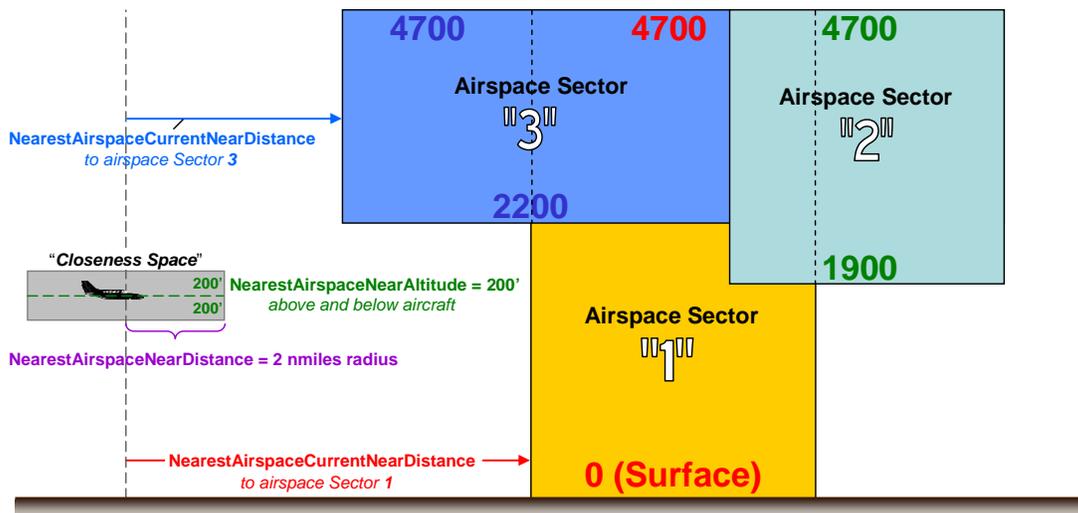
In addition to x,y,z position, enroute time "closeness" also controls, or triggers, [CurrentStatus](#) changes.

**EXAMPLE AIRSPACE STATUS and MESSAGES**

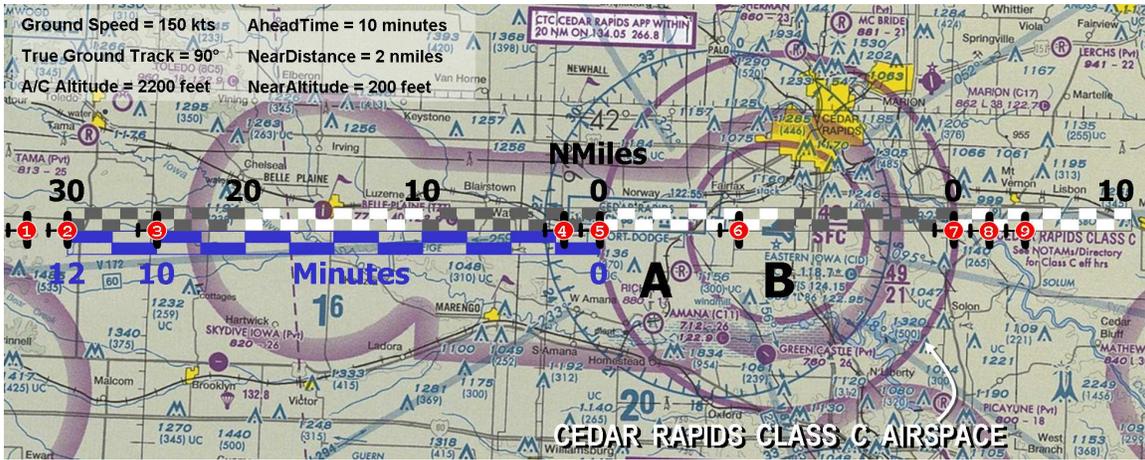
**MILWAUKEE CLASS C AIRSPACE**



**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	$CurrentAheadTime > 1.20 \times AheadTime$ Aircraft too far from Airspace Sector A to display in Search	0 = NONE	none
2	0	Toward	Time	$CurrentAheadTime \leq 1.20 \times AheadTime$ Aircraft appears in <b>NearestAirspace</b> search display	0 = NONE	none
3	2	Toward	Time	$CurrentAheadTime \leq 1.00 \times AheadTime$	2 = AHEAD	Airspace ahead -- less than 10 minutes
4	3	Toward	Distance	$CurrentNearDistance \leq 1.00 \times NearDistance$	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 $CurrentNearDistance$ counts upward from 0.00	1 = NEAR	none
8	0	Away	Distance	$CurrentNearDistance \geq 1.00 \times NearDistance$	0 = NONE	none
9	none	N/A	Distance	$CurrentNearDistance \geq 2.00 \times NearDistance$ 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

```

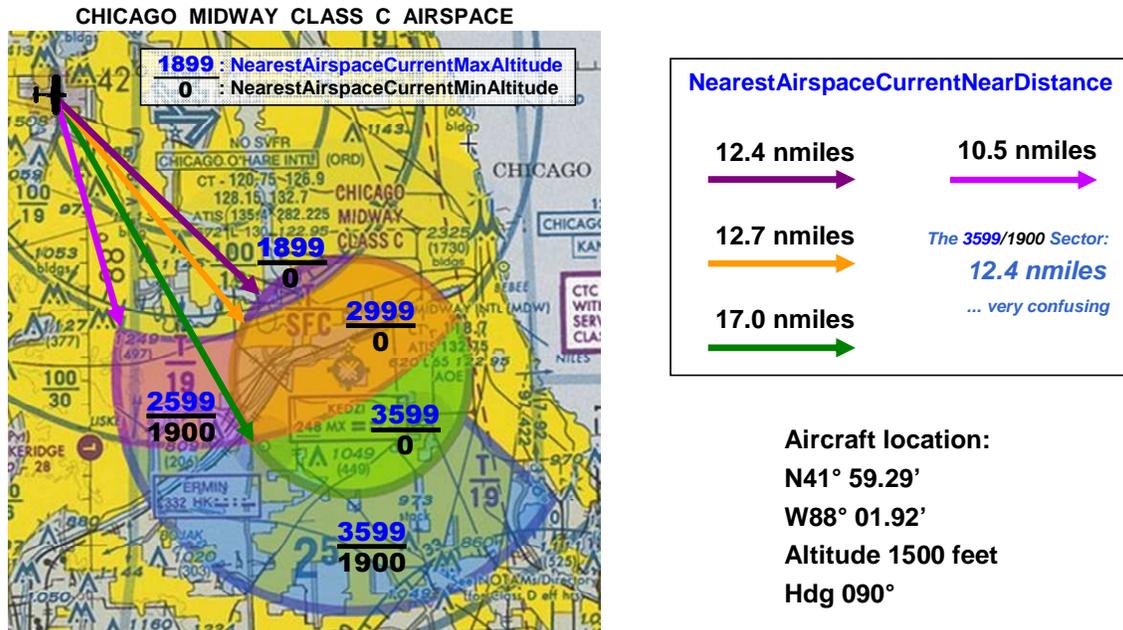
**Point 9** - **CurrentNearDistance** = 2.0 = 2.00 x **NearDistance**. At 2 times **NearDistance**, the airspace sector is dropped out of the **NearestAirspace** search display. The 2.00 factor appears to be hard coded into the **gps** module.

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.



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 [CurrentNearDistance](#) 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 the 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.

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

Only 'A', 'V', 'N', 'X', or 'W' may be used in [IcaoSearchStartCursor](#).

Combinations of the letters can also be entered. 'AVNW' will enable a search of all types of ICAOs. However, duplicate ICAOs will be returned for a facility that serves a double role as Navaid ('V' or 'N') and 'W'aypoint. For example, 'VNW' [StartCursor](#) will yield double (two) **VK3** \_ \_ \_ \_ **CVA** ICAOs for a 'CVA' [EnterChar](#).

**StartCursor FACILITY**

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

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. XML Script (code) Entry.** The users XML gauge code may enter data into the [IcaoSearchEnterChar](#) variable:

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

For the variable [StartCursor](#), code entry is always used, not keyboard or mouse entry.

```
(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	<a href="#">EnterChar</a>	<a href="#">CurrentIdent</a>	<a href="#">String Length</a>
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

	<b>Cursor Advance</b>	<b>Concatenation</b>
<b>Keyboard Direct Entry</b>	Automatic	Automatic
<b>Mouse Entry</b>	by Mouse	Automatic
<b>Code Entry</b>	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

ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890

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 Identifiers 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](#):

<u>IcaoType</u>	<u>Facility</u>
A	Airport
V	VOR, ILS, LOC
N	VOR
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 IDENT 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 38).

## 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.

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

## **DATA ENTRY METHODS FOR NameSearch**

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

- 1. 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>
```

- 2. Mouse Entry.** 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>
```

### 3. XML Script (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. If more than 80 characters are entered, then the previous, the last, entry is replaced.

	Cursor Advance	Concatenation
<b>Keyboard Direct Entry</b>	Automatic	Automatic
<b>Mouse Entry</b>	by Mouse	Automatic
<b>Code Entry</b>	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]**

[NameSearchCurrentName](#) 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.

#	Route Type	#	Route Type
0	DIRECT	2	LOWALT
1	VOR	3	HIGHALT

[http://msdn.microsoft.com/en-us/library/cc526954.aspx#ATC\\_RouteType](http://msdn.microsoft.com/en-us/library/cc526954.aspx#ATC_RouteType)

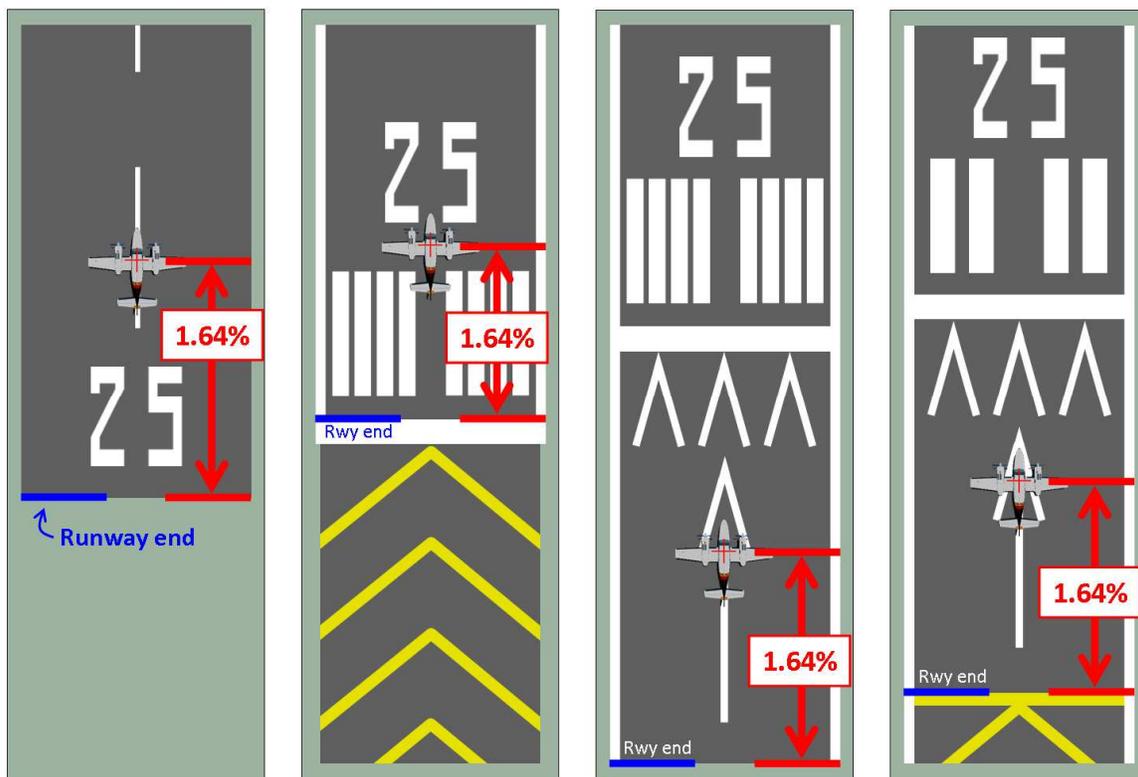
❑ **FlightPlanCruisingAltitude (feet) [Get]**

A general discussion can be found at the end of this section.

❑ **FlightPlanDepartureLatitude**

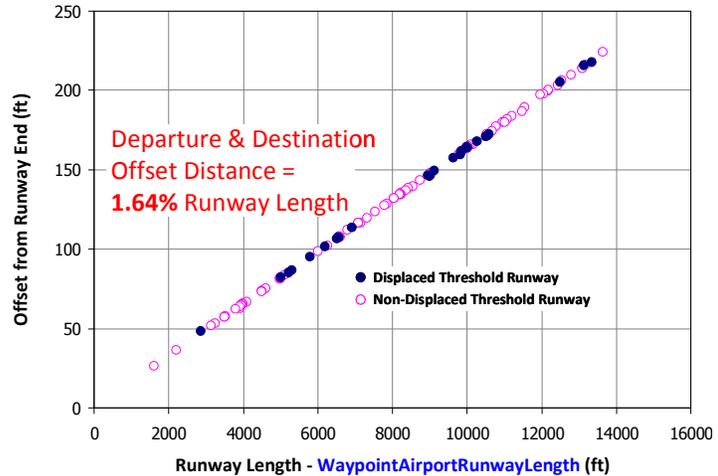
❑ **FlightPlanDepartureLongitude (degrees, radians) [Get]**

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, each representing different coordinates. It is fixed at the starting point and does not change as the aircraft taxis for takeoff. If FS9/X Flight Planner is used to position the aircraft on the Active Runway, then as shown in the figure below, the aircraft will be placed at a point offset from the end of the runway 1.64% of runway length (WaypointAirportRunwayLength).



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

(Only 100+ runways around the world were spot checked, but the offset appears to be a consistent 1.64% of airport runway length)



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

The 3 to 4 character Ident of the Departure Airport.

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

Name of the Departure Airport.

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

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

❑ **FlightPlanDestinationLatitude**

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

Latitude and Longitude of a point offset from the approach end of the active runway. The landing (Destination) offset distance is, similarly, 1.64% of runway length. It appears that for the sake of simplicity in the fs9gps world, the departure and destination points of the active runway are coincident. This is true even in the case of displaced thresholds, where in real-world aviation, landing before the threshold is not permitted.



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

The 3 to 4 character Destination Airport Ident.

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

Destination airport Name.

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

Altitude (asl) of the destination point.

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.

<b>fs9gps variable</b>	<b>FS9 Flight Planner (File.pln)</b>
	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 [1]	8 departure_id=VDPP, N11* 32.09', E104* 50.27', +000040.00
	9 departure_position=5
See id detail below [2]	10 destination_id=WSSS, N1* 19.75', E103* 59.11', +000021.99
FlightPlanDepartureName	11 departure_name=Pochentong Intl
FlightPlanDestinationName	12 destination_name=Changi
FlightPlanWaypoint Info	13 waypoint.0=, VDPP, , VDPP, A, N11* 32.09', E104* 50.27', +000040.00,
FlightPlanWaypoint Info	14 waypoint.1=, WSSS, , WSSS, A, N1* 19.75', E103* 59.11', +000021.99,
	[1] departure_id=VDPP, N11* 32.09', E104* 50.27', +000040.00
FlightPlanDepartureAirportIdent	VDPP
FlightPlanDepartureLatitude	N11* 32.09'
FlightPlanDepartureLongitude	E104* 50.27'
FlightPlanDepartureAltitude	+000040.00
	[2] destination_id=WSSS, N1* 19.75', E103* 59.11', +000021.99
FlightPlanDestinationAirportIdent	WSSS
FlightPlanDestinationLatitude	N1* 19.75'
FlightPlanDestinationLongitude	E103* 59.11'
FlightPlanDestinationAltitude	+000021.99

- ❑ **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 that does this exists, I don't know) to define the Alternate using these variables.

I've edited lines 13 and 14 to the .PLN file to add an 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.

<b>fs9gps variable</b>	<b>FS9 Flight Planner (File.pln)</b>
	1 [flightplan]
	2 AppVersion=9.1.40901
<a href="#">FlightPlanTitle</a>	3 title=VDPP to WSSS
<a href="#">FlightPlanDescription</a>	4 description=VDPP, WSSS
<a href="#">FlightPlanFlightPlanType</a>	5 type=IFR
<a href="#">FlightPlanRouteType</a>	6 routetype=0
<a href="#">FlightPlanCruisingAltitude</a>	7 cruising_altitude=34000
	8 departure_id=VDPP, N11* 32.09', E104* 50.27', +000040.00
	9 departure_position=5
	10 destination_id=WSSS, N1* 19.75', E103* 59.11', +000021.99
<a href="#">FlightPlanDepartureName</a>	11 departure_name=Pochentong Intl
<a href="#">FlightPlanDestinationName</a>	12 destination_name=Changi
<a href="#">See id detail below [3]</a>	13 alternate_id=WIKB, N1* 07.13', E104* 06.85', +000126.99
<a href="#">FlightPlanAlternateName</a>	14 alternate_name=Hang Nadim
<a href="#">FlightPlanWaypoint Info</a>	15 waypoint.0=, VDPP, , VDPP, A, N11* 32.09', E104* 50.27', +000040.00,
<a href="#">FlightPlanWaypoint Info</a>	16 waypoint.1=, WSSS, , WSSS, A, N1* 19.75', E103* 59.11', +000021.99,
	[3] alternate_id=WIKB, N1* 07.13', E104* 06.85', +000126.99
<a href="#">FlightPlanAlternateAirportIdent</a>	WIKB
<a href="#">FlightPlanAlternateLatitude</a>	N1* 07.13'
<a href="#">FlightPlanAlternateLongitude</a>	E104* 06.85'
<a href="#">FlightPlanAlternateAltitude</a>	+000126.99

After lines 13 and 14 are added to the Flight Plan .pln file, then the [FlightPlanAlternate](#) variables are populated, but it requires editing the .pln file because [FlightPlanAlternate](#) variables are Get only.

If desired, an Airport ICAO can be 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. Once made into an ICAO, it can then be used to specify a new approach Airport, for example. 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, it may 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 the simulator running, storing the Ident as chr >L:Vars or XMLVars for later use if needed.

## 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:

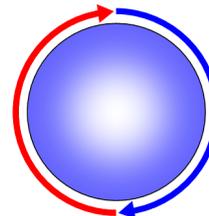
- U.S. F.A.R. Part 91 Cruising Altitude rules are applied world-wide. Above 18,000 feet altitude where, in U.S.A. airspace, flight is governed by Instrument Flight Rules, FS9 Flight Planner still adds 500 feet to the cruising altitude of a flight that is set up as a VFR flight.

### F.A.R. Part 91 Cruising Altitude Rules

**180° - 359°**

☐ VFR: Even thousands of feet + 500 ft

☐ IFR: Even thousands of feet or Flight Levels



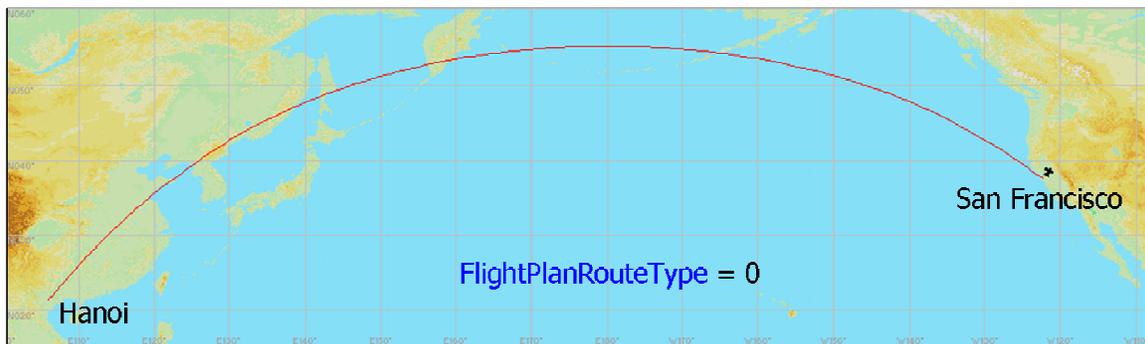
**360° - 179°**

☐ VFR: Odd thousands of feet + 500 ft

☐ IFR: Odd thousands of feet or Flight Levels

- 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.
- Flight Planner calculates GPS-Direct routes along Great Circle paths.



In summary, I am guessing as to FS9 Flight Planner's rules for computing [CruisingAltitude](#). However, after consulting your charts, [FlightPlanCruisingAltitude](#) is manually settable from within FS9 Flight Planner and can be changed during flight by request to ATC.

❑ **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, SIDS and STARS can be created and edited using fs9gps variables, and with the help of the LOGGER module, saved to and read from hard drive. LOGGER is available for download free of charge from

<http://robbiemcelrath.com/blackbox/?logger>.

The LOGGER module is also included as a class in Tom Aguilo's XMLTools XML expansion module. XMLTools can be freely downloaded from

<http://fsdeveloper.com/forum/resources/xmltools-2-0-xml-expansion-module-for-fsx.148/>

and is highly recommended for all serious XML gauge programmers.

[FlightPlanDirectToDestination](#) is used to create a new Flight Plan when one is not currently loaded. Use of the [FlightPlanDirectToDestination](#) variable is reviewed later in this section.

### **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. Entering Latitude and Longitude will create a new Type 5 User waypoint that will not have an associated ICAO. Entering the ICAO **VED** \_ \_ \_ **BAM** will create a new Type 3 VOR waypoint.

### **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.

[FlightPlanWaypointIdent](#) can accommodate a string length up to 10. Although Idents with slen greater than 5 do not exist in the fs9gps database, it is possible to create a Type 5 User Waypoint and assign an Ident name up to 10 characters long by through use of [FlightPlanNewWaypointIdent](#), for example:

```
'ABC1234567' (>@c:FlightPlanNewWaypointIdent)
```

A User defined Ident such as this will not permanently update the fs9gps database nor is it searchable by ICAO Search.

