CoPilot

Database Formats

Laurie J. Davis

Email: ldavis@ieee.org

Website: http://lauriedavis9.tripod.com/copilot/

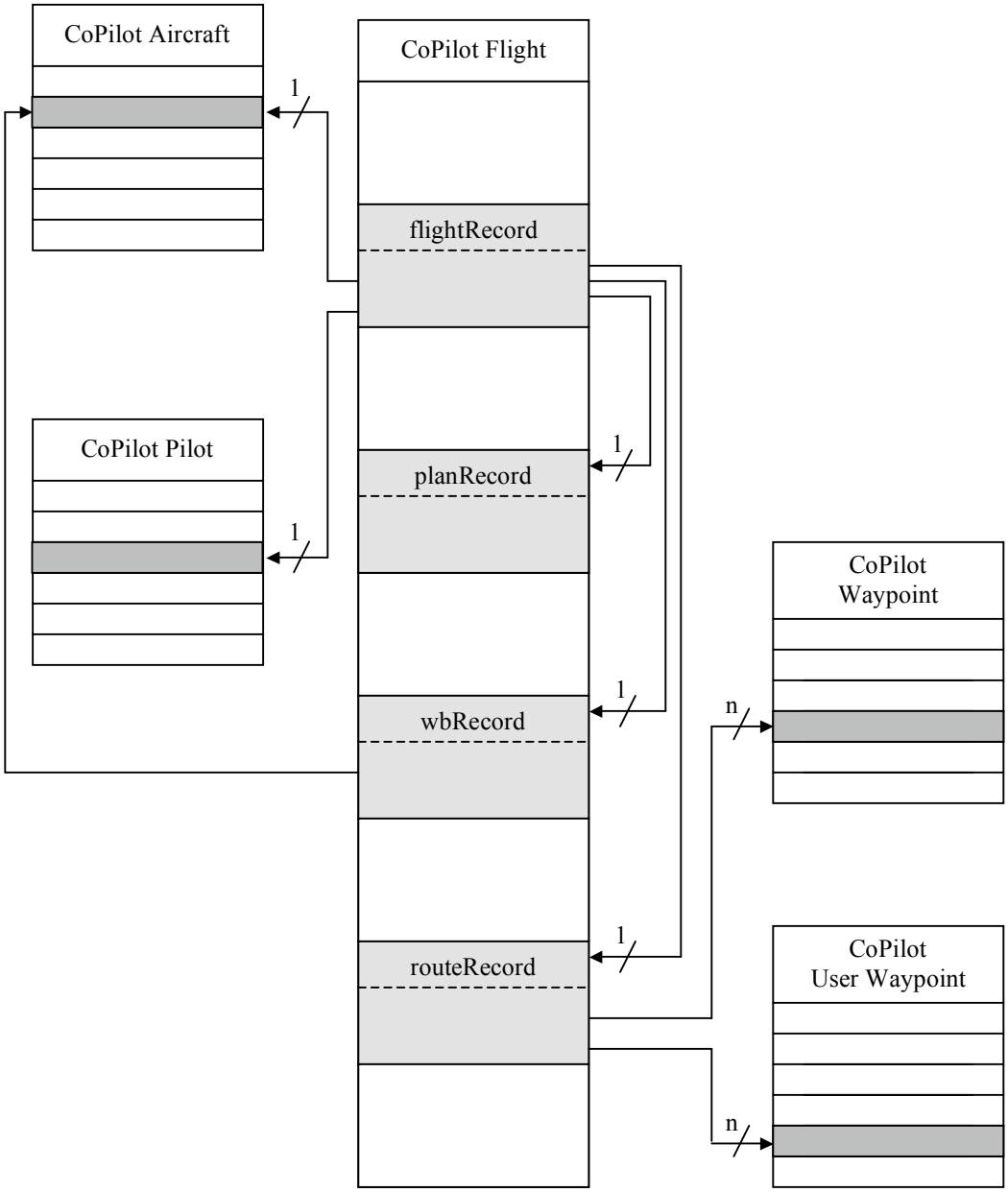
Version 5.3

CoPilot Database Formats

This document describes the format of the various databases used by the CoPilot flight planning program.

General Structure

The flight database “CoPilot Flight” is the main database. It stores all of the information regarding a particular flight by linking to the other databases.