## 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)`

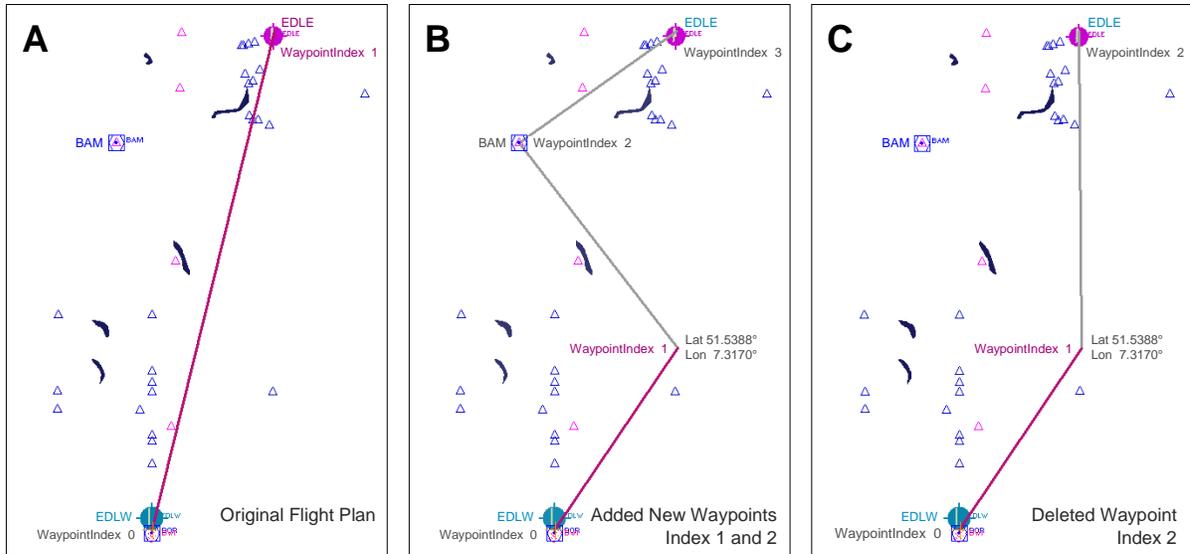
will add a new waypoint at [WaypointIndex](#) 2 with whatever Lat and Lon had previously been specified using [NewWaypointLatitude](#) and [Longitude](#), or [NewWaypointICAO](#). The waypoint previously at [WaypointIndex](#) 2 is advanced to become [WaypointIndex](#) 3. 3 becomes 4, and so forth. New 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

FlightPlanAddWaypoint and FlightPlanDeleteWaypoint are demonstrated in the following examples:



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

FLIGHT PLAN: EDLW to EDLE

```
2 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType 2 :FlightPlanFlightPlanType
```

----- FlightPlanWaypoint -----														
Idx	ICAO	111	Ident	Alt	Type	Mag	Lat	Lon	Dist	Ttl	Rem	Dist	Ttl	ETE
0	A	EDLW	EDLW	424	1	0	51.52333	7.62633	0.0	0.0	27.1	0.0	0.0	0.0
1	A	EDLE	EDLE	424	1	256	51.40017	6.92983	27.1	27.1	0.0	27.1	27.1	7.5

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.

```

FLIGHT PLAN: EDLW to EDLE
 4 :FlightPlanWaypointsNumber  1 :FlightPlanActiveWaypoint
 0 :FlightPlanRouteType        2 :FlightPlanFlightPlanType
----- FlightPlanWaypoint -----
Idx  ICAO  111  Ident  Alt  Type  Mag  Lat  Lon  Dist  Ttl  Rem  Dist  Ttl  ETE
 0   A   EDLW  EDLW  424  1    0   51.52333  7.62633  0.0  0.0  35.5  0.0  0.0  0.0
 1   A   EDLW  EDLW  0    5   276  51.53880  7.31700  11.6  11.6  23.9  11.6  11.6  3.2
 2   VED  BAM   BAM   0    3   204  51.32776  7.17699  13.7  25.3  10.2  13.7  25.3  3.8
 3   A   EDLE  EDLE  424  1   297  51.40017  6.92983  10.2  35.5  0.0  10.2  35.5  2.8

```

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:

```

FLIGHT PLAN: EDLW to EDLE
 3 :FlightPlanWaypointsNumber  1 :FlightPlanActiveWaypoint
 0 :FlightPlanRouteType        2 :FlightPlanFlightPlanType
----- FlightPlanWaypoint -----
Idx  ICAO  111  Ident  Alt  Type  Mag  Lat  Lon  Dist  Ttl  Rem  Dist  Ttl  ETE
 0   A   EDLW  EDLW  424  1    0   51.52333  7.62633  0.0  0.0  28.3  0.0  0.0  0.0
 1   A   EDLW  EDLW  0    5   276  51.53880  7.31700  11.6  11.6  16.7  11.6  11.6  3.2
 2   A   EDLE  EDLE  424  1   242  51.40017  6.92983  16.7  28.3  0.0  16.7  28.3  4.6

```

## ❑ [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 [FlightPlanNewWaypointLongitude](#), 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)
```

**CAUTION:** When Flight Simulator's Flight Planner creates a Flight Plan, Flight Plan string variables [FlightPlanTitle](#), [Description](#), [DepartureAirportIdent](#), [DepartureAirportName](#), [DestinationAirportIdent](#) and [DestinationAirportName](#) are filled in. However, [FlightPlanDirectToDestination](#) creates a Flight Plan in which Flight Plan string variables other than [FlightPlanTitle](#) are blank. **A word of caution here** - string operations such as SLEN on a blank string creates an error that will cause Flight Simulator to crash to desktop. Arguably, it may be a bug in Flight Simulator that such an error causes the simulation to crash.

As an example, the following operations will cause the sim to crash if they are preceded 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.

Other operations *may* also cause the simulation to crash or hang. Suffice it to say that I have found that a simulation crash can occur following [FlightPlanDirectToDestination](#), so be on the alert, especially when including FlightPlan string variables in your code.

#### ❑ **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.

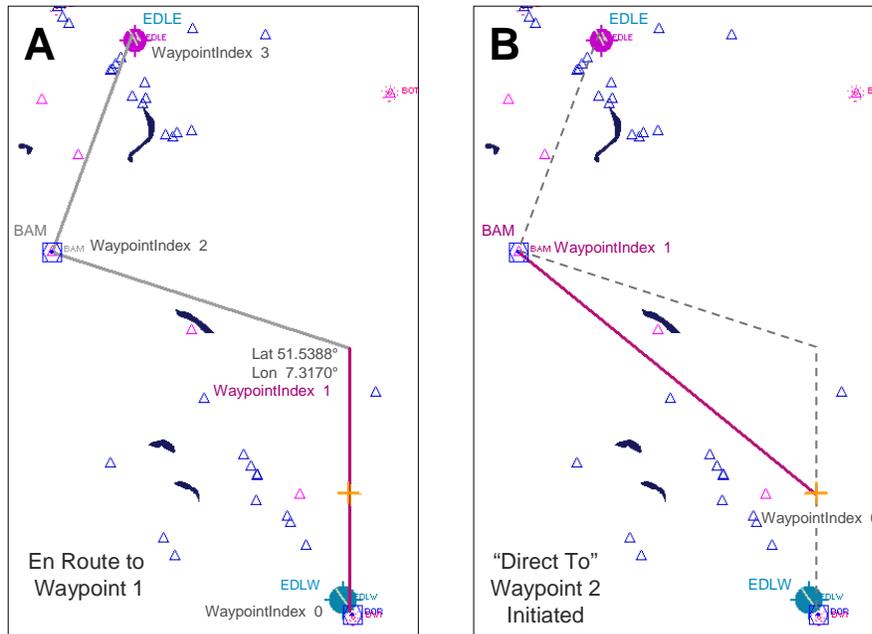
The xml:

```
(>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.

FLIGHT PLAN: EDLW to EDLE

4 :FlightPlanWaypointsNumber    1 :FlightPlanActiveWaypoint  
 0 :FlightPlanRouteType        2 :FlightPlanFlightPlanType

FlightPlanWaypoint														
Idx	ICAO	111	Ident	Alt	Type	Mag	Lat	Lon	Dist	Dist	Dist	Rem	Ttl	ETE
0	A	EDLW	EDLW	424	1	0	51.52333	7.62633	0.0	0.0	35.5	0.0	0.0	0.0
1				0	5	276	51.53880	7.31700	11.6	11.6	23.9	11.6	11.6	3.2
2	VED	BAM	BAM	0	3	204	51.32776	7.17699	13.7	25.3	10.2	13.7	25.3	3.8
3	A	EDLE	EDLE	424	1	297	51.40017	6.92983	10.2	35.5	0.0	10.2	35.5	2.8

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.

```
FLIGHT PLAN: Direct to BAM
 2 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
 0 :FlightPlanRouteType      0 :FlightPlanFlightPlanType
```

FlightPlanWaypoint														
Idx	ICAO	111	Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Ttl	Dist	Rem	Ttl	ETE
0				3049	5	0	51.53023	7.49048	0.0	0.0	16.9	0.0	0.0	0.0
1	VED	BAM	BAM	0	3	226	51.32776	7.17699	16.9	16.9	0.0	16.9	16.9	4.7

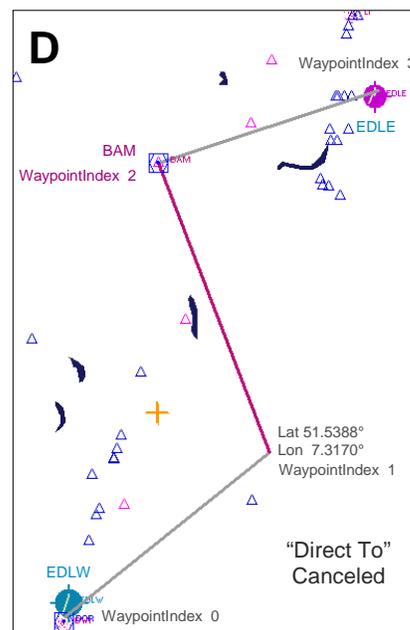
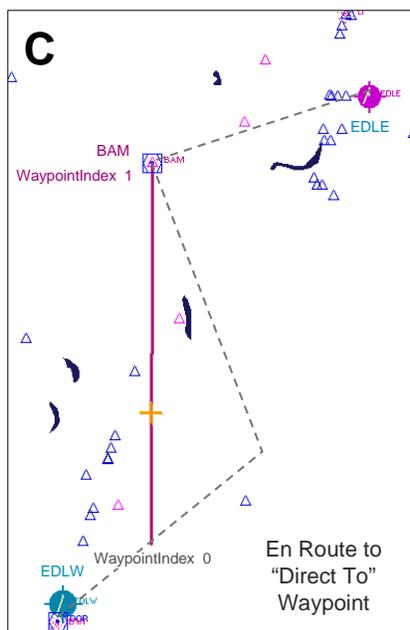
Map **C** shows progress of the flight en route to the **Direct To** waypoint. At this point, a **CancelDirectTo** is initiated. The xml:

```
(>@c:FlightPlanCancelDirectTo)
```

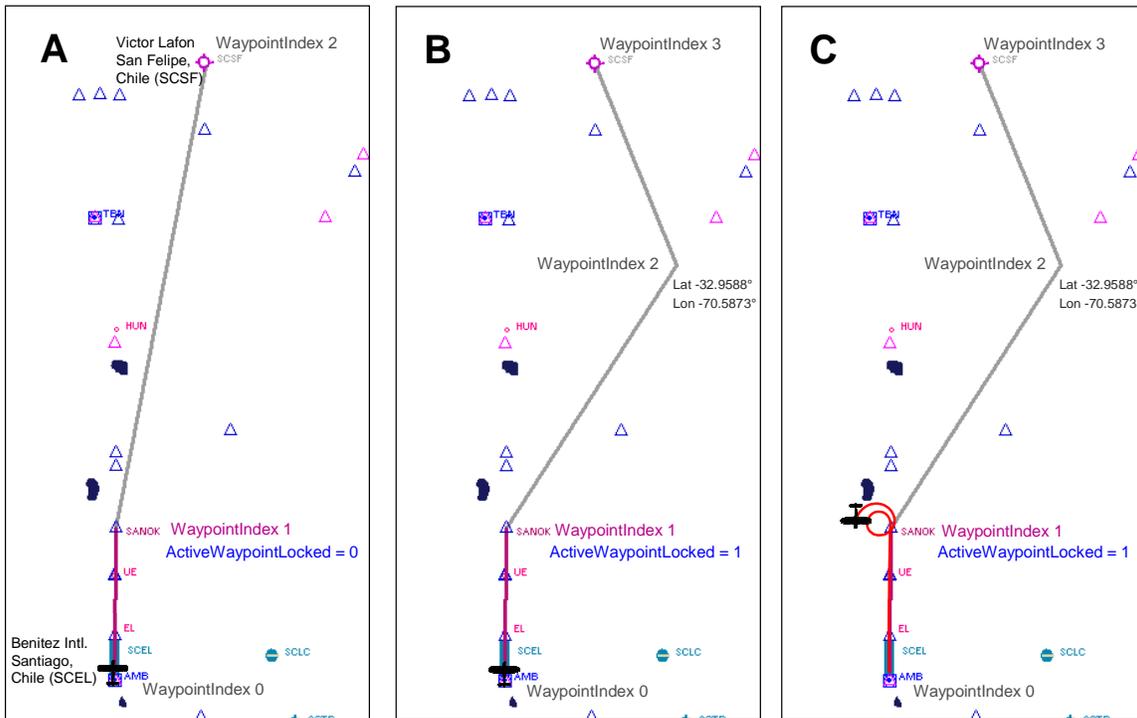
Map **D** reflects the Flight Plan immediately after **CancelDirectTo**, and the table below lists the **FlightPlanWaypoint** variables. Note that the original Flight Plan is restored and that distance and fuel variables associated with **ActiveWaypoint 2** are updated. The **ActiveWaypoint** is now 2, and, if on autopilot, the aircraft will begin a right turn to intercept the Waypoint 1 to Waypoint 2 leg.

```
FLIGHT PLAN: EDLW to EDLE
 4 :FlightPlanWaypointsNumber 2 :FlightPlanActiveWaypoint
 0 :FlightPlanRouteType      2 :FlightPlanFlightPlanType
```

FlightPlanWaypoint														
Idx	ICAO	111	Ident	Alt	Type	Mag Hdg	Lat	Lon	Dist	Ttl	Dist	Rem	Ttl	ETE
0	A	EDLW	EDLW	424	1	0	51.52333	7.62633	0.0	0.0	35.5	0.0	0.0	0.0
1				0	5	276	51.53880	7.31700	11.6	11.6	23.9	0.0	0.0	3.2
2	VED	BAM	BAM	0	3	204	51.32776	7.17699	13.7	25.3	10.2	5.2	5.2	3.8
3	A	EDLE	EDLE	424	1	297	51.40017	6.92983	10.2	35.5	0.0	10.2	15.5	2.8



### 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: SCEL to SCSF

```

3 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint 0 :FlightPlanIsActiveWaypointLocked
1 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 1 :FlightPlanIsActiveWaypoint
    
```

FlightPlanWaypoint													
Idx	ICAO	111	Ident	Alt	Type	Mag	Lat	Lon	Dist	Ttl	Rem	Dist	ETE
0	A	SCEL	SCEL	1554	1	0	-33.40933	-70.78483	0.0	0.0	39.9	0.0	0.0
1	WCS	SCELSANOK	SANOK	0	2	352	-33.25284	-70.79138	9.4	9.4	30.5	9.4	2.6
2	A	SCSF	SCSF	2160	1	3	-32.74933	-70.70200	30.5	39.9	0.0	30.5	8.5

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)
    
```

In FS9, executing [AddWaypoint](#) simultaneously locks the active waypoint causing [ActiveWaypointLocked](#) to return 1. In FSX, on the other hand, [AddWaypoint](#) does not lock the active waypoint.

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: SCEL to SCSF

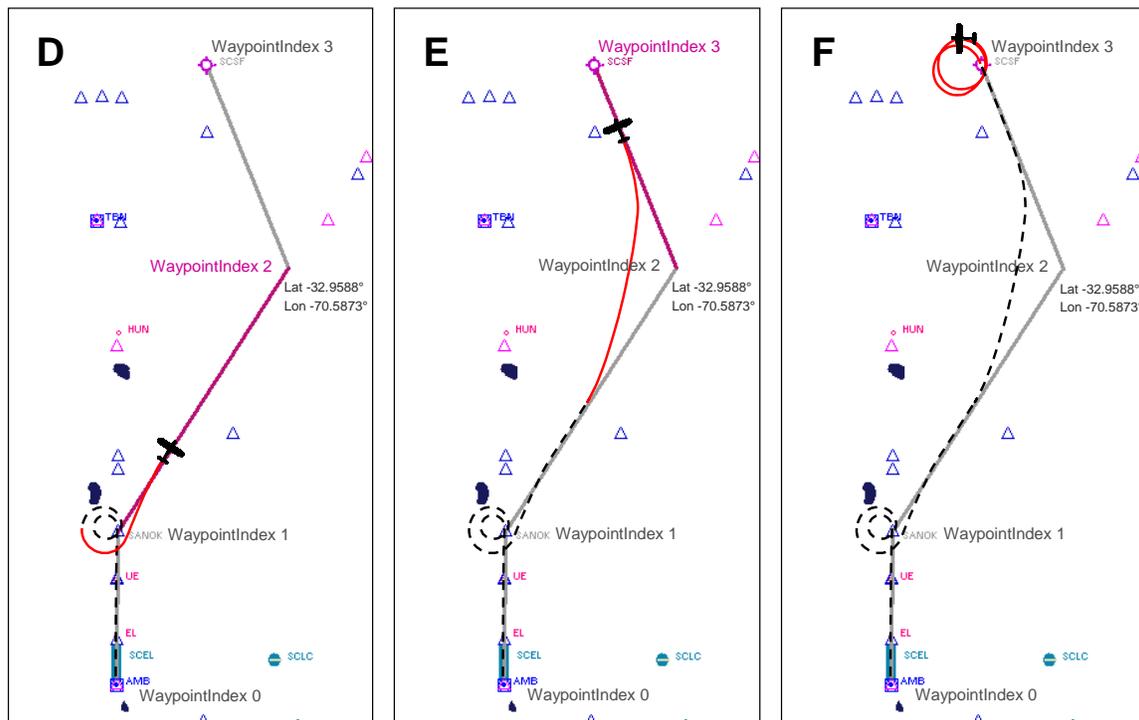
```

4 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint 1 :FlightPlanIsActiveWaypointLocked
1 :FlightPlanRouteType 2 :FlightPlanFlightPlanType 1 :FlightPlanIsActiveWaypoint

```

FlightPlanWaypoint														
Idx	ICAO	111	Ident	Alt	Type	Mag	Lat	Lon	Dist	Ttl	Dist	Rem	Ttl	ETE
0	A	SCEL	SCEL	1554	1	0	-33.40933	-70.78483	0.0	0.0	43.6	0.0	0.0	0.0
1	W	SCSCELSANOK	SANOK	0	2	352	-33.25284	-70.79138	9.4	9.4	34.2	9.4	9.4	2.6
2				0	5	24	-32.95883	-70.58733	20.4	29.8	13.8	20.4	29.8	5.7
3	A	SCSF	SCSF	2160	1	330	-32.74933	-70.70200	13.8	43.6	0.0	13.8	43.6	3.8

Figure C shows the flight at Waypoint 1. Because [ActiveWaypointLocked](#) is 1 (FS9), 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: SCEL to SCSF
 4 :FlightPlanWaypointsNumber 3 :FlightPlanActiveWaypoint 0 :FlightPlanIsActiveWaypointLocked
 1 :FlightPlanRouteType       2 :FlightPlanFlightPlanType 1 :FlightPlanIsActiveWaypoint
```

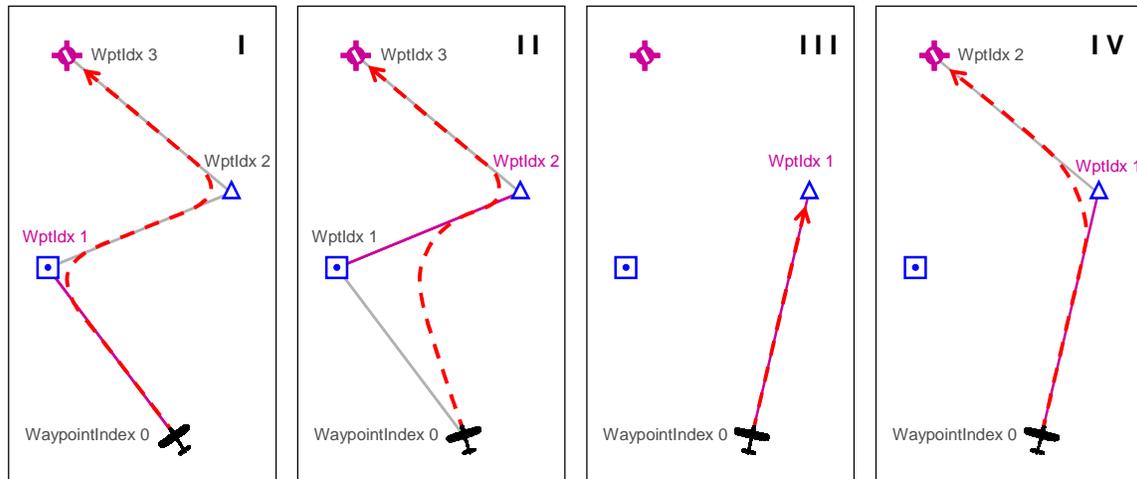
----- FlightPlanWaypoint -----														
Idx	ICAO	111	Ident	Alt	Type	Mag	Lat	Lon	Dist	Ttl	Rem	Dist	Dist	ETE
0	A	SCEL	SCEL	1554	1	0	-33.40933	-70.78483	0.0	0.0	43.6	0.0	0.0	0.0
1	WSCSCELSANOK	SANOK	SANOK	0	2	352	-33.25284	-70.79138	9.4	9.4	34.2	0.0	0.0	2.6
2				0	5	24	-32.95883	-70.58733	20.4	29.8	13.8	0.0	0.0	5.7
3	A	SCSF	SCSF	2160	1	330	-32.74933	-70.70200	13.8	43.6	0.0	21.9	21.9	3.8

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: SCEL to SCSF
 4 :FlightPlanWaypointsNumber 3 :FlightPlanActiveWaypoint 0 :FlightPlanIsActiveWaypointLocked
 1 :FlightPlanRouteType       2 :FlightPlanFlightPlanType 0 :FlightPlanIsActiveWaypoint
```

----- FlightPlanWaypoint -----														
Idx	ICAO	111	Ident	Alt	Type	Mag	Lat	Lon	Dist	Ttl	Rem	Dist	Dist	ETE
0	A	SCEL	SCEL	1554	1	0	-33.40933	-70.78483	0.0	0.0	43.6	0.0	0.0	0.0
1	WSCSCELSANOK	SANOK	SANOK	0	2	352	-33.25284	-70.79138	9.4	9.4	34.2	0.0	0.0	2.6
2				0	5	24	-32.95883	-70.58733	20.4	29.8	13.8	0.0	0.0	5.7
3	A	SCSF	SCSF	2160	1	330	-32.74933	-70.70200	13.8	43.6	0.0	0.0	0.0	3.8

## 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 (not Ident) 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:

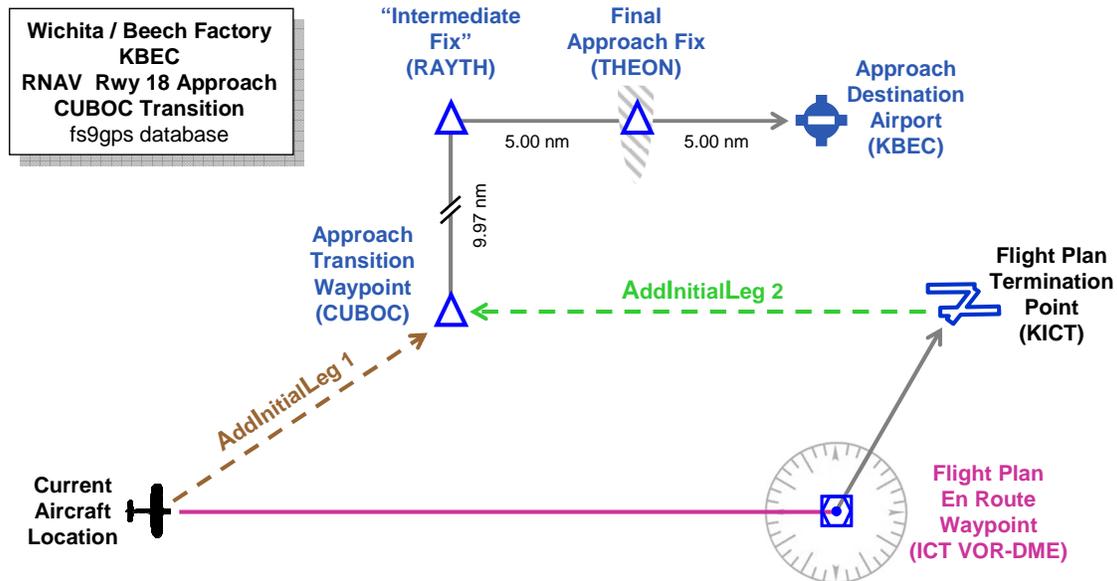
- 0 or negative = **No action**
- 1 **Load.** The [NewApproachAirport](#), [NewApproachApproach](#), [NewApproachTransition](#), [NewApproachMissed](#) and [NewApproachAddInitialLeg](#) variables are Loaded but not Activated.
- 2 **Load and Activate.** The [NewApproachAirport](#), [NewApproachApproach](#), [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

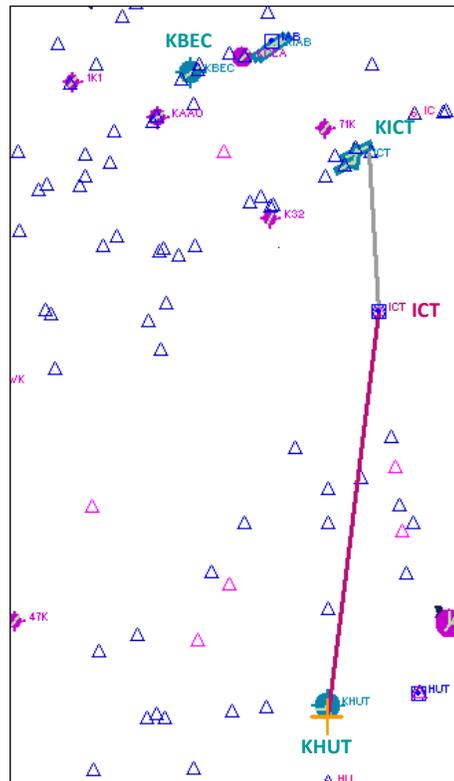
Example 5 demonstrates use of the [NewApproach](#) Group variables. It uses the fs9gps RNAV Rwy 18 Approach into the Beech Aircraft Factory Airport (KBEC), Wichita, Kansas, USA. The stock fs9gps database seems not to contain current RNAV Waypoints; instead, the approach is defined using fs9gps terminal waypoints ( $\Delta$ ). The waypoint names and positions and approach nomenclature used in Example 5 are shown below.



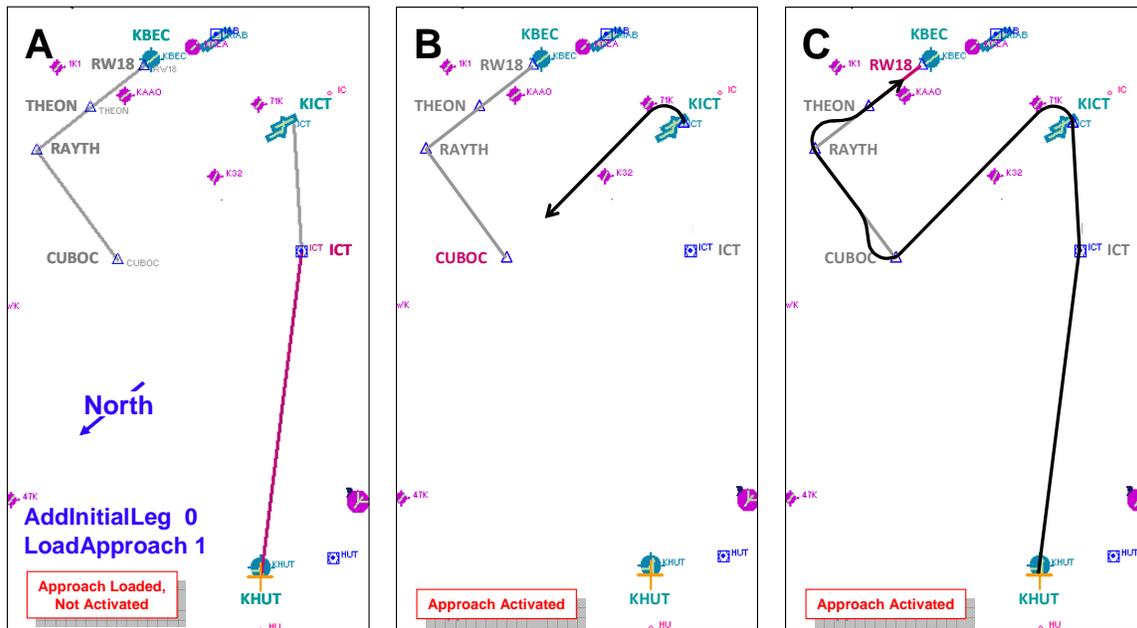
The example begins with a Flight Plan from KHUT to KICT via the ICT VOR-DME en route waypoint.

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: KHUT to KICT**

```
3 :FlightPlanWaypointsNumber  1 :FlightPlanActiveWaypoint
1 :FlightPlanRouteType        2 :FlightPlanFlightPlanType
```

FlightPlanWaypoint														
Idx	ICAO	111	Ident	Alt	Type	Mag	Lat	Lon	Dist	Dist	Dist	Rem	Ttl	ETE
0	A		KHUT	1541	1	0	38.07383	-97.87067	0.0	0.0	33.2	0.0	0.0	0.0
1	VK3		ICT	1470	3	138	37.74517	-97.58383	23.9	23.9	9.3	23.9	23.9	7.2
2	A		KICT	1332	1	128	37.63550	-97.44583	9.3	33.2	0.0	9.3	33.2	2.8

**FLIGHT PLAN NEW APPROACH: KBEC: RNAV 18 approach. CUBOC transition**

```
4 :ApprWaypointsNumber  6 :FlightPlanApprType  3 :FlightPlanWaypointApprIndex
RNAV 18 :FlightPlanApprName  CUBOC :FlightPlanApprTransName  0 :FlightPlanActiveApprWaypoint
0 :FlightPlanIsActiveAppr  2 :FlightPlanApproachIndex  5 :FlightPlanApprTransIndex
```

FlightPlanWaypointApproach											
Idx	ICAO	111	Name	Type	Mode	Lat	Lon	Alt	Trgt	Leg	Course
0	WK3KBEC	CUBOC	CUBOC	1	1	37.90306	-97.38021	4000	0	0.00	-1.0
1	WK3KBEC	RAYTH	RAYTH	1	1	37.86932	-97.17403	3000	0	9.97	101.8
2	WK3KBEC	THEON	THEON	1	1	37.78730	-97.19327	3000	0	5.01	190.5
3	RK3KBEC	RW18	RW18	1	2	37.70527	-97.21246	1453	0	5.01	184.0

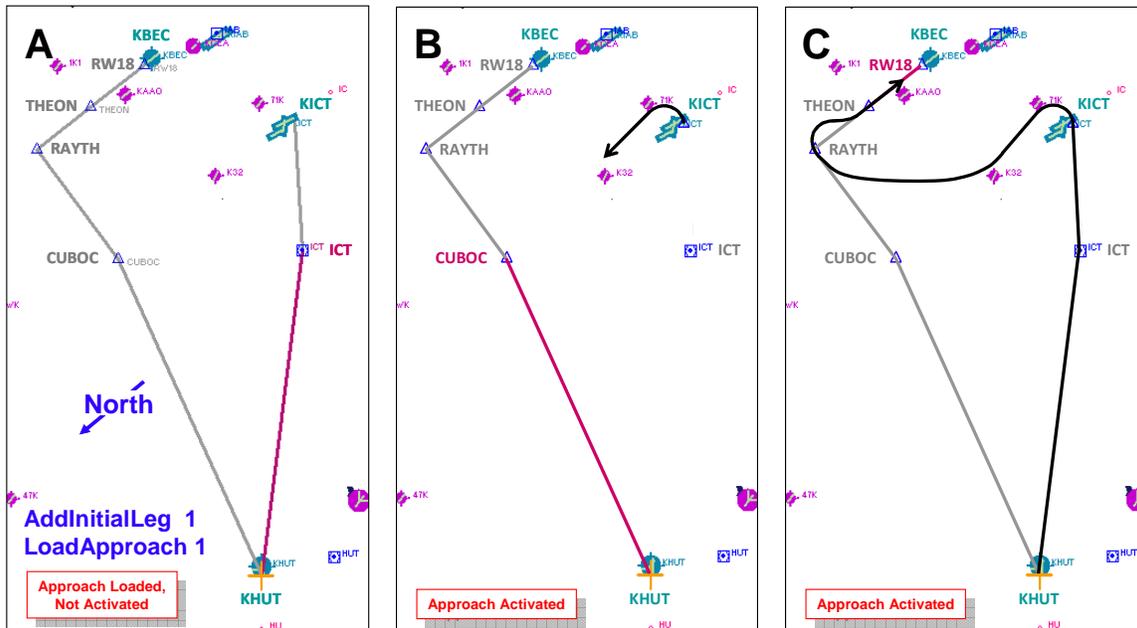
The Flight Plan and Approach segments are listed above.

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:

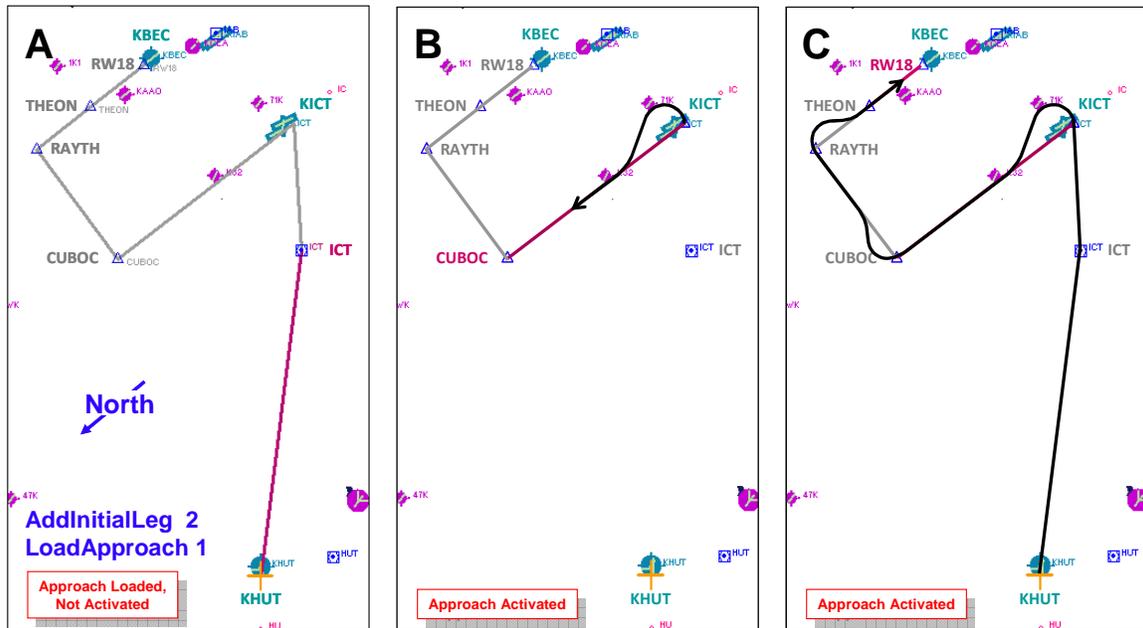
```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
1 (>@c:FlightPlanNewApproachAddInitialLeg)
1 (>@c:FlightPlanLoadApproach)
```

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:

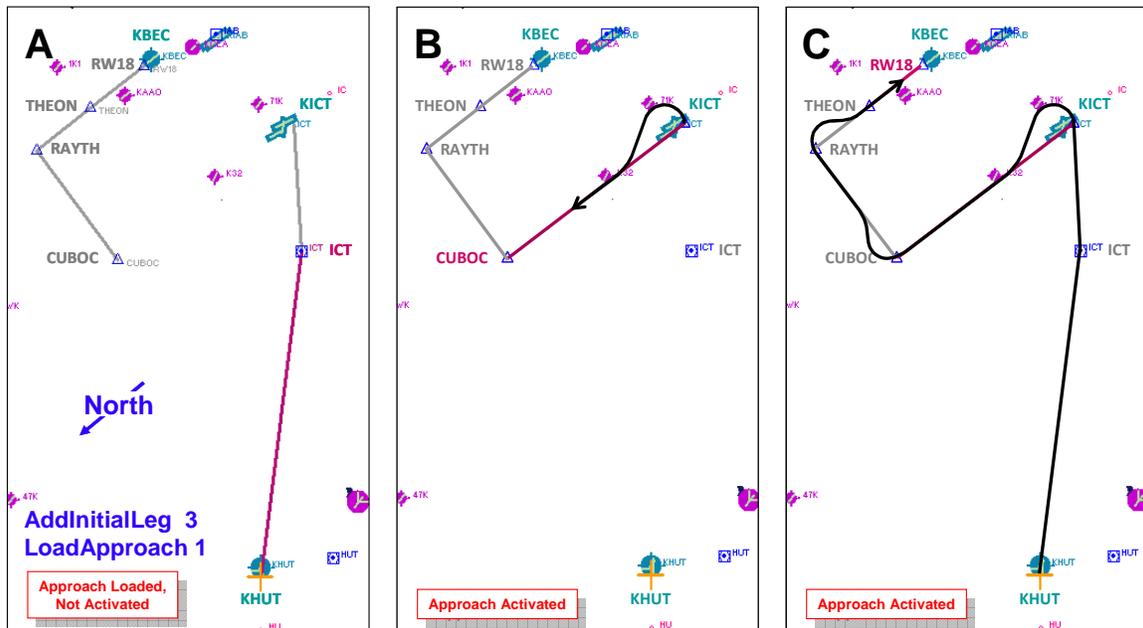
```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
2 (>@c:FlightPlanNewApproachAddInitialLeg)
1 (>@c:FlightPlanLoadApproach)
```

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 initial 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)
```

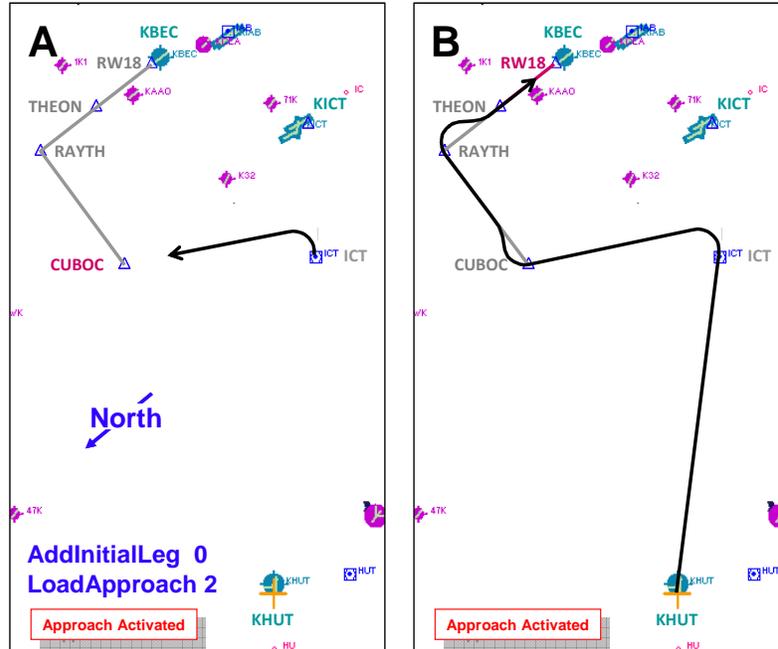
Figure **A** shows approach segments beginning at CUBOC and ending at the destination runway waypoint. No initial legs are shown.