The databases are standard PDB files as described in the following document from Palm:

Palm® File Format Specification – document 3008-003
<http://www.palmos.com/dev/support/docs/>

The PDB headers have the following values:

Name	Type	Creator	Version
CoPilot Flight	'Flgt'	'GXBU'	7
CoPilot Pilot	'pilt'	'GXBU'	3
CoPilot Aircraft	'Airc'	'GXBU'	7
CoPilot Waypoint	'wayp'	'GXBU'	4
CoPilot User Waypoint	'wayu'	'GXBU'	4
CoPilot Prefs	'Pref'	'GXBU'	4

Databases loaded into the Palm device via a HotSync will replace the existing database. There are also database types defined that will cause the database to be appended to the existing database. These database types are 'swpu' instead of 'wayp', 'uwpu' instead of 'wayu', 'arcu' instead of 'Airc', and 'pltu' instead of 'pilt'. The database name can be anything other than the name of the primary database (see list above). When these update databases are loaded, the uniqueId of each record is checked. If a matching uniqueId exists in the primary database, the new record replaces it. If there is no match, the new record is added to the primary database.

Flight Database

The flight database consists of four different record types (flightRecord, routeRecord, wbRecord and planRecord). The record type is indicated by the first word of the database record. The flight records are stored at the start of the flight database followed by the route, wb and plan records. The flight records are sorted by date.

```
enum RecordType {noRecordType, flightRecord = 321, routeRecord,
                 wbRecord, planRecord};
```

Flight Record

```
const UInt16 maxLegs = 20;
const UInt16 maxSegments = maxLegs + 1;
const UInt16 costItems = 7;

struct SegmentTable
{
   UInt32  wb;     // uniqueID of entry in the flight database
   UInt32  plan;   // uniqueID of entry in the flight database
};

struct FlightStruct
{
   RecordType recordType;  // flightRecord
   UInt32  aircraft;  // uniqueID of entry in the aircraft database
   UInt32  pilot;     // uniqueID of entry in the pilot database
   UInt32  route;     // uniqueID of entry in the flight database
   SegmentTable segment[maxSegments];
   UInt16  currentWbSegment;   // currently displayed W&B segment
   UInt16  currentPlanSegment; // currently displayed Plan segment
   UInt32  seconds;   // time (seconds since 1/1/1904)
```

```
   Double  fuelCost;  // negative indicates checkbox should be off
   Double  cost[costItems];     // array of costs
   UInt32  costType[costItems]; // array of costTypes
   UInt32  acDistance;          // Boolean - true to include
                                // aircraft distance amount
   UInt32  acTime;              // Boolean - true to include
                                // aircraft time amount
   UInt32  acTakeoffs;          // Boolean - true to include
                                // aircraft takeoffs amount
   // the rest of the structure is a set of packed strings
   Char    description[];
   Char    note[];
   Char    costText[costItems][];
};
```

Route Record

```
enum WaypointType {system, user};

const UInt16 airwaySize = 12;

struct Waypoint
{
   UInt32  uniqueID;
   WaypointType  wpType;
};

enum Quadrant {north, west, south, east, noIntersection};
enum AirwayType {airw, sid, star, approach};

struct LegStruct
{
   Waypoint waypoint[2];  // uniqueId of the from and to waypoints
   Double     altitude;      // flight altitude in feet
   Int16      windDirection; // wind direction in degrees
   Double     windSpeed;     // wind speed in knots
   Double     pressure;      // pressure in inches of mercury
   Double     temp;          // temperature at altitude
   Char       airway[airwaySize + 1]; // text string for airway
   AirwayType airwayType : 3;  // airway, sid, star or approach
   Boolean    newSegment : 1;  // new segment starts at this leg
   Boolean    refuel: 1;       // valid only for newSegment leg
   UInt16     groundTime;    // minutes, UInt16(-1) = alternate
   Double     intersection;  // intersection (lat or lon) in rads
   Quadrant   quadrant;      // intersection N, E, W, or S
};

struct RouteStruct
{
   RecordType recordType;   // routeEecord
   UInt16     numLegs;      // number of legs used in the route
   LegStruct  leg[maxLegs];
};
```

GroundTime is only valid if the two waypoints are the same. A leg with groundTime = UInt16(-1) indicates a leg that is the start of the route to the alternate destination.