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



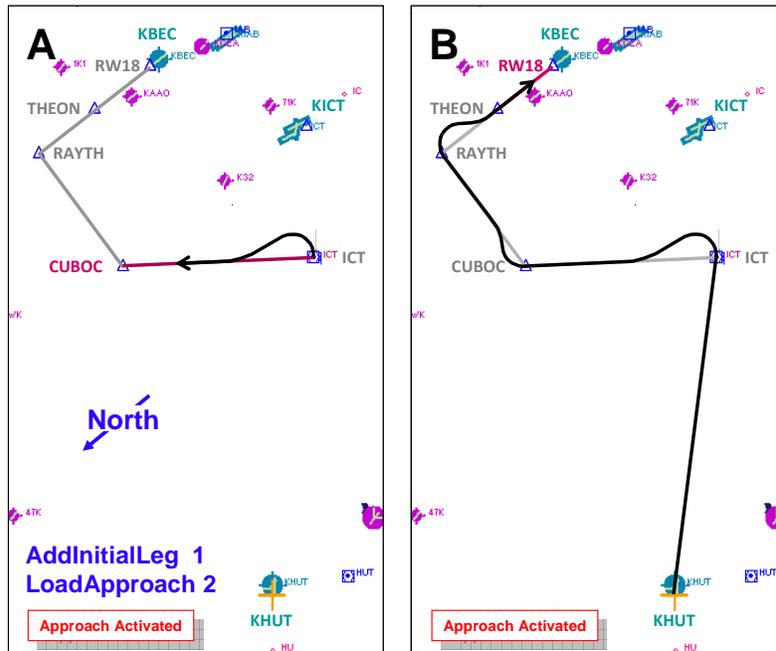
The xml:

```
'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



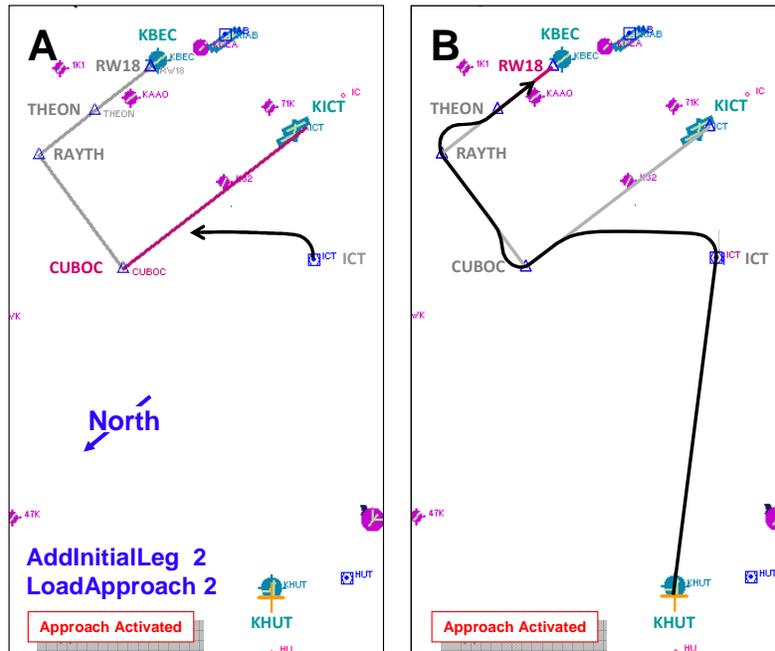
The xml:

```
'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



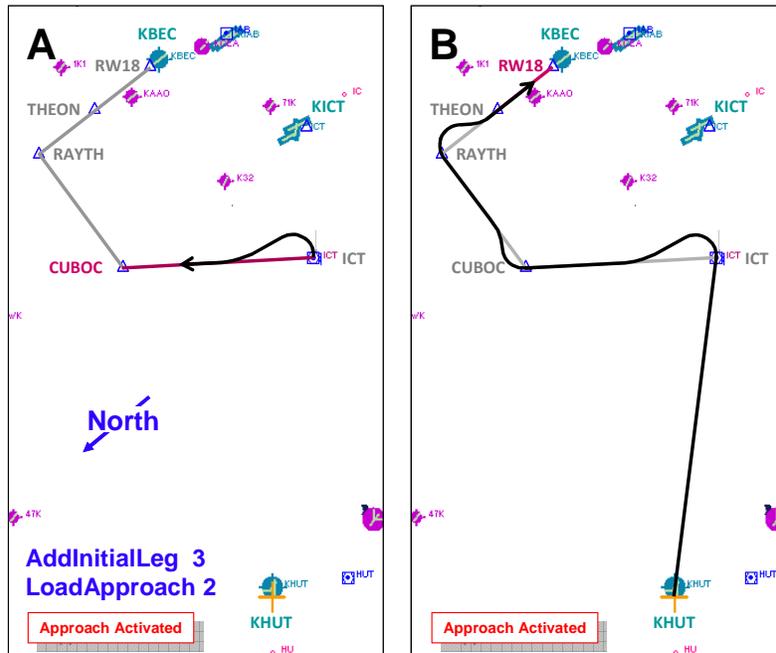
The xml:

```
'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:

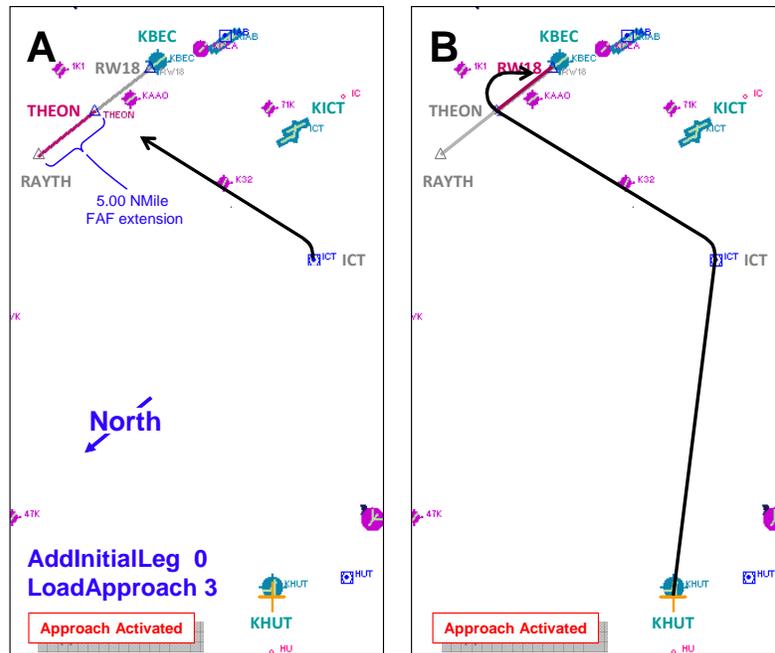
```
'A      KBEC' (>@c:FlightPlanNewApproachAirport)
2 (>@c:FlightPlanNewApproachApproach)
5 (>@c:FlightPlanNewApproachTransition)
0 (>@c:FlightPlanNewApproachMissed)
3 (>@c:FlightPlanNewApproachAddInitialLeg)
2 (>@c:FlightPlanLoadApproach)
```

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: KHUT to KICT**

```

3 :FlightPlanWaypointsNumber  1 :FlightPlanActiveWaypoint
1 :FlightPlanRouteType        2 :FlightPlanFlightPlanType
----- FlightPlanWaypoint -----
Idx  ICAO  111  Ident  Alt  Type  Hdg  Lat  Lon  Dist  Ttl  Dist  Rem  Ttl  ETE
0    A    KHUT  KHUT  1541  1    0    38.07383  -97.87067  0.0  0.0  33.2  0.0  0.0  0.0
1    VK3   ICT   ICT   1470  3    138  37.74517  -97.58383  23.9  23.9  9.3   23.9  23.9  7.2
2    A    KICT  KICT  1332  1    128  37.63550  -97.44583  9.3   33.2  0.0   9.3   33.2  2.8

```

**FLIGHT PLAN NEW APPROACH: KBEC: RNAV 18 approach. VECTORS transition**

```

2 :ApprWaypointsNumber  6 :FlightPlanApprType  1 :FlightPlanWaypointApprIndex
RNAV 18 :FlightPlanApprName  VECTORS :FlightPlanApprTransName  0 :FlightPlanActiveApprWaypoint
1 :FlightPlanIsActiveAppr  2 :FlightPlanApproachIndex  0 :FlightPlanApprTransIndex
----- FlightPlanWaypointApproach -----
Idx  ICAO  111  Name  Type  Mode  Lat  Lon  Alt  Trgt  Leg  Course
0    WK3KBEC  THEON  11    2    37.78730  -97.19327  3000  0    5.00  181.5
1    RK3KBECRW18  RW18  1     2    37.70527  -97.21246  1453  0    5.01  184.0

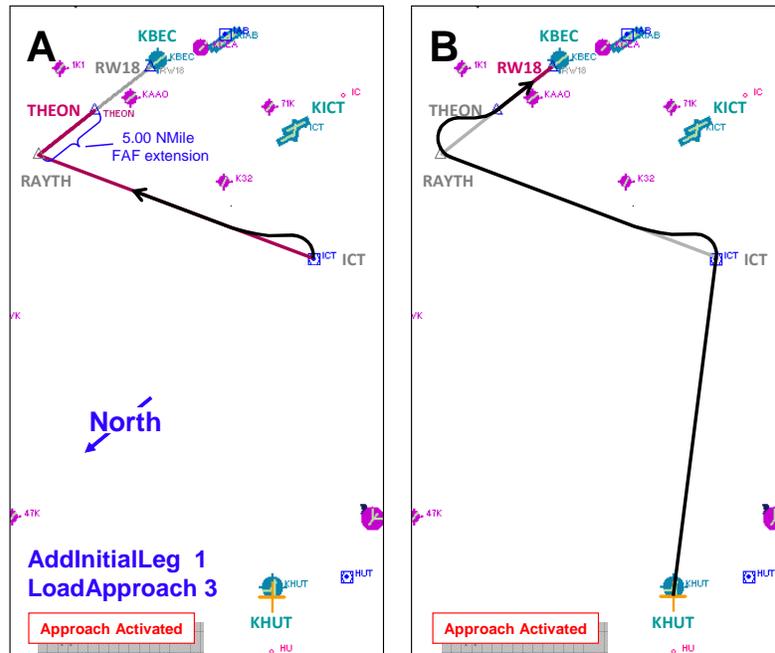
```

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 not 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 demonstrate actions of the different [FlightPlanNewApproach](#) selections more clearly.

## 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]**

For Waypoint types 1, 3, and 4 (Airport, VOR, VOR) the default value is the asl ground elevation of the currently indexed waypoint. Default value for Intersection (Type 2) and User Waypoints (Type 5) is zero.

[FlightPlanWaypointAltitude](#) can be changed by manually editing the altitude field of the .pln file and reloading the flight plan. For example, the new altitude for Waypoint 3 is 15000':

```
waypoint.3=K1, LKV, , LKV, V, N42* 29.57', W120* 30.43', +015000.00,
```

### ❑ **FlightPlanWaypointICAO (string) [Get]**

The ICAO of the currently indexed Waypoint.

### ❑ **FlightPlanWaypointIdent (string) [Get]**

The Ident of the currently indexed Waypoint. [FlightPlanWaypointIdent](#) can accommodate a string length up to 10. Although Idents with slen greater than 5 do not exist in the fs9gps database, it is possible to create a Type 5 User Waypoint and assign an Ident name up to 10 characters long by through use of [FlightPlanNewWaypointIdent](#), for example:

```
'ABC1234567' (>@c:FlightPlanNewWaypointIdent)
```

A User defined ident such as this will not permanently update the fs9gps database nor is it searchable by ICAO Search.

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

The Low Alt (Victor) or High Alt (Jet) Airway Ident if the currently indexed leg is part of an Airway and [FlightPlanRouteType](#) = 2 or 3 (Low Alt or High Alt Airways).

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

#	Waypoint Type	#	Waypoint Type
0	NONE	4	NDB
1	AIRPORT	5	USER
2	INTERSECTION	6	ATC
3	VOR		

[http://msdn.microsoft.com/en-us/library/cc526954.aspx#ATC\\_WAYPOINT\\_TYPE](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]**

[WaypointMinAltitude](#) is the Minimum En route Altitude assigned to Low Altitude Victor and High Altitude Jet Airways. Only flight plan legs that are part of a Victor or Jet Airway have [WaypointMinAltitude](#) and it is returned only when [FlightPlanRouteType](#) = 2 or 3. Flight Planner **Find Route**   must also first be clicked.

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: KLIC to KDTA

```

18 :FlightPlanWaypointsNumber      1 :FlightPlanActiveWaypoint
 2 :FlightPlanRouteType           2 :FlightPlanFlightPlanType      18000 :FlightPlanCruisingAltitude
----- FlightPlanWaypoint -----

```

Idx	ICAO	111	Airwy	Ident	Ident	Alt	Min Alt	Type	Mag	Spd	Dist	Dist	Dist	Rem	ETE
0	A	KLIC	KLIC			5364	0	1	0	200	0.0	0.0	430.3	0.0	0.0
1	VK2	FQF	FQF			0	0	3	289	200	50.6	50.6	379.7	50.6	15.2
2	WK2	BREWS	BREWS	V134		0	16500	2	254	200	25.9	76.5	353.8	25.9	7.8
3	WK2	FUNDS	FUNDS	V134		0	16500	2	254	200	41.3	117.9	312.4	41.3	12.4
4	WK2	DAVY	DAVY	V134		0	1600	2	245	200	13.7	131.6	298.7	13.7	4.1
5	WK2	LAWNS	LAWNS	V134		0	16000	2	245	200	5.5	137.1	293.2	5.5	1.7
6	WK2	HERLS	HERLS	V134		0	16000	2	245	200	12.0	149.1	281.2	12.0	3.6
7	VK2	DBL	DBL	V134		0	14000	3	244	200	8.0	157.1	273.3	8.0	2.4
8	WK2	LINDZ	LINDZ	V134		0	14000	2	244	200	12.6	169.6	260.7	12.6	3.8
9	WK2	GLENO	GLENO	V591		0	14000	2	244	200	10.1	179.7	250.6	10.1	3.0
10	WK2	SLOLM	SLOLM	V591		0	14000	2	243	200	12.4	192.1	238.2	12.4	3.7
11	WK2	EDUKY	EDUKY	V591		0	13000	2	243	200	23.0	215.2	215.2	23.0	6.9
12	WK2	PACES	PACES	V591		0	13000	2	243	200	8.2	223.4	206.9	8.2	2.5
13	VK2	JNC	JNC	V591		0	9000	3	243	200	24.8	248.2	182.2	24.8	7.4
14	WK2	EZEZE	EZEZE	V134		0	11900	2	278	200	81.7	329.9	100.4	81.7	24.5
15	VK2	PUC	PUC	V134		0	11900	3	276	200	15.0	344.8	85.5	15.0	4.5
16	WK2	ARBIH	ARBIH	V134		0	13000	2	293	200	10.0	354.8	75.5	10.0	3.0
17	A	KDTA	KDTA			4754	0	1	242	200	75.5	430.3	0.0	75.5	22.6

FLIGHT PLAN: KLIC to KDTA

```

6 :FlightPlanWaypointsNumber      1 :FlightPlanActiveWaypoint
3 :FlightPlanRouteType            2 :FlightPlanFlightPlanType      34000 :FlightPlanCruisingAltitude

```

Idx	ICAO		Airwy		Alt	FlightPlanWaypoint			Dist	Dist Ttl	Dist Rem	Dist	ETE	
	111	112	Ident	Ident		Min Alt	Type	Mag						Spd
0	A	KLIC	KLIC		5363	0	1	0	200	0.0	0.0	419.9	0.0	0.0
1	VK2	FQF	FQF		0	0	3	289	200	50.6	50.6	369.3	50.6	15.2
2	VK2	EKR	EKR	J116	0	20000	3	268	200	153.8	204.4	215.5	153.8	46.1
3	WK2	HELPR	HELPR	J199	0	33000	2	249	200	122.2	326.6	93.3	122.2	36.7
4	WK2	OFFIG	OFFIG	J199	0	33000	2	247	200	41.3	367.9	52.0	41.3	12.4
5	A	KDTA	KDTA		4753	0	1	239	200	52.0	419.9	0.0	52.0	15.6

**Units bug:** `MinAltitude` is populated in the database in feet, but its Units flag is the FS distance default, meters. If you want `MinAltitude` in **feet**, you must either omit units or specify units as 'meters'. For example, V10 airway between Hutchinson VOR (HUT) and STAFF intersection (Kansas, USA) has a real-life MEA of 3700 feet:

`(C:fs9gps:FlightPlanWaypointMinAltitude) = 3700` or

`(C:fs9gps:FlightPlanWaypointMinAltitude, meters) = 3700`

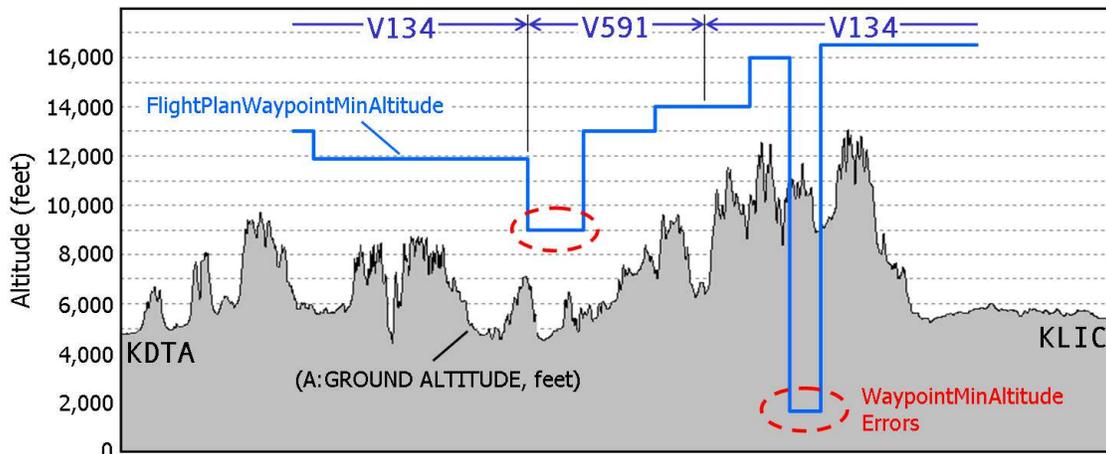
But, if you specify units = feet:

`(C:fs9gps:FlightPlanWaypointMinAltitude, feet) = 12140`

the database altitude will be interpreted in default FS units, 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, for example:

`(C:fs9gps:FlightPlanWaypointMinAltitude) 3.281 /`  
`(>L:FlightPlanWaypointMinAltitude, meters)`

Not all segments of all airways within the FS9 gps database have assigned MEAs - some isolated segments simply have a value of zero. Additionally, some other segments may have an obviously incorrect value while the adjoining segments show the proper MEA as demonstrated below. With FSX, these infrequent errors may have been corrected, but I'm not sure.



### ✖ **FlightPlanWaypointFrequency (MHz) [Get]**

[FlightPlanWaypointFrequency](#) returns incorrect VOR and NDB frequency data in FS9 but functions properly in FSX. Note that default Flight Sim frequency units are Hertz, so if this variable is displayed using MHz units that would be appropriate for VORs, then KHZ NDB frequencies displayed in the *same* list will 'appear' incorrect (off by a factor of 1000) but in fact are valid frequencies.

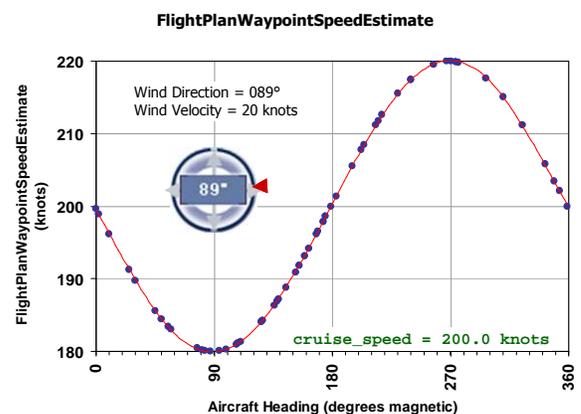
### ☐ **FlightPlanWaypointMagneticHeading (degrees) [Get]**

[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.



FLIGHT PLAN: VTUK to VTTP

```

4 :FlightPlanWaypointsNumber      1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType            2 :FlightPlanFlightPlanType      8000 :FlightPlanCruisingAltitude
    
```

FlightPlanWaypoint												
Idx	ICAO	111	Ident	Alt	Type	Mag	Spd	Dist	Dist	Dist	Rem	RemTtl
0	A	VTUK	VTUK	670	1	0	200	0.0	0.0	144.8	0.0	0.0
1	VVT	CMP	CMP	650	3	284	200	46.6	46.6	98.2	46.6	46.6
2	VVT	PCB	PCB	449	3	274	200	45.6	92.2	52.6	45.6	92.2
3	A	VTTP	VTTP	145	1	277	200	52.6	144.8	0.0	52.6	144.8

❑ **FlightPlanWaypointRemainingDistance (nmiles) [Get]**

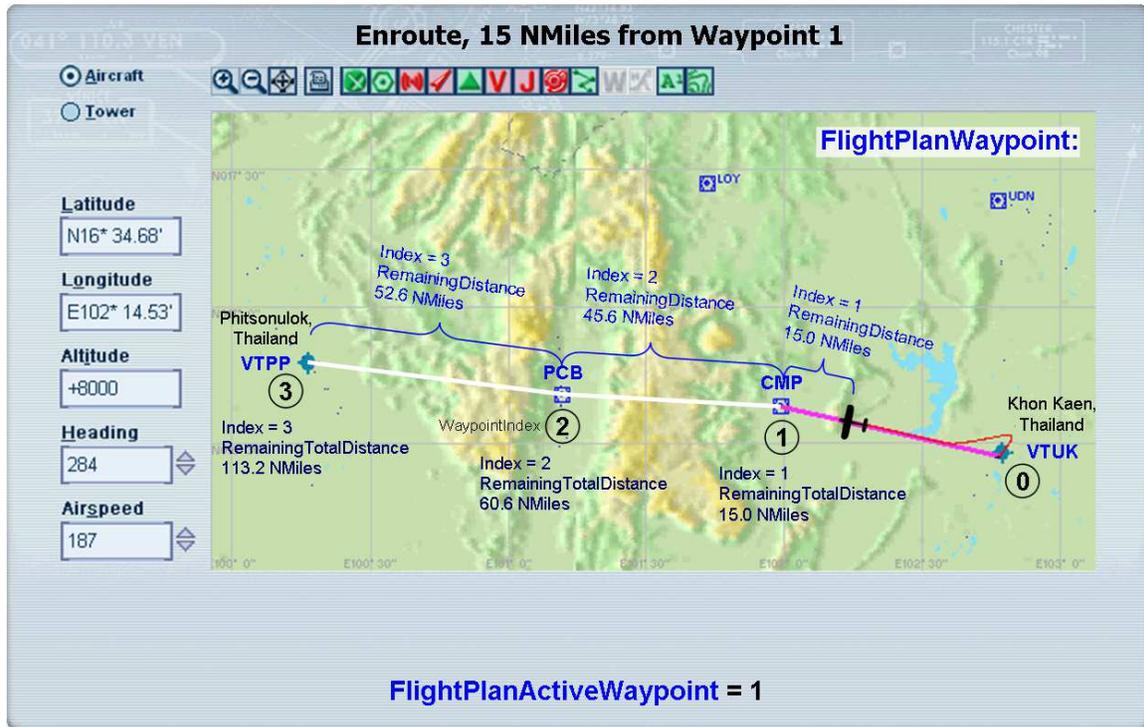
[FlightPlanWaypointRemainingDistance](#) is the leg distance remaining to be flown. For the active waypoint, that is, for the segment currently being flown, it is the remaining distance from the aircraft's current position to the next Waypoint. For Waypoints beyond that, it is just the total length of 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 [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.



### FLIGHT PLAN: VTUK to VTPP

```
4 :FlightPlanWaypointsNumber      1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType            2 :FlightPlanFlightPlanType      8000 :FlightPlanCruisingAltitude
```

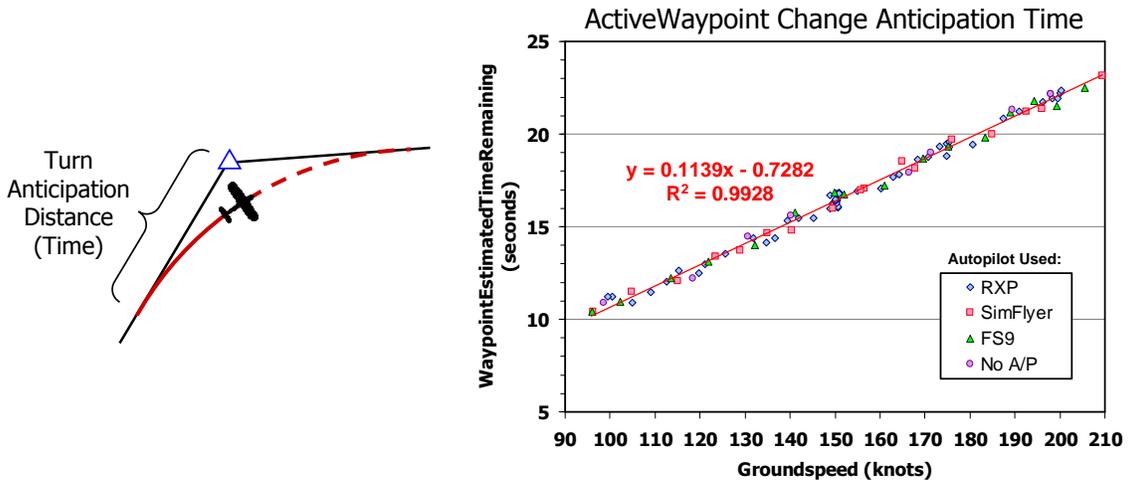
Idx	FlightPlanWaypoint												ETE
	ICAO	Ident	Alt	Type	Mag	Spd	Dist	Dist	Dist	Rem	Rem	RemTtl	
0	A	VTUK	670	1	0	200	0.0	0.0	144.8	0.0	0.0	0.0	0.0
1	VVT	CMP	650	3	284	200	46.6	46.6	98.2	15.0	15.0	14.0	14.0
2	VVT	PCB	449	3	274	200	45.6	92.2	52.6	45.6	60.5	13.7	13.7
3	A	VTPP	145	1	277	200	52.6	144.8	0.0	52.6	113.1	15.8	15.8

## 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 time is reached (**WaypointEstimatedTimeRemaining**). 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: VTUK to VTPP
4 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint
0 :FlightPlanRouteType       2 :FlightPlanFlightPlanType      8000 :FlightPlanCruisingAltitude
```

----- FlightPlanWaypoint -----												
Idx	ICAO	111	Ident	Alt	Type	Mag	Spd	Dist	Dist	Rem	RemTtl	ETE
0	A	VTUK	VTUK	670	1	0	200	0.0	0.0	144.8	0.0	0.0
1	VVT	CMP	CMP	650	3	284	200	46.6	46.6	98.2	1.4	14.0
2	VVT	PCB	PCB	449	3	274	200	45.6	92.2	52.6	45.6	13.7
3	A	VTPP	VTPP	145	1	277	200	52.6	144.8	0.0	52.6	15.8

[ActiveWaypoint](#) changes to 2 when [EstimatedTimeRemaining](#) = 0.38 (23 seconds) at which point, the aircraft is 1.4 NMiles from Waypoint 2. At the instant [ActiveWaypoint](#) changes, Waypoint 1 [RemainingDistance](#) becomes 0.0 and the 1.4 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 1.4 NMiles.

### ActiveWaypoint Change, WP 1 to WP 2

**Aircraft**

**Tower**

**Latitude**  
N16° 37.80'

**Longitude**  
E102° 0.66'

**Altitude**  
+8000

**Heading**  
284

**Airspeed**  
187

ActiveWaypoint advances by 1 prior to the aircraft actually arriving at the Waypoint.

RemainingDistance for Waypoint Index 2 then becomes the sum of WaypointDistance (45.6 NMiles) plus the ActiveWaypoint Change Lead Distance (1.3 NMiles) for a total of 46.9 NMiles.

**FlightPlanWaypoint:**

**FlightPlanActiveWaypoint = 2**

**FLIGHT PLAN: VTUK to VTTP**

4 :FlightPlanWaypointsNumber    2 :FlightPlanActiveWaypoint  
 0 :FlightPlanRouteType            2 :FlightPlanFlightPlanType        8000 :FlightPlanCruisingAltitude

FlightPlanWaypoint												
Idx	ICAO	111	Ident	Alt	Type	Mag	Spd	Dist	Dist	Rem	RemTtl	ETE
	123456789012					Hdg	Est	Dist	Ttl	Rem	Dist	Dist
0	A	VTUK	VTUK	670	1	0	200	0.0	0.0	144.8	0.0	0.0
1	VVT	CMP	CMP	650	3	284	200	46.6	46.6	98.2	0.0	14.0
2	VVT	PCB	PCB	449	3	274	200	45.6	92.2	52.6	1.4	13.7
3	A	VTTP	VTTP	145	1	277	200	52.6	144.8	0.0	52.6	15.8

A moment later, Waypoint Index 2 RemainingDistance is updated by adding Waypoint Index 2 distance, 45.6 NMiles, which results in Waypoint 2 RemainingDistance = 46.8 NMiles.

**FLIGHT PLAN: VTUK to VTTP**

4 :FlightPlanWaypointsNumber    2 :FlightPlanActiveWaypoint  
 0 :FlightPlanRouteType            2 :FlightPlanFlightPlanType        8000 :FlightPlanCruisingAltitude

FlightPlanWaypoint												
Idx	ICAO	111	Ident	Alt	Type	Mag	Spd	Dist	Dist	Rem	RemTtl	ETE
	123456789012					Hdg	Est	Dist	Ttl	Rem	Dist	Dist
0	A	VTUK	VTUK	670	1	0	200	0.0	0.0	144.8	0.0	0.0
1	VVT	CMP	CMP	650	3	284	200	46.6	46.6	98.2	0.0	14.0
2	VVT	PCB	PCB	449	3	274	200	45.6	92.2	52.6	46.8	13.7
3	A	VTTP	VTTP	145	1	277	200	52.6	144.8	0.0	52.6	15.8

It's tedious to go through the steps of a waypoint change like this, but informative to understand. 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]**

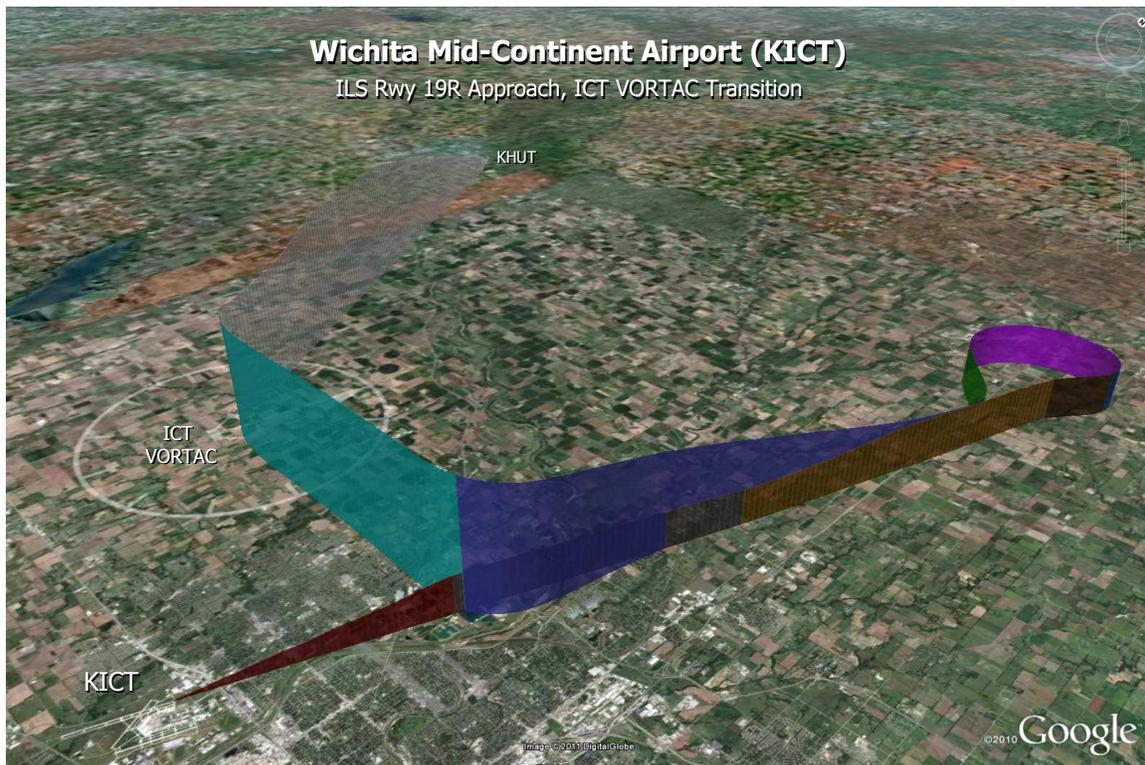
Fuel calculations are based on fuel consumption rates derived from the aircraft model design and configuration and fuel quantity values. Fuel consumption rates and quantity can be accessed from A:Vars, for example, (`A:ENG1 FUEL FLOW GPH`, gallons per hour) and (`A:FUEL TOTAL QUANTITY`, gallons).

## 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.

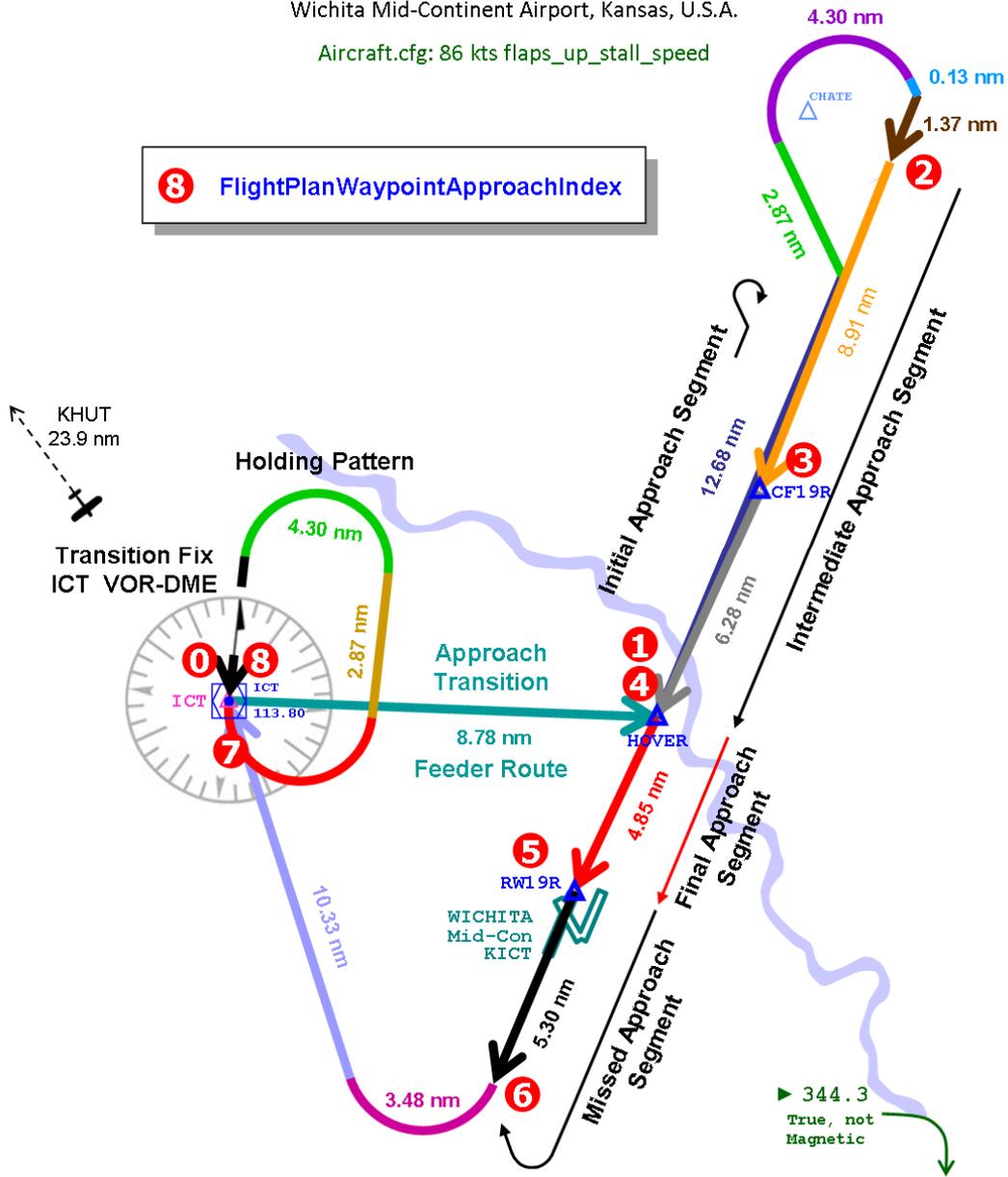
Throughout the discussion of Approach variables, an example instrument flight from Hutchinson Municipal Airport (Hutchinson, Kansas, USA, "KHUT") to Wichita Mid-Continent Airport (Wichita, Kansas, USA, "KICT") is used, incorporating the ILS Rwy 19R Approach, ICT VORTAC Transition into Wichita.

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 at which point guidance is switched from GPS to NAV. 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



**8** FlightPlanWaypointApproachIndex

**FLIGHT PLAN: KHUT to KICT**

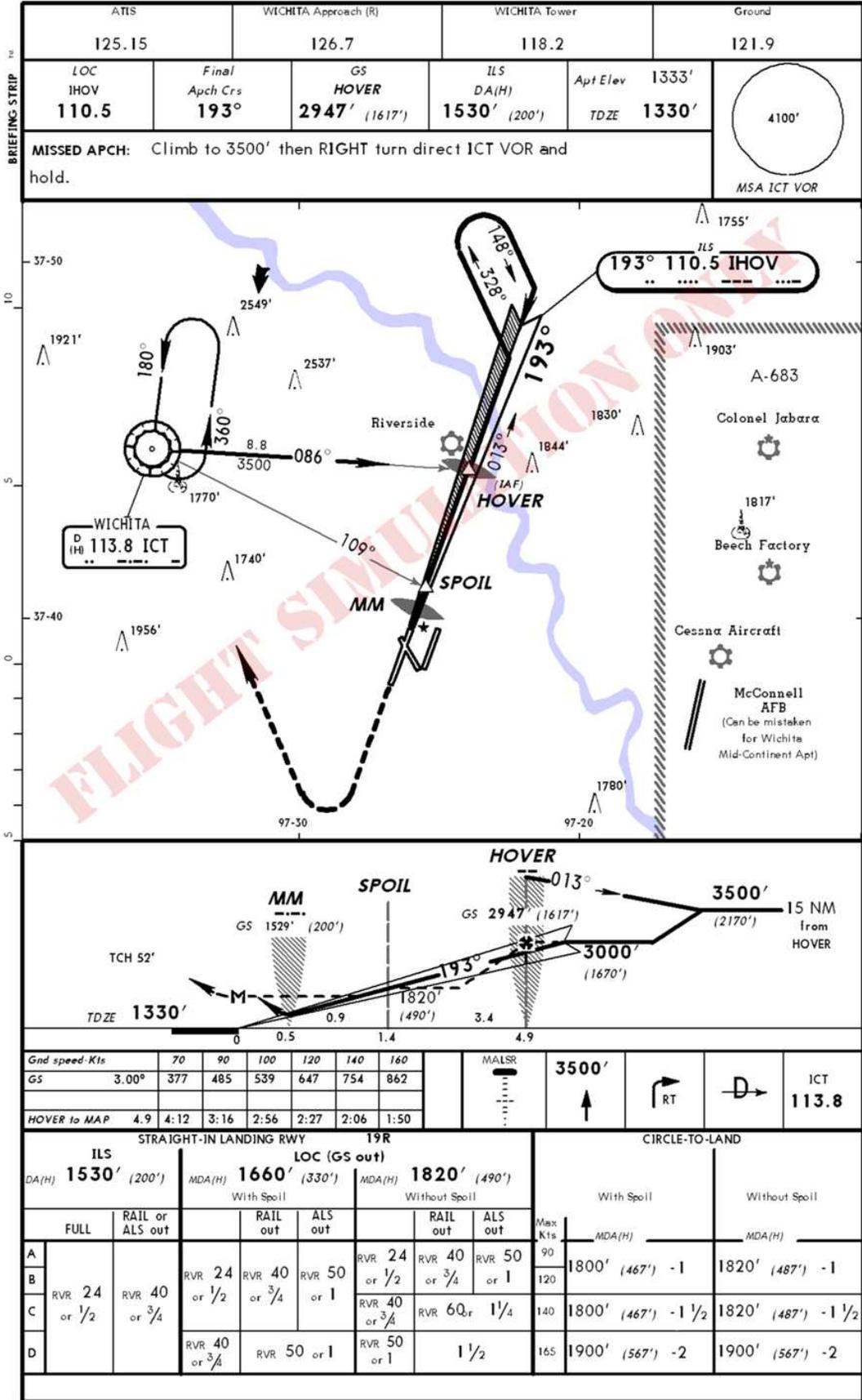
5000 :FlightPlanCruisingAltitude 2 :FlightPlanFlightPlanType  
 1 :FlightPlanRouteType 3 :FlightPlanWaypointsNumber 1 :FlightPlanActiveWaypoint

FlightPlanWaypoint													
Idx	ICAO	111	Ident	Alt	Type	Mag	Hdg	Est	Dist	Dist Ttl	Dist Rem	Rem Dist	ETE
0	A	KHUT	KHUT	1541	1	0	199	0.0	0.0	0.0	33.2	0.0	0.0
1	VK3	ICT	ICT	1470	3	138	200	23.9	23.9	9.3	23.9	7.2	7.2
2	A	KICT	KICT	1332	1	128	200	9.3	33.2	0.0	9.3	2.8	2.8

**FLIGHT PLAN APPROACH: KICT: ILS 19R approach. ICT transition**

3 :FPApprIndex 1 :FPApprTransIdx 9 :ApprWaypointsNumber 1 :FlightPlanRouteType

FlightPlanWaypointApproach												
Idx	ICAO	111	Name	Type	Mode	Lat	Lon	Alt	Trgt	Leg Dist	Course (mag)	
0	VK3	ICT	ICT	1	1	37.74523	-97.58382	0	0	0.00	-1.0	
1	WK3	KICTHOVER	HOVER	1	1	37.73756	-97.39899	3500	0	8.78	► 93.1	
2	WK3	KICTHOVER	HOVER	3	1	37.97751	-97.29716	3500	0	21.35	328.0	
3	WK3	KICTCF19R	CF19R	1	2	37.83587	-97.35387	3500	0	8.91	► 197.5	
4	WK3	KICTHOVER	HOVER	1	2	37.73756	-97.39899	3000	0	6.28	193.0	
5	RK3	KICTRW19R	RW19R	1	2	37.66160	-97.43382	1382	0	4.85	193.0	
6				9	3	37.57814	-97.47008	3500	3500	5.30	193.0	
7	VK3	ICT	ICT	1	3	37.74523	-97.58382	3500	0	13.81	► 344.3	
8	VK3	ICT	ICT	6	3	37.74523	-97.58382	3500	0	14.34	180.0	



© JEPPESSEN SANDERSON, INC., 1998, 2003. ALL RIGHTS RESERVED.

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

[FlightPlanApproachWaypointType](#) is an enum referring to the navigation procedure objective of the currently *active* (as opposed to currently indexed) approach segment.

#	Name	#	Name	#	Name
0	NONE	4	DME_ARC_LEFT	8	DISTANCE
1	FIX	5	DME_ARC_RIGHT	9	ALTITUDE
2	PROC_TURN_LEFT	6	HOLDING_LEFT	10	MANUAL_SEQ
3	PROC_TURN_RIGHT	7	HOLDING_RIGHT	11	VECTORS_TO_FINAL

[http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS\\_Approach\\_Waypoint\\_Type](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.

#	Approach Mode	#	Approach Mode
0	NONE	2	FINAL
1	TRANSITION	3	MISSED

[http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS\\_APPR\\_TYPE](http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS_APPR_TYPE)

[ApproachMode](#) is similar to US F.A.A. Approach Segment nomenclature as listed below:

<b>US FAA Approach Procedure Segments</b>	<b>FS9, FSX FlightPlanApproachMode</b>	<b>US FAA Approach Procedure Segments</b>	<b>FS9, FSX FlightPlanApproachMode</b>
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.

This can be 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 down 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.

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

<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	ILS 01L	6	GPS 01R	12	RNAV 01R
1	ILS 01R	7	GPS 14	13	RNAV 14
2	ILS 19L	8	GPS 19L	14	RNAV 19L
<b>3</b>	<b>ILS 19R</b>	9	GPR 19R	15	RNAV 19R
4	VOR 01R	10	GPS 32	16	RNAV 32
5	GPS 01L	11	RNAV 01L	17	VOR 14

### ❑ **FlightPlanApproachName (string) [Get]**

[FlightPlanApproachName](#) is the name of the selected approach. In the KICT example, it is "**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
<b>1</b>	<b>ICT</b>
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.

#	Name	#	Name	#	Name
0	NONE	4	DME_ARC_LEFT	8	DISTANCE
1	FIX	5	DME_ARC_RIGHT	9	ALTITUDE
2	PROC_TURN_LEFT	6	HOLDING_LEFT	10	MANUAL_SEQ
3	PROC_TURN_RIGHT	7	HOLDING_RIGHT	11	VECTORS_TO_FINAL

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

[FlightPlanWaypointApproachMode](#) is a number used to describe the *currently indexed* segment of the approach.