W&B Record

```
struct WbStruct
{
   RecordType recordType;    // wbRecord
   UInt32  aircraftUniqueId;
   Double  weight[];        // size of array depends on aircraft
};
```

Plan Record

```
struct PlanStruct
{
   RecordType recordType;    // routeEecord
   Int32 planType;          // {canadian, american, icao}
   // the rest of this structure is a set of packed strings
   Char  aircraftIdent[];
   Char  flightRules[];
   Char  flightType[];
   Char  number[];
   Char  aircraftType[];
   Char  wakeTurbulance[];
   Char  equipment[];
   Char  departureIdent[];
   Char  departureTime[];
   Char  cruiseSpeed[];
   Char  altitudeRoute[];
   Char  destinationIdent[];
   Char  ete[];
   Char  sarTime[];
   Char  info[];
   Char  rndurance[];
   Char  persons[];
   Char  elt[];
   Char  description[];
   Char  remarks[];
   Char  arrival[];
   Char  sarContact[];
   Char  pilotName[];
   Char  pilotAddress[];
   Char  pilotPhone[];
   Char  pilotLicence[];
   Char  alternateIdent[];
   Char  typeEquip[];
   Char  spare1[];
   Char  homebase[];
   Char  initialAltitude[];
   Char  americanRoute[];
   Char  spare2[];
};
```

Aircraft Database

The aircraft database is sorted by ident.

```
const UInt16  wbLimitPoints = 8;
const UInt16  cgItems = 8;
const UInt16  cruiseItems = 8;
```

```
const UInt16  climbItems = 8;

struct Equip
{
   Boolean  vhf    : 1;
   Boolean  loran  : 1;
   Boolean  dme    : 1;
   Boolean  adf    : 1;
   Boolean  vfrGps : 1;
   Boolean  ils    : 1;
   Boolean  vor    : 1;
   Boolean  modeA  : 1;
   Boolean  modeC  : 1;
   Boolean  spare  : 1;
   Boolean  ifrGps : 1;
   Boolean  rnav   : 1;
   UInt16          : 5;  // force into second word
};

struct Quadratic
{
   Double  a;
   Double  b;
   Double  c;
};

struct AircraftStruct
{
   // general
   UInt32  wake;  // {Light, Medium, Heavy}
   UInt32  elt;   // {none, A, AD, F, AF, AP, P, W, S}
   // cruise performance
   Double  cruiseAlt[cruiseItems];
   Double  cruiseSpd[cruiseItems];
   Quadratic cruise;   // quadratic curve fit, a(x**2) + b(x) + c
                       // where x is the altitude (in feet)
   // equipment
   Equip   equipment;
   // weight & balance 2
   Double  normalWeight[wbLimitPoints];
   Double  normalCofG[wbLimitPoints];
   Double  utilityWeight[wbLimitPoints];
   Double  utilityCofG[wbLimitPoints];
   // fuel data
   Double  cruiseFuel[cruiseItems];
   Quadratic fuel;     // quadratic curve fit, a(x**2) + b(x) + c
                       // where x is the altitude (in feet)
   Double  startTaxiStop;
   Int32   fuelUnits;
   // climb data
   Double  climbTransitionAltitude;  // switch from profile 1 to
                                     //profile 2 at this altitude
   Double  climbAlt[climbItems];
   Double  climbRate[climbItems];
   Double  climbSpeed[climbItems];
   Double  climbFuel[climbItems];
   Quadratic  rate1;   // quadratic curve fit, a(x**2) + b(x) + c
                       // where x is the altitude (in feet)
```

```
                        // quadratic based on ft/hr
   Quadratic  rate2;    // same as rate1
   Quadratic  speed1;   // quadratic curve fit, a(x**2) + b(x) + c
                        // where x is the altitude (in feet)
                        // quadratic based on ft/hr
                        // covert ias to tas
                        // extract horizontal component
   Quadratic  speed2;   // same as speed1
   Quadratic  fuel1;    // quadratic curve fit, a(x**2) + b(x) + c
   Quadratic  fuel2;    // same as fuel1
   // fuel density
   Double  fuelDensity;     // pounds per US gallon
   // cost information
   Double  perHour;
   Double  perNauticalMile;
   Double  perLeg;
   UInt32  fuelMeasurment;  // {climbFuelFlow, climbFuelUsed}
   // weight & balance 1
   Uint32  numStations;
   UInt16  fuelTank[numStations];
   Double  arm[numStations];
   Double  weight[numStations];
   // the rest of the structure is a set of packed strings
   Char    ident[];
   Char    type[];
   Char    desc[];
   Char    homeBase[];
   Char    wbName[numStations][];
};
```

Pilot Database

The pilot database is sorted by last name.

```
struct PilotStruct
{
   // this structure is a set of packed strings
   Char  last[];
   Char  first[];
   Char  licence[];
   Char  phone[];
   Char  address[];
   Char  contact[];
   Char  flightRules;
   Char  flightType[];
   Char  sarTime[];
   Char  numAircraft[];
};
```