#	Approach Mode	#	Approach Mode
0	NONE	2	FINAL
1	TRANSITION	3	MISSED

[http://msdn.microsoft.com/en-us/library/cc526954.aspx#GPS\\_APPR\\_TYPE](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:

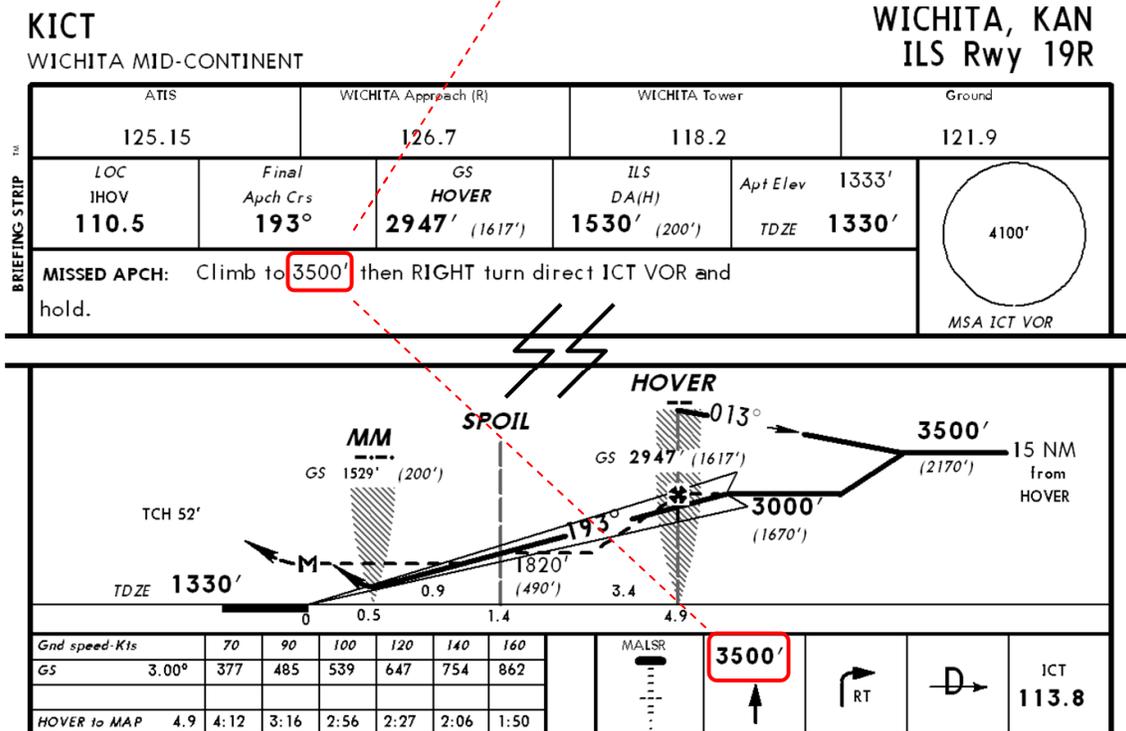
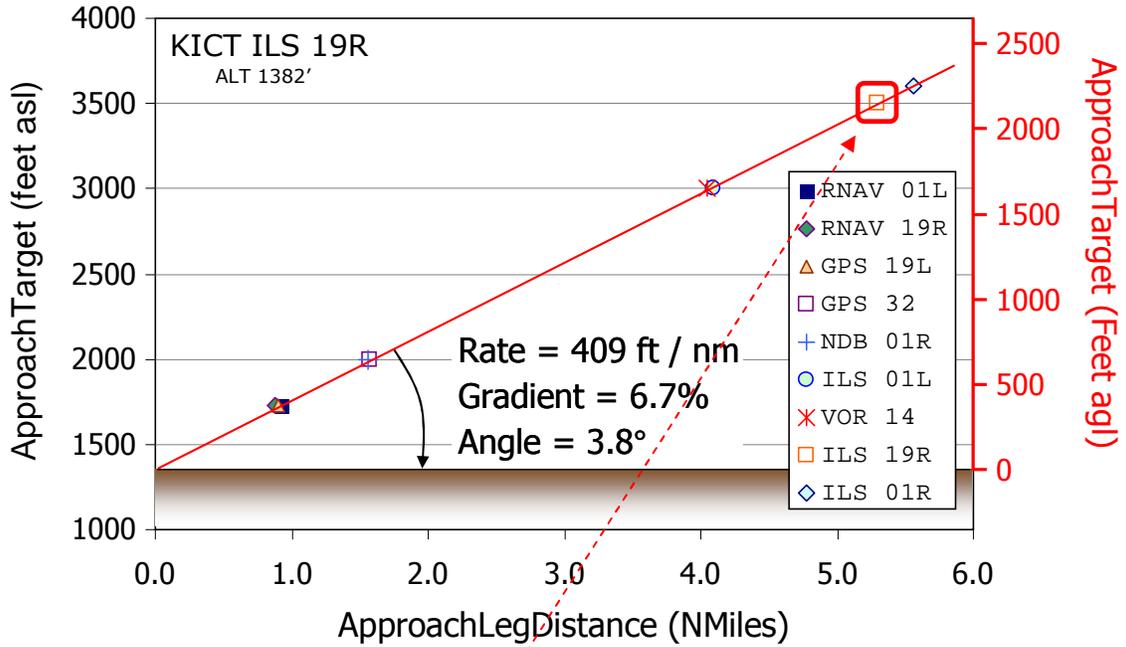
- **Procedure Turn:** [ApproachCourse](#) is the magnetic bearing of the 45° turn. In the KICT ILS Rwy 19R example, 328°. Fs9gps correctly returns 328° M.
- **Missed Approach Turn Toward Holding Fix:** [ApproachCourse](#) is the bearing of the culmination of the Missed Approach turn. Fs9gps returns a 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** font in the FS9 Transitions and Approach Segments ILS 19R approach diagram (Page 176) are degrees True, but fs9gps should have returned degrees Magnetic. I have no idea why this occurs.

- ❑ **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.

# WaypointApproachTarget

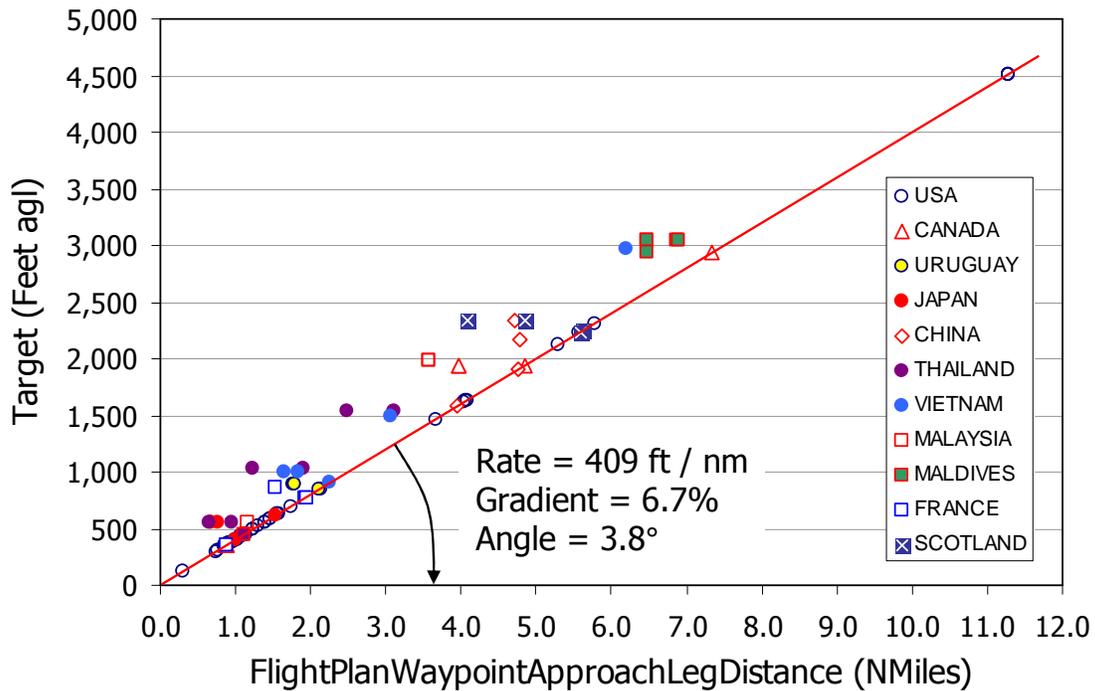


The approach segment that contains **WaypointApproachTarget** will be the first approach segment with **WaypointApproachMode** = 3. It may or may not have an ICAO or Name. At minimum, the Target waypoint has Lat and Lon, Type and Mode, which are sufficient information to define an Approach Waypoint.

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.

## WaypointApproachTarget



### ❑ **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. [WaypointApproachRemainingDistance](#) is 0.0 for approach segments already passed.

[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	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	WK3	KICTHOVER	HOVER	1	1	37 44.2538	-97 23.9393	37.737564	-97.398989	3500	0	8.78	93.06
2	WK3	KICTHOVER	HOVER	3	1	37 58.6505	-97 17.8293	37.977508	-97.297155	3500	0	21.35	328.00
3	WK3	KICTCF19R	CF19R	1	2	37 50.1523	-97 21.2320	37.835872	-97.353867	3500	0	8.91	197.52
4	WK3	KICTHOVER	HOVER	1	2	37 44.2538	-97 23.9393	37.737564	-97.398989	3000	0	6.28	193.00
5	RK3	KICTRW19R	RW19R	1	2	37 39.6962	-97 26.0290	37.661604	-97.433817	1382	0	4.85	193.00
6					9	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

```



ICT

En Route Fix Segment 0 Index

Segment	FlightPlanWaypointApproach				Leg		
	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

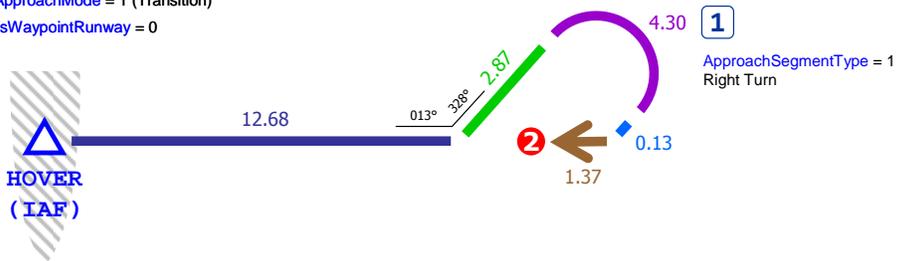


Approach Transition Segment						1	Index
Segment	FlightPlanWaypointApproach				Course	Leg	
	Type	Mode	Altitude	Target		Distance	
Appr. Transition	1	1	3500	N/A (0')	093°	8.78	Begins at Enroute Fix. Ends at IAF - HOVER
						8.78 nm	:FlightPlanWaypointApproachLegDistance

## FlightPlanWaypointApproachIndex 2:

### Initial Approach

FlightPlanActiveApproachWaypoint = 2 (the Index)  
 FlightPlanWaypointApproachType = 3 (Procedure Turn Right)  
 FlightPlanWaypointApproachMode = 1 (Transition)  
 FlightPlanApproachIsWaypointRunway = 0

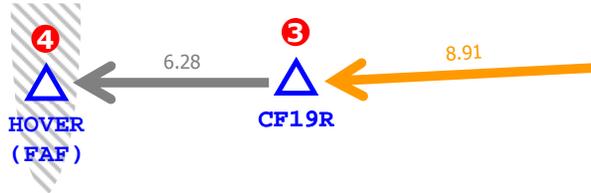


Initial Approach Segment						2	Index
Sub-Segment	FlightPlanWaypointApproach				Course	Leg	
	Type	Mode	Altitude	Target		Distance	
Outbound	3	1	3500 ft	N/A (0')		12.68	Begins at Initial Approach Fix (IAF) - HOVER
45° Turn	3	1	3500 ft	N/A (0')	328°	2.87	328° = FlightPlanWaypointApproachCourse
180° Turn	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

## FlightPlanWaypointApproachIndex 3 and 4:

### Intermediate Approach

FlightPlanActiveApproachWaypoint = 3 and 4 (the Index)  
 FlightPlanWaypointApproachType = 1 (Fix)  
 FlightPlanWaypointApproachMode = 2 (Final)  
 FlightPlanApproachIsWaypointRunway = 0



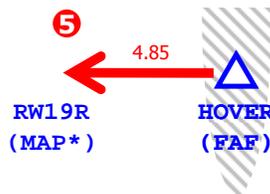
Intermediate Approach Segment 1						3	Index
Segment	FlightPlanWaypointApproach Type	Mode	Altitude	Target	Course	Leg Distance	
Intermediate - 1	1	2	3500 ft	N/A (0')	198°	8.91	Begins at Intermed. Fix (IF) or Inbound to FAF
						8.91 nm	:FlightPlanWaypointApproachLegDistance

Intermediate Approach Segment 2						4	Index
Segment	FlightPlanWaypointApproach Type	Mode	Altitude	Target	Course	Leg Distance	
Intermediate - 2	1	2	3000 ft	N/A (0')	193°	6.28	Ends at Final Approach Fix (FAF)
						6.28 nm	:FlightPlanWaypointApproachLegDistance

## FlightPlanWaypointApproachIndex 5:

### Final Approach

FlightPlanActiveApproachWaypoint = 5 (the Index)  
 FlightPlanWaypointApproachType = 1 (Fix)  
 FlightPlanWaypointApproachMode = 2 (Final)  
 FlightPlanApproachIsWaypointRunway = 1

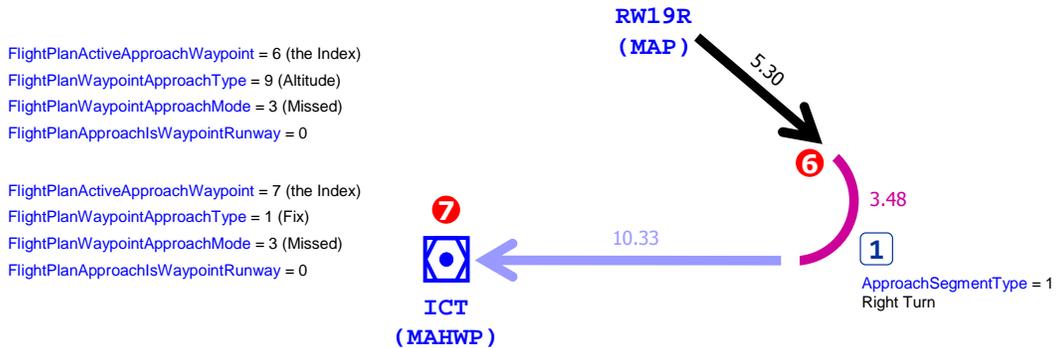


\* The Missed Approach Point

Final Approach Segment						5	Index
Segment	FlightPlanWaypointApproach Type	Mode	Altitude	Target	Course	Leg 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



#### Missed Approach Climb-Out Segment 6 Index

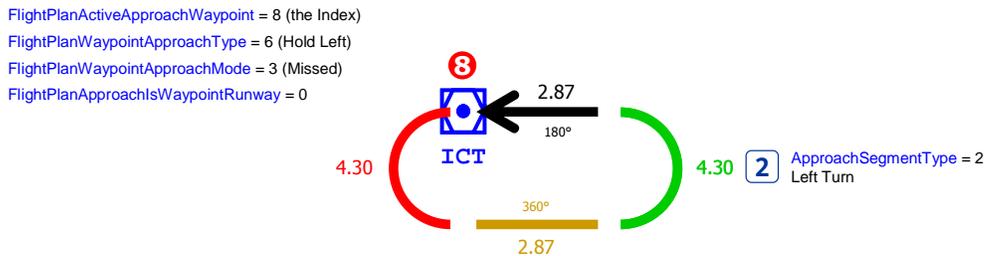
Segment	FlightPlanWaypointApproach					Leg	Description
	Type	Mode	Altitude	Target	Course	Distance	
Straight Climb-out	9	3	3500 ft	3500 ft	193°	5.30	Straight climb to Target altitude
						5.30 nm	:FlightPlanWaypointApproachLegDistance

#### Missed Approach Holding Turn Segment 7 Index

Sub-Segment	FlightPlanWaypointApproach					Leg	Description
	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:

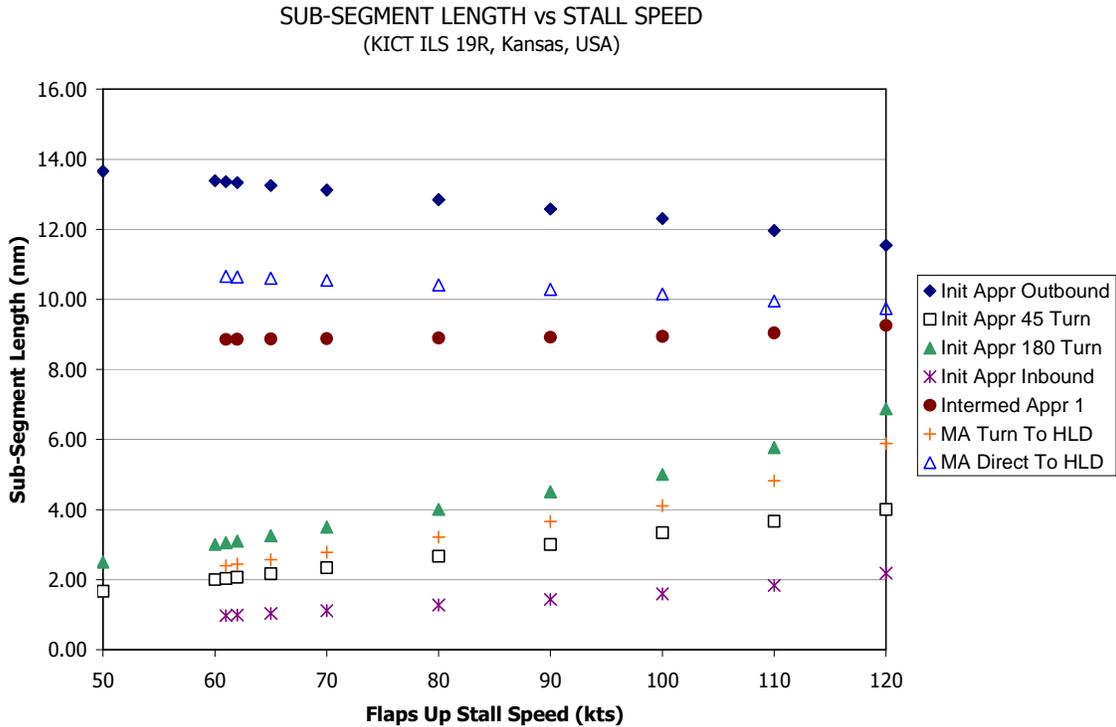
### Holding Pattern



#### Holding Pattern Segment 8 Index

Sub-Segment	FlightPlanWaypointApproach					Leg	Description
	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` 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.

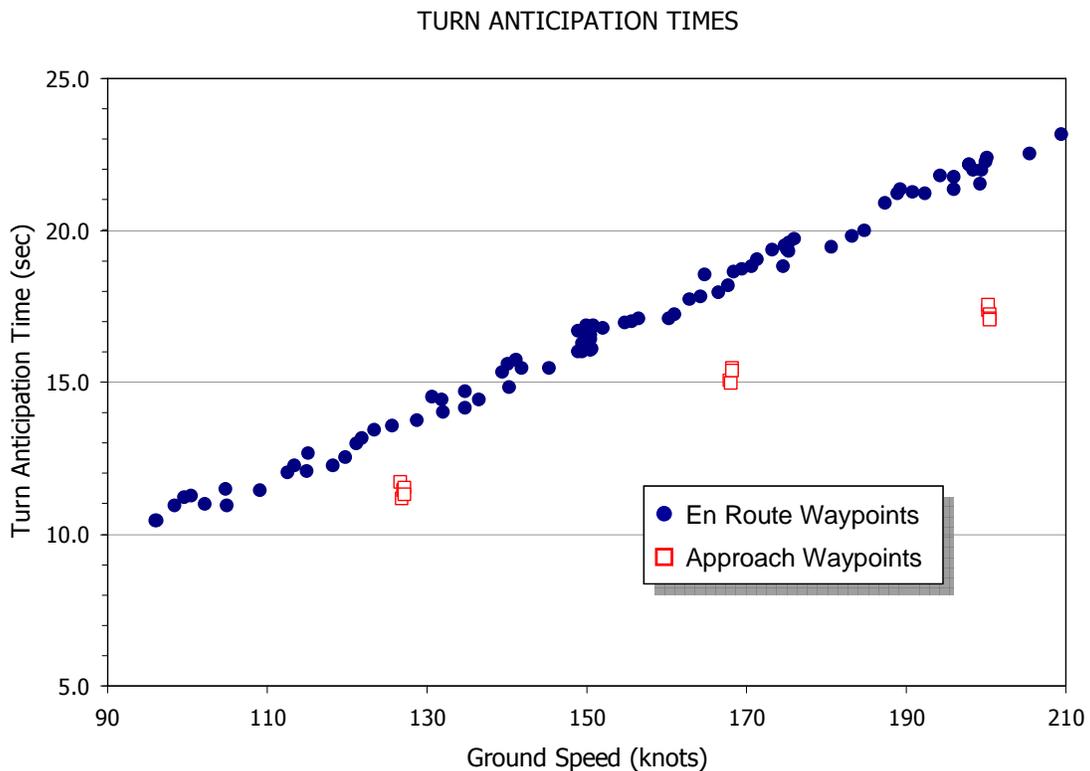
Other segments and some sub-segments are geographically fixed and do not vary by aircraft stall speed. KICT ILS19R examples include the Approach Transition, Intermediate Approach "2", Final Approach, and Missed Approach Straight Climb-out segments.

## 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.

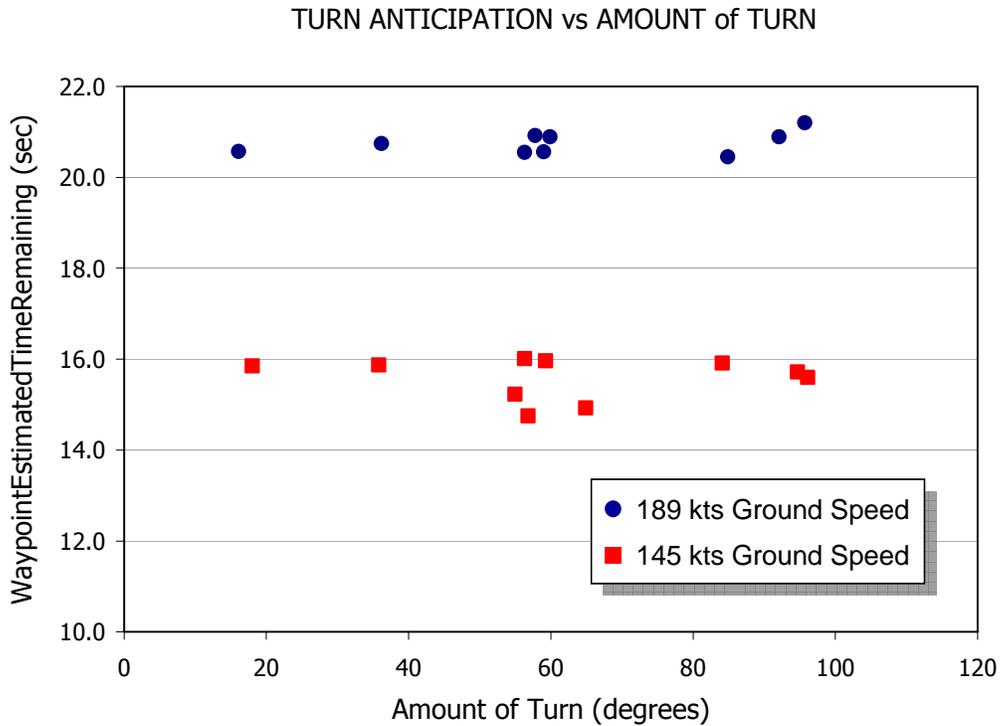


The smaller times result in more precise turns in the Approach phase when the aircraft should be traveling at approach rather than cruise airspeeds. So, closer tolerance on the Fly-Over Waypoints is part of the package.

## TURN ANTICIPATION vs. AMOUNT OF TURN

Does the amount of turn (number of degrees turned) influence Turn Anticipation? Apparently not.

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.

## 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 `IcaoSearchStartCursor = 'VNW'`:

	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:WaypointVorIcao)
(@c:IcaoSearchCurrentIcao) (>@c:WaypointIntersectionIcao)
```

followed by,

```
(@c:WaypointAirportLatitude, degrees)
(@c:WaypointAirportLongitude, degrees)
(@c:WaypointVorLatitude, degrees)
(@c:WaypointVorLongitude, degrees)
(@c:WaypointVorLatitude, degrees)
(@c:WaypointVorLongitude, 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, VOR, 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** = VOR
- **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, VOR, 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 VORs, a City Name appears in parentheses as part of the **FacilityName** (e.g., "HI" VOR = ABATE (PORTLAND)).

❑ **FacilityRegion (string) [Get]**

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

❑ **FacilityLatitude**

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

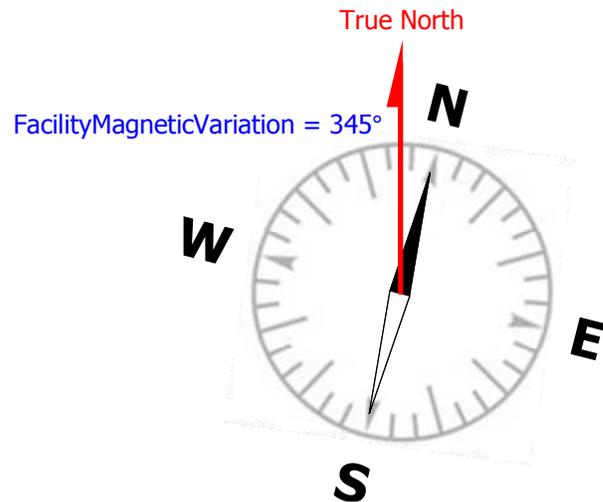
Latitude and Longitude of the Facility.

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

**FacilityMagneticVariation** is the compass direction of true north.

In this example, A:GPS MAGVAR and A:MGVAR would equal 15°(15E).

To derive the magnetic course from a gps var that returns true bearing (which is mostly the case) subtract A:MAGVAR from the true bearing.



## 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:

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

A reminder about Units: GPS variables are sometimes coded without indicating Units, for example, (C:fs9gps:NearestVorCurrentLine). 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 function correctly in fs9gps.dll. Azimuth can be set, but it does not seem to work with [ExtrapolationLatitude](#), [Longitude](#).

#### ❑ **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.

#### ❑ **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](#). I believe it's based on a spherical law of cosines formula rather than Haversine, but I cannot tell which. With Latitude and Longitude expressed in **radians**, the spherical law of cosines formula is:

**Distance =**

$$R * \arccos[\sin(\text{Lat1}) * \sin(\text{Lat2}) + \cos(\text{Lat1}) * \cos(\text{Lat2}) * \cos(\text{Lon2} - \text{Lon1})]$$

[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. [GeoCalcDistance](#) does not consider aircraft altitude or facility elevation and appears to be a sea level measurement with the following assumptions (R is the average Earth radius):

- R = 3438.1158 nautical miles, or
- R = 6367.3902 kilometers, or
- R = 3956.5128 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 =**

$$R * \arccos[\sin(\text{Lat1}/57.2958) * \sin(\text{Lat2}/57.2958) + \cos(\text{Lat1}/57.2958) * \cos(\text{Lat2}/57.2958) * \cos(\text{Lon2}/57.2958 - \text{Lon1}/57.2958)]$$

### ⊗ **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. Although [GeoCalcIsIntersect](#) is included in the gps.dll module, it's not used in the gps\_500 gauge xml code and may 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. At any rate, [GeoCalcIsIntersect](#) and [GeoCalcIntersectionLatitude](#) do not appear to be implemented in fs9gps in the first place.

Perhaps the original intent was for [IntersectionLatitude](#) and [Longitude](#) to represent the coordinates of two intersecting radials defined by [GeoCalcLatitude1](#), [Longitude1](#), [Azimuth1](#), [Length1](#), and [GeoCalcLatitude2](#), [Longitude2](#), [Azimuth2](#), [Length2](#). However, there is only one [Length](#) variable, not two. The [GeoCalcIntersection](#) variables may not function properly at all.

If the reader is interested, math formulas for the intersection of two radials and other aviation formulae can be found in this excellent and well known reference:

#### **"Aviation Formulary V1.46" by Ed Williams**

<http://williams.best.vwh.net/avform.htm>

### ❑ **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.

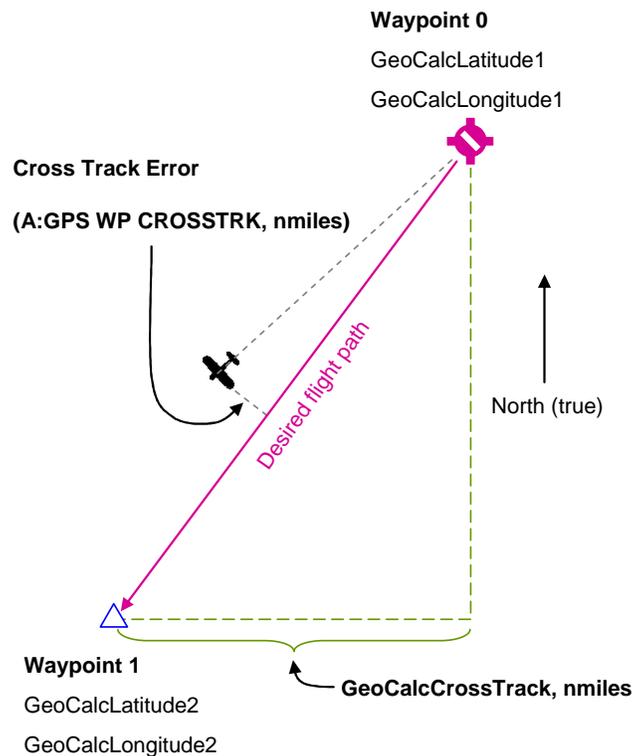
### ❑ **GeoCalcCrossTrack (nmiles) [Get]**

The first thing to note is that [GeoCalcCrossTrack](#) is not the same as navigational Cross Track Error. Cross Track Error is the distance the aircraft is from the desired flight path, measured perpendicular to the desired flight path.

[GeoCalcCrossTrack](#) is the *distance* between [GeoCalcLongitude2](#) and [GeoCalcLongitude1](#), ([Longitude2](#) minus [Longitude1](#)) always measured at [GeoCalcLatitude2](#). See the figure below.

If [Longitude1](#) is to the west of [Longitude2](#), then [CrossTrack](#) is positive distance. If [Longitude1](#) is eastward of [Longitude2](#), then [CrossTrack](#) is negative distance.

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. It's assumed this is a spherical geometry calculation.



# Data Entry and Working with Strings

Entry, manipulation, and storage of string data are regularly required in use of the gps module. String data are more complicated to work with than numbers principally because they are not as easily stored in memory while FS is running. To the point, L:Vars cannot store strings – only numbers.

This section reviews topics related to string entry, manipulation, and storage (apart from the methods used in the stock gps\_500 Garmin GPS 500 gauge) that are pertinent with use of the gps module.

Contents of this Chapter:

## **STRING OPERATORS**

### **ASCII CODE**

### **STRING ENTRY METHODS**

- Keyboard Direct Entry
- Mouse Click Entry Using a Keypad Image

### **CONCATENATION and STRING STORAGE**

- 5 characters maximum can be stored in a single L:Var
- Shift Register
- XMLVars – Custom L:Vars
- String Storage Macros
  - The macros (12 in all)
  - Using the String Storage Macros

### **OTHER**

- Storage in internal registers
- LOGGER (XML > HDD > XML)
- String Storage v2.0.1

## STRING OPERATORS

String operators commonly used with gps XML script:

- ❑ **chr** – converts an ascii number to a character. Example: `65 chr` returns 'A'. Only one number and one character at a time.
- ❑ **ord** – converts a character to an ascii number. Example: `'A' ord` returns 65. Only one character and one number at a time.
- ❑ **slen** – String Length, number of characters. Example: `'ABC D' slen` returns 5.
- ❑ **scat** – concatenates strings. Examples:

`'A' 'B' scat` yields the string AB.

`'A' (C:fs9gps:FlightPlanDestinationAirportIdent) scat` yields the ICAO of the destination airport. (That's 'A' followed by six spaces concatenated with the destination airport Ident).

- ❑ **ssub** – extracts a portion of a string. **ssub** requires three arguments. Example:

`'ABCD123' 2 4 ssub` returns CD12.

The first argument is the string to be evaluated.

The second argument, 2, indicates the position in the string to start looking. The first character is always position 0, so, 2 means start at C.

The third argument indicates the number of characters to return.

## ASCII CODE

ASCII, abbreviated from American Standard Code for Information Interchange, is a character-encoding scheme. 'Originally based on the English alphabet, it encodes specified characters into 7-bit binary integers' as shown by the ASCII chart below (Google search definition).

Char	Ascii Number	Char	Ascii Number	Char	Ascii Number	Char	Ascii Number	Char	Ascii Number
Null	0	3	51	C	67	L	76	U	85
Space	32	4	52	D	68	M	77	V	86
+	43	5	53	E	69	N	78	W	87
,	44	6	54	F	70	O	79	X	88
-	45	7	55	G	71	P	80	Y	89
.	46	8	56	H	72	Q	81	Z	90
0	48	9	57	I	73	R	82		
1	49	A	65	J	74	S	83		
2	50	B	66	K	75	T	84		

Key = "Ascii" returns only Upper Case letters, numbers, and special characters "space" "+" "," "-" and "."

Key = "Alphanumeric" returns only Upper Case letters, numbers, and "space"

Keystroke entry must be made using lower case letters only for both "Acsii" and "Alphanumeric"

As an example, to store the letter 'W' into an L:Var, it must first be converted to its ascii code number equivalent, and then the ascii code is stored.

'W' odr (>L:Letter\_W, enum). Where, (L:Letter\_W, enum) stores the number 87.

## STRING ENTRY METHODS

### Keyboard Direct Entry



When ICAO and Name searches are initiated or when file names are entered for hard drive storage using LOGGER, it may be more convenient to use the computer's keyboard to enter strings rather than use of a mouse to click knob or keyboard images in your gauge to produce string and number entry.

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, bool) 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> blocks.

- **Line 2:** The choices are `Key="Alphanumeric"`, `Key="Ascii"`, or `Key="Backspace"`

Using `Alphanumeric`, lower case alphabet letters, space, and number 0 through 9 keystrokes are accepted as keyboard entry. 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 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/decimal point) are also accepted. If a gauge requires latitude and longitude entry, then `Ascii` is the appropriate choice because it provides the ability to enter the decimal point. Additionally, note that the stock FS9 gps\_500 gauge uses `Ascii` for `NameSearch` entry (see line 3947).

- **Line 3:** The <Visible> tag. This is an especially important toggle that enables or disables keyboard entry via the M:Key instruction. Only when the <Visible> condition is true will code lines that follow the <Visible> statement such as the M:Key line be executed.

- The visibility condition, `(L:KeyEntry, bool) 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, line 4 will be executed whenever there is keyboard entry. While M:Key is enabled, it traps all keyboard entry and you lose the use of normal keyboard assignments such as "G" for Landing Gear toggle.
- As a consequence, you need the ability to turn on and off Line 4 using <Visible>, limiting its use only to the specific situation where you want to enable M:Key direct keyboard entry.

- **Line 4: (M:Key)** traps the keyboard entry result associated with each individual keystroke. When a keystroke occurs while the <Visible> condition is "true",

M:Key returns the ascii code number that is mapped to the specific keyboard character that was pressed. A keyboard does not generate a string character per se, rather, it yields the ascii code equivalent of each keystroke. Refer to the Ascii table above.

In the example above, if Line 4 was written

```
(M:Key) chr (>L:Value, enum)
```

and the letter "W" was typed, then (L:Value, enum) would return 0 because an L:Var cannot store a string.

Alternatively, if Line 4 was written

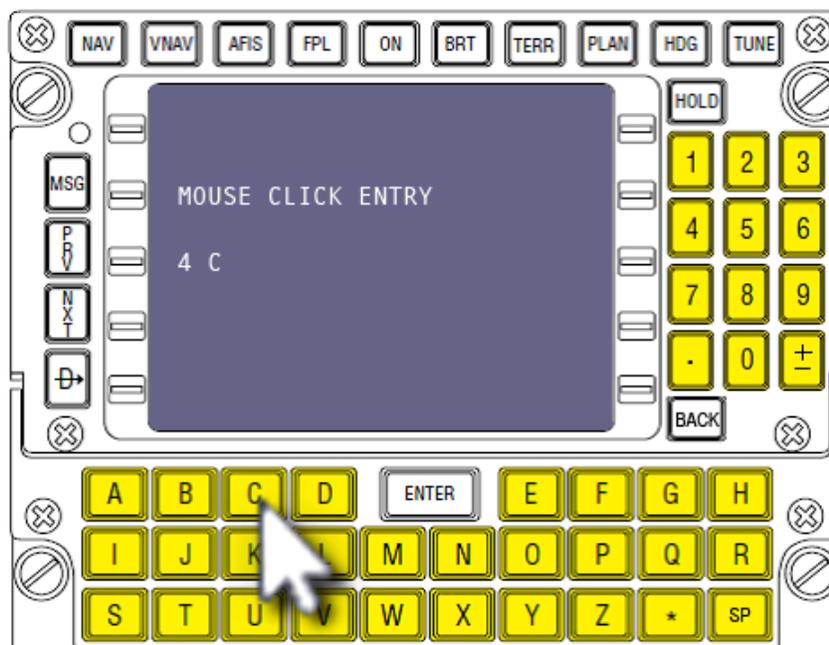
```
(M:Key) (>L:Value, enum)
```

and the letter 'W' was typed, then (L:Value, enum) would return 87.

#### ❑ Mouse Click Entry Using a Keypad Image

Each character is assigned a mouse click area in the gauge as shown by the yellow shading below. When the key image is clicked, the associated ascii number equivalent should be generated and stored into an L:Var or the key character entered directly into a fs9gps string variable that requires input.

The ascii code is returned using the XML **ord** string operator for all characters, both string and number. The example XML shows the <Mouse> instructions for characters "4", and "C".



```

<Area Left="370" Top="106" Width="24" Height="33">
  <Cursor Type="Hand"/>
  <Click>
    '4' ord (>L:MouseEntry, enum)
  </Click>
</Area>

```

```

<Area Left="121" Top="257" Width="35" Height="29">
  <Cursor Type="Hand"/>
  <Click>
    'C' ord (>L:MouseEntry, enum)
  </Click>
</Area>

```

If keypad image entry is limited to [IcaoSearch](#) and [NameSearch](#), the 'C' mouse click might look something like:

```

<Area Left="121" Top="257" Width="35" Height="29">
  <Cursor Type="Hand"/>
  <Click>
    <Visible>(L:IcaoSearchEntry, bool)</Visible>
    'C' (>C:fs9gps:IcaoSearchEnterChar)

    <Visible>(L:NameSearchEntry, bool)</Visible>
    'C' (>C:fs9gps:NameSearchEnterChar)
  </Click>
</Area>

```

## CONCATENATION and STRING STORAGE

**Name** and **Ident** input require that individual keystrokes be combined (concatenated) to form a word and saved. The gps module does this automatically for **IcaoSearch** and **NameSearch** operations which is all that the stock gps\_500 gauge needs. Refer to discussions in the **ICAO Search Group** and **Name Search Group** chapters.

However, if you are building a gauge that requires entry of latitude and longitude, or file names of saved flight plans for example, or any other strings, then you will probably need the ability to concatenate individual key input and/or save the string in memory. There are a few ways to do this.

### ❑ 5 characters maximum can be stored in a single L:Var

Ascii numbers representing individual characters can be aggregated and stored into a single L:Var, but the maximum is 5 characters only per L:Var. This is a good solution for an **Ident**, but insufficient for an **ICAO** or most names (file names, airport names, etc.).

As one example, the **Ident** string CF03R could be stored in an L:Var as follows:

Char	Ascii Number	
C	67	} 6770485182 (>L:Ident, enum)
F	70	
0	48	
3	51	
R	82	

And **ssub** used to extract each character when the **Ident** is needed.

### ❑ Shift Register

Use of a shift register is the traditional XML method used to store multiple alphanumeric keystroke entries.

The direct keyboard entry example below accommodates typing of up to 5 keystrokes, storing the ascii code value of each individual character typed into a separate L:Var, shifting the previous L:Var 'to the left' when the next character is typed. This technique can store as many individual keystroke entries as the user wants as long as a sufficient number of separate L:Vars are included in the shift register script to begin with.

```

1 <Keys>
2   <On Key="Ascii">
3     <Visible> (L:KeyEntry, bool) 1 == </Visible>
4     (L:Entry39_4,enum) (>L:Entry39_5,enum)
5     (L:Entry39_3,enum) (>L:Entry39_4,enum)
6     (L:Entry39_2,enum) (>L:Entry39_3,enum)
7     (L:Entry39_1,enum) (>L:Entry39_2,enum)
8     (M:Key) (>L:Entry39_1,enum)
9   </On>
10 </Keys>

```

- **Lines 1-3:** As before.
- **Lines 4-7:** The "shift register". Before converting the current key entry to ascii and storing that number into L:Entry39\_1 (Line 8), the ascii value of the *previous* keystroke entry which was initially stored in L:Entry39\_1, is stored into L:Entry39\_2 (Line 7). Preceding that (Line 6), the value in L:Entry39\_2 is shifted up to (stored into) L:Entry39\_3, and so forth. This is done because M:Key is always stored into L:Entry39\_1 (Line 8). After 5 keystrokes, the ascii value of first keystroke entered will end up being stored in L:Entry39\_5.
- **Line 8:** M:key returns the ascii value of the current keystroke and that is stored into L:Entry39\_1.
- 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, Entry39\_1 through Entry39\_5.
- A drawback to traditional shift register code is that it can get quite long when accommodating lengthy strings and multiple string variable names.

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

```

1 <Element Name="Entry Box 39>
2   <Position X="775" Y="86"/>
3   <FormattedText X="101" Y="20" Adjust="left"
4     Font="Courier New" Color="#101010" FontSize="10"
5     LineSpacing="10" Bright="Yes">
6     <String>
7       %(
8         (L:Entry39_5,enum) chr
9         (L:Entry39_4,enum) chr scat
10        (L:Entry39_3,enum) chr scat
11        (L:Entry39_2,enum) chr scat
12        (L:Entry39_1,enum) chr scat
13      )%!s!
14     </String>
15   </FormattedText>
16 </Element>

```

The following shows results of typing "a", "b", "space", "Shift+c", "d", ".", "5", "f" using the shift register code above:

Key = "Alphanumeric"

Entry #	Keystroke	L:Vars					Concatenated string
		Entry39_1	Entry39_2	Entry39_3	Entry39_4	Entry39_5	
1	a	65	0	0	0	0	A
2	b	66	65	0	0	0	AB
3	space	32	66	65	0	0	AB then "space"
4	Shift+c	32	66	65	0	0	AB then "space"
5	d	68	32	66	65	0	AB D
6	.	68	32	66	65	0	AB D
7	5	53	68	32	66	65	AB D5
8	f	70	53	68	32	66	B D5F

Key = "Ascii"

Entry #	Keystroke	L:Vars					Concatenated string
		Entry39_1	Entry39_2	Entry39_3	Entry39_4	Entry39_5	
1	a	65	0	0	0	0	A
2	b	66	65	0	0	0	AB
3	space	32	66	65	0	0	AB then "space"
4	Shift+c	32	66	65	0	0	AB then "space"
5	d	68	32	66	65	0	AB D
6	.	46	68	32	66	65	AB D.
7	5	53	46	68	32	66	B D.5
8	f	70	53	46	68	32	D.5F

## ❑ XMLVars – Custom L:Vars

Tom Aguilo's XMLTools XML enhancement module includes the powerful XMLVars class that provides an XML solution for storing strings up to 128 characters in length. Like L:Vars, XMLVars can be read by any gauge in an aircraft panel, can have any name, can hold numeric values of any kind but, unlike L:Vars, can also store strings.

Additionally, because of XMLVars flexibility of naming variables, it can handle standard multidimensional array-style manipulation of data. Data arrays introduce a powerful capability not possible using native L:Vars unless a complex group of macros is applied. XMLTools can be freely downloaded here:

<http://fsdeveloper.com/forum/showthread.php?t=207518>

XMLTools requires installation of the XMLTools.dll module. Complete installation and User Guide instructions are provided.

Some examples of XMLVars use:

```

1 <Macro Name="MakeName">(>C:XMLVARS:StoreVarName, string)</Macro>
2 <Macro Name="FindIndex">(>C:XMLVARS:SearchVarName, string)</Macro>
3 <Macro Name="WriteString">(>C:XMLVARS:StringValue, string)</Macro>
4 <Macro Name="ReadString">(C:XMLVARS:StringValue, string)</Macro>
5 <!-- Find Index, Read String -->
6 <Macro Name="FIRString">
7   (>C:XMLVARS:SearchVarName, string) (C:XMLVARS:StringValue, string)
8 </Macro>
9 <!-- Find Index, Write String -->
10 <Macro Name="FIWString">
11   (>C:XMLVARS:SearchVarName, string) (>C:XMLVARS:StringValue, string)
12 </Macro>
13 <Macro Name="WriteNumber">(>C:XMLVARS:NumberValue, number)</Macro>
14 <Macro Name="ReadNumber">(C:XMLVARS:NumberValue, number)</Macro>
15 <!-- Find Index, Read Number -->
16 <Macro Name="FIRNumber">
17   (>C:XMLVARS:SearchVarName, string) (C:XMLVARS:NumberValue, number)
18 </Macro>
19 <Macro Name="XMLVARBackspace">
20   @1 @FIRString 0 @1 @FIRString slen 1 - ssub @WriteString
21 </Macro>
22 <Macro Name="Reset">1 (>C:XMLVARS:Reset)</Macro>
23 <Macro Name="c">C:fs9gps</Macro>
24 <Macro Name="C">C:fs9gps</Macro>
25
26 <Macro Name="InitTable">
27   'MouseString' @MakeName
28   'CurrentRwy' @MakeName
29   'Name1' @MakeName
30 </Macro>
31
32 <Macro Name="MouseKey">
33   'MouseString' @FIRString 122 chr scat 'MouseString' @FIWString
34 </Macro>
35
36 <Macro Name="Clear2">
37   'CurrentRwy' @FindIndex ' ' @WriteString
38 </Macro>
39
40 <Update Hidden="No">
41   (L:InitSequence,bool) 0 ==
42   if{
43     @InitTable
44     1 (>L:InitSequence,bool)
45   }
46 </Update>

```

**Lines 1 – 22:** Macros associated with XMLVars. My preference is to write a macro whose name is the action performed by each XMLVar function. It is easier for me to read my script that way.

**Lines 26-30:** A macro that is performed only when the gauge first loads. In this case, it creates three XMLVar variable names.

**Lines 32-34:** Macro used to concatenate and store a string entered by mouse click of a keypad image in a gauge.

**Lines 36-38:** A macro that clears 'CurrentRwy' XMLVar by entering a null value ( ' ' ) into it.

**Lines 41-45:** Gauge initiation sequence. Such a sequence is run one time only – immediately after the gauge initially loads.

```
48 <Area Name="LETTER Q" Left="5" Top="5" Width="20" Height="20">
49   <Cursor Type="Hand"/>
50   <Click Kind="LeftSingle">
51     'Q' ord s22 @MouseKey
52   </Click>
53 </Area>
54
55 <Keys>
56   <On Key="Ascii">
57     <Visible>(L:KeyboardEnable, bool)</Visible>
58     'Name1' @FIRString (M:Key) chr scat @WriteString
59   </On>
60 </Keys>
```

**Lines 48-53:** Action taken when the "Q" key is clicked on the keypad image.

**Lines 55-60:** Using XMLVars with keyboard entry. The XMLVar string named 'Name1' is read, the keystroke is converted to a character, concatenated, and re-stored as 'Name1'.

In my opinion, every serious XML gauge developer, especially anyone interested in working with the complex gps module, should consider installing the XMLTools suite.

Tom provides thorough installation instructions and user documentation.

## ❑ String Storage Macros

XML String Storage macros written by Robbie McElrath offer another method to store strings. They can be used to enter, concatenate, store and read strings up to 64 characters long in a "single" L:Var. The XML macros and user instructions are provided below. These macros can be written into the gauge and as such, do not require separate installation of a .dll module.

Behind the scenes, the macros actually construct sixteen L:Vars containing 4 ascii characters each by utilizing the ascii hexadecimal value of each character together with XML's built in Bit Operators (>>, <<, |, &) to keep track of the order.

The user chooses an arbitrary variable name which is attached to the 16 L:Vars as a prefix, uniquely labeling them (name\_[1-4]\_[1-4]). The 16 building block L:Vars are never shown individually, and just one L:Var name (prefix) is entered. Therefore, to the user, the appearance is that a single L:Var holds a string up to 64 characters long.

All ascii characters except **&**, **/**, **\**, **<**, and **'** (single quote) can be accommodated. Backspace and Clear operations are also included.

### The macros (5 are 'Command Macros', 7 are 'administrative')

```
1 <Macro Name="Read4">
2   (@1_4, number) 7 &lt;&lt; 0xFFFFFFFF &amp; (@1_3, number) 21 >> 0x7F &amp; | (>@1_4, number)
3   (@1_3, number) 7 &lt;&lt; 0xFFFFFFFF &amp; (@1_2, number) 21 >> 0x7F &amp; | (>@1_3, number)
4   (@1_2, number) 7 &lt;&lt; 0xFFFFFFFF &amp; (@1_1, number) 21 >> 0x7F &amp; | (>@1_2, number)
5   (@1_1, number) 7 &lt;&lt; 0xFFFFFFFF &amp; (@2, number) @3 0x7F &amp; | (>@1_1, number)
6 </Macro>
7
8 <Macro Name="ReadKB">
9   @Read4(@1_4, @1_3_4, 21 >>)
10  @Read4(@1_3, @1_2_4, 21 >>)
11  @Read4(@1_2, @1_1_4, 21 >>)
12  @Read4(@1_1, M:Key)
13 </Macro>
14
15 <Macro Name="Read4Str">
16  (@1_4, number) 7 &lt;&lt; 0xFFFFFFFF &amp; (@1_3, number) 21 >> 0x7F &amp; | (>@1_4, number)
17  (@1_3, number) 7 &lt;&lt; 0xFFFFFFFF &amp; (@1_2, number) 21 >> 0x7F &amp; | (>@1_3, number)
18  (@1_2, number) 7 &lt;&lt; 0xFFFFFFFF &amp; (@1_1, number) 21 >> 0x7F &amp; | (>@1_2, number)
19  (@1_1, number) 7 &lt;&lt; 0xFFFFFFFF &amp; @2 0x7F &amp; | (>@1_1, number)
20 </Macro>
21
22 <Macro Name="ReadStr">
23  @Read4(@1_4, @1_3_4, 21 >>)
24  @Read4(@1_3, @1_2_4, 21 >>)
25  @Read4(@1_2, @1_1_4, 21 >>)
26  @Read4Str(@1_1, r)
27 </Macro>
28
29 <Macro Name="Backspace4">
30  (@1_1, number) 7 >> (@1_2, number) 0x7F &amp; 21 &lt;&lt;&lt; | (>@1_1, number)
31  (@1_2, number) 7 >> (@1_3, number) 0x7F &amp; 21 &lt;&lt;&lt; | (>@1_2, number)
32  (@1_3, number) 7 >> (@1_4, number) 0x7F &amp; 21 &lt;&lt;&lt; | (>@1_3, number)
33  (@1_4, number) 7 >> (@2_1, number) 0x7F &amp; 21 &lt;&lt;&lt; | (>@1_4, number)
34 </Macro>
```

```

36 <Macro Name="Backspace">
37   @Backspace4(@1_1, @1_2)
38   @Backspace4(@1_2, @1_3)
39   @Backspace4(@1_3, @1_4)
40   @Backspace4(@1_4, L:PleaseBeZero)
41 </Macro>
42
43 <Macro Name="Clear4">
44   0 (>@1_1, number) 0 (>@1_2, number) 0 (>@1_3, number) 0 (>@1_4, number)
45 </Macro>
46
47 <Macro Name="Clear">
48   @Clear4(@1_1) @Clear4(@1_2) @Clear4(@1_3) @Clear4(@1_4)
49 </Macro>
50
51 <Macro Name="ToString1">
52   (@1, number) 21 >> 0x7F &amp; chr
53   (@1, number) 14 >> 0x7F &amp; chr scat
54   (@1, number) 7 >> 0x7F &amp; chr scat
55   (@1, number)      0x7F &amp; chr scat
56 </Macro>
57
58 <Macro Name="ToString4">
59   @ToString1(@1_4) @ToString1(@1_3) @ToString1(@1_2) @ToString1(@1_1) scat scat scat
60 </Macro>
61
62 <Macro Name="ToString">
63   @ToString4(@1_4) @ToString4(@1_3) @ToString4(@1_2) @ToString4(@1_1) scat scat scat
64 </Macro>
65
66 <Macro Name="FromString">
67   :221 d slen 0 > if{ d 0 symb ord @ReadStr(@1) d slen 1 - 1 r ssub g221 }
68 </Macro>

```

## Using the String Storage Macros

Use of the string storage macros is very simple. There are 5 "command" macros:

1. **ReadKB:** Read keyboard (or mouse) entry and concatenate
2. **Backspace:** Backspace one character
3. **Clear:** Clear the string store L:Var values
4. **ToString:** Convert the string store L:Var values to a string
5. **FromString:** Convert a string into the L:Var values

### 1. Converting String Data Into An L:Var

**1.1** Entering one character at a time such as in direct keyboard entry, automatically concatenating as you type. The use of <On Key="Ascii"> will enable keyboard entry of "-" and "." as well as all the other alphanumeric:

- KEYBOARD DIRECT ENTRY XML:

```
<On Key="Ascii"> <!-- Keyboard Entry -->
    <Visible> (L:KeyboardEntryEnabled, bool )</Visible>
    @ReadKB(L:StringLVar1)
</On>

<On Key="Backspace"> <!-- Backspace -->
    <Visible> (L:KeyboardEntryEnabled, bool) </Visible>
    @Backspace(L:StringLVar1)
</On>
```

The user needs to choose an arbitrary L:Var name, such as [StringLVar1](#). No units. The macros will take care of the units.

- MOUSE CLICK ENTRY XML:

Entering data by mouse clicking on an FMS keypad image in your gauge is a simple variation. You need to adjust the M:Key reference in the ReadKB macro. The last line (12) of the ReadKB macro should read @Read4(@1\_1, L:LKey) rather than @Read4(@1\_1, M:Key).

Then, in the <Mouse> section, for a mouse click on the letter "A", for example:

```
<Area Left="25" Top="220" Width="15" Height="15">
    <Cursor Type="Hand"/>
    <Click>
        'A' ord (>L:LKey, enum)
        @ReadKB(L:StringLVar1)
    </Click>
</Area>
```

**1.2** Entering a string of characters all at once, by code . If you want to store something like:

'VED BAM', or (A:NAV1 IDENT, string), then the following can be used:

The XML:

```
@Clear(L:BAM_VOR_ICAO)
'VED BAM' @FromString(L:BAM_VOR_ICAO)
```

or

```
@Clear(L:Nav1Name)
(A:NAV1 IDENT, string) @FromString(L:Nav1Name)
```

## 2. Displaying the 'String' L:Var:

The XML:

```
<String>%(@ToString(L:StringLVar1))%!  
</String>
```

## 3. Passing a 'String' L:Var:

The XML:

```
@ToString(L:BAM_VOR_ICAO) (>C:fs9gps:WaypointVorIcao)
```

## 4. Clearing the 'String' L:Var:

The XML:

```
@Clear(L:StringLVar1)
```

## OTHER

### ❑ Storage in internal registers

Strings can be stored in the internal memory registers (s0 ...s49) with the provision that the registers are cleared each gauge update cycle.

### ❑ LOGGER (XML > HDD > XML)

LOGGER is a module written by Robbie McElrath that enables data (number and string) to be written to and read from a hard drive using XML instructions. LOGGER can be freely downloaded here:

<http://robbiemcelrath.com/fs/logger/about>

Additionally, now LOGGER is included as a Class in XMLTools.

### ❑ String Storage v2.0.1

Doug Dawson also provides a string storage dll module that can be freely downloaded here:

<http://www.douglassdawson.ca/>

## <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.

```
10 <Update Frequency="18" Hidden="No">
11     20 (>C:fs9gps:NearestVorMaximumItems, enum)
12     100 (>C:fs9gps:NearestVorMaximumDistance, nmiles)
13     62 (>C:fs9gps:NearestVorCurrentFilter)
14     (A:PLANE LATITUDE, degrees) (>C:fs9gps:NearestVorCurrentLatitude, degrees)
15     (A:PLANE LONGITUDE, degrees) (>C:fs9gps:NearestVorCurrentLongitude, degrees)
16 </Update>
17
18 <Element Name="NEAREST VOR LOOP DISPLAY">
19     <Position X="10" Y="25" />
20     <FormattedText X="800" Y="800" Font="Courier New" FontSize="12"
21     LineSpacing="12" Color="#101010" Bright="Yes" >
22         <Color Value="blue" />
23         <Color Value="darkgreen" />
24         <String>
25             \{clr2}NEAREST VOR SEARCH\n\n\{clr3}
26             %((C:fs9gps:NearestVorCurrentLatitude, degrees))%!9.4f! :Current Lat
27             %((C:fs9gps:NearestVorMaximumItems, enum))%!5d! :Max Items
28             %((@c:NearestVorItemsNumber))%!4d! :Items Num\n
29             %((C:fs9gps:NearestVorCurrentLongitude, degrees))%!9.4f! :Current Lon
30             %((C:fs9gps:NearestVorMaximumDistance, nmiles))%!5d! :Max Dist
31             %((@c:NearestVorCurrentFilter))%!5d! :Filter
32             \n\n\{clr2}
33             ----- NearestVorCurrent -----\n
34             %      ICAO      111\n
35             Line 123456789012 Ident Type      Freq      Dist      Brg\n
36             %((@c:NearestVorItemsNumber) s2 0 !=)
37             %if}
38                 %0 sp1)
39             %loop}
40                 %(l1 (>@c:NearestVorCurrentLine))
41                 \{clr}%((@c:NearestVorCurrentLine))%!-5d!
42                 %((@c:NearestVorCurrentICAO))%!13s!
43                 %((@c:NearestVorCurrentIdent))%!7s!
44                 %((@c:NearestVorCurrentType))%!5d!
45                 %((@c:NearestVorCurrentFrequency, mhz))%!9.2f!
46                 %((@c:NearestVorCurrentDistance, nmiles))%!8.1f!
47                 %((@c:NearestVorCurrentTrueBearing, degrees))%!6d!\n
48                 %(l1 ++ s1 12 &lt;t;)
49             %next}
50             %end}
51         </String>
52     </FormattedText>
53 </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     80.7      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     86.6      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.

## Panel Reload Gauge

After making changes to XML script in the gauge editing process, the panel must be refreshed before the edited gauge can function.

One convenient way to accomplish this is to build a simple gauge that executes the panel re-load event (>K:RELOAD\_PANELS).

In FSX, "RELOAD\_USER\_AIRCRAFT" is used rather than "RELOAD\_PANELS" which works only in FS9.

```
<Gauge Name="Panel Reload" Version="1.0" >
  <Image Name="Reload.bmp" Bright="Yes" ImageSizes="20,20,0,0"/>
  <Mouse>
    <Area Left="0" Top="0" Width="20" Height="20">
      <Cursor Type="Hand"/>
      <Click>
        (A:Fuel weight per gallon, pounds per gallon) 0 ==
          if{ (>K:RELOAD_USER_AIRCRAFT) } // FSX
          els{ (>K:RELOAD_PANELS) } // FS9
      </Click>
    </Area>
  </Mouse>
</Gauge>
```

Reload.bmp is a 20x20 pixel icon  that can be clicked to initiate the panel reload. It can be positioned anywhere on the screen by using appropriate placement coordinates in panel.cfg.

Note that the variable (A:Fuel weight per gallon, pounds per gallon) returns 6 in FS9 but 0 in FSX. Thus, it can be used to distinguish between the two sims.

## 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. Maybe Lockheed Martin has already taken care of these, plus more.

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

Group	Variable	Bug
FlightPlan	<a href="#">FlightPlanIsActiveWaypoint</a>	It is not Settable (SDK error)
FlightPlan	<a href="#">FlightPlanIsDirectTo</a>	It is not Settable (SDK error)
FlightPlan	<a href="#">FlightPlanActiveWaypoint</a>	It is Settable (SDK error)
FlightPlan	<a href="#">FlightPlanNewApproachAddInitialLeg</a>	MSFT ESP web page <i>suggests</i> this may be Inop (units Unavailable), but it is operational
FlightPlan	<a href="#">FlightPlanWaypointFrequency</a>	Value returned is not a valid VOR frequency. Already noted by MSFT ACES
FlightPlan	<a href="#">FlightPlanWaypointMinAltitude</a>	Units bug - must specify 'meters' to get feet
FlightPlan	<a href="#">FlightPlanApproachSegmentLength</a>	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	<a href="#">FlightPlanWaypointMinAltitude</a>	Database issue - some segments have an obviously incorrect (too low) MEA, such as 0
FlightPlan	<a href="#">FlightPlanWaypointApproachCourse</a>	Some bearings are True but all should be Magnetic
GeoCalc	<a href="#">GeoCalcAzimuth2</a>	Not active in fs9gps
GeoCalc	<a href="#">GeoCalcCrossTrack</a>	Maybe not a bug as much as it is quite misleading
GeoCalc	<a href="#">GeoCalcIsIntersect</a>	Appears Inop
GeoCalc	<a href="#">GeoCalcIntersectionLatitude, Lon</a>	Appears Inop
NearestAirspace	<a href="#">NearestAirspaceCurrentNearDistance</a>	Incorrectly identifies distance with the far, 'occluded' airspaces
WaypointAirport	<a href="#">WaypointAirportRegion</a>	Unnecessary variable. Regions do not exist for Airports
WaypointAirport	<a href="#">WaypointAirportRadarCoverage</a>	Inop
WaypointAirport	<a href="#">WaypointAirportAirspace</a>	Inop
NearestNDB	<a href="#">NearestNdbCurrentFilter</a>	does not exist
Nearest__	<a href="#">Nearest__MaximumDistance</a>	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	<a href="#">WaypointNdbCity</a>	Always returns a blank string
WaypointNDB	<a href="#">WaypointNdbWeatherBroadcast</a>	Inop
WaypointVOR	<a href="#">WaypointVorWeatherBroadcast</a>	Inop
WaypointVOR	<a href="#">WaypointVorCity</a>	Always returns a blank string
WaypointIntersection	<a href="#">WaypointIntersectionCity</a>	Always returns a blank string

# fs9gps Guidebook Updates

## v.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 re: A:PLANE and A:GPS update freq
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

## v.2.0

Page	Edit
many	Added descriptions of FSX-only variables
3	Added page on FS9 vs. FSX
15	Update list of gps operations requiring multiple cycles
46	Corrected <a href="#">RunwayDirection</a> (True, not Mag)
49	Updated <a href="#">FrequencyName</a> list
51	Corrected Runway direction (True, not Mag)
52	Added <a href="#">RunwayLength</a> and <a href="#">Width</a> graphic
60	Corrected discussion of <a href="#">WaypointIntersectionNearestVor</a>
64	Corrected explanation of <a href="#">WaypointNdbMagneticVariation</a>
69	Corrected explanation of <a href="#">WaypointVorMagneticVariation</a>
70	Corrected <a href="#">LongestRunway</a> direction (True, not Mag)

96	Corrected <a href="#">NearestNdbCurrentFrequency</a> variable name
98	Added <a href="#">NearestAirspaceCurrentLongitude</a> variable
128	Corrected definition of <a href="#">FlightPlanDepartureLat</a> and <a href="#">Lon</a>
129	Corrected definition of <a href="#">FlightPlanDestinationLat</a> and <a href="#">Lon</a>
136	Updated Creating a Flight Plan to mention LOGGER
138	Added <a href="#">FlightPlanWaypointIdent</a> custom name explanation
166	Corrected description of <a href="#">FlightPlanWaypointAltitude</a>
166	Added <a href="#">FlightPlanWaypointIdent</a> custom name explanation
168	Added <a href="#">FlightPlanWaypointMinAltitude</a> graphic
181	Corrected list of Approaches into KICT
199	Updated average Earth radius and formula description
202	Expanded chapter on Data Entry Working With Strings
219	Added page on Panel Reload Gauge

## v.2.0.1

Page	Edit
21, 23	v.2.0 had mix of FS9 and FSX. Corrected to just FS9