Waypoint Database

The waypoint databases are sorted by ident.

```
struct WaypointStruct
{
   Double  latitude;   // latitude in radians (north is positive)
   Double  longitude;  // longitude in radians (west is positive)
```

```
   Float  magVar       // variance in radians (west is positive)
   Float  elevation;   // waypoint elevation in feet (even if
                       // other units are used for display)
   // the rest of the record is a set of zero terminated strings
   Char    ident[];    // max 10 characters PLUS the zero
   Char    name[];     // max 100 characters PLUS the zero
   Char    notes[];    // max 1000 characters PLUS the zero
};
```

Preferences Database

The Preferences database stores the state of the application. It is used to store information that needs to be preserved between invocations of CoPilot.

```
const UInt16 numLastIdents = 7;
const UInt16 costItems = 7;
const UInt16 costSize = 10;
const UInt16 waypointCacheSize = 41;
const UInt16 numActivationCodes = 8;
const UInt16 numActivationFeatures = 20;

struct WpCache
{
   Waypoint waypoint;
   Char     identStr[identSize + 1];
   UInt16   dbRecordNum;
};

struct CopilotPreferences
{
   // flight that is currently selected
   UInt16   currentFlightIndex;   // index of current flight
   // default values for new flight
   UInt32   defaultAircraft;      // uniqueId
   UInt32   defaultPilot;         // uniqueId
   Int32    defaultPlanType;      // {canadian, american, icao}
   // default quadrants
   UInt16   lastMagVar;           // 0 = west, 1 = east
   UInt16   lastLongitude;        // 0 = west, 1 = east
   UInt16   lastLatitude;         // 0 = north, 1 = south
   // units that have been selected
   UInt16   distanceUnits;        // {nm, mi, km}
   UInt16   altitudeUnits;        // {ft, m}
   UInt16   pressureUnits;        // {inHg, hPa}
   UInt16   weightUnits;          // {lb, kg}
   UInt16   armUnits;             // {in, cm, metre}
   UInt16   speedUnits;           // {nm, mi, km}
   UInt16   climbRateUnits;       // {ftPm, ftPs, mPm, mPs}
   UInt16   fuelDensityUnits;     // {lbPgalUS, lbPgalImp, kgPL}
   // only show the preferences form once (until OK button)
   UInt16   preferencesShown;     // true if prefs have been Oked
   // keep track of most popular initial character in idents
   Char     initialChar;          // default initial ident char
   Char     lastIdents[numLastIdents];  // init char of last idents
   // save values in crosswind calculator
   Int16    windDirection;        // last value in crosswind calc
   Int16    windSpeed;            // last value in crosswind calc
```

```
Int16    runway;               // last value in crosswind calc
// save values in true airspeed and density altitude calculator
Double   altitude;             // last value in da/tas calc
Double   pressure;             // last value in da/tas calc
Double   temperature;          // last value in da/tas calc
Double   airspeed;             // last value in da/tas calc
// save the unit conversion selection in the calculator
UInt16   calculatorUnits;      // last calculator conversion
// save the state of true/indicated in da/tas calculator
Int16    trueOrIndicated;      // 0 = true, 1 = indicated
// save the last calculator result
Double   displayResult;        // last calculator result
// save default cost strings and types
Char     costString[costItems][costSize + 1]; //
Int16    costType[costItems];  // {empty, perNm, perMi, perKm,
         // perHour, perTakeoff, perFlight, percentSubtotal}
// default values for cost selections
Int16    acTime;               // 0 = not selected, 1 = selected
Int16    acDistance;           // 0 = not selected, 1 = selected
Int16    acTakeoffs;           // 0 = not selected, 1 = selected
// default value for fuel cost
Double   fuelCost;             // negative = not selected
// what to display in each column of the route form
UInt16   routeColumn[4];       // {distance, magTrack,
         // magHeading, eTime, trueTrack, cruiseTAS, cruiseGS,
         // altitude, pressureAlt, densityAlt, fuel, airway,
         // cumulativeETE, wind, temperature, pressure}
// printing
Int16    printSoftware;        // {clipboard, memo,
                               // palmPrintSW,tealPrintSW}
Int16    printDescription;     // 0 = no print, 1 = print
Int16    printRoute;           // 0 = no print, 1 = print
Int16    printWB;              // 0 = no print, 1 = print
Int16    printFlightPlan;      // 0 = no print, 1 = print
Int16    printWaypoints;       // 0 = no print, 1 = print
// default waypoint database to use
UInt16   currentWaypointDB;    // 0 = system, 1 = user
// save values entered with the “Previous” button in Leg Edit
Double   previousAltitude;
Int16    previousWindDirection;
Double   previousWindSpeed;
Double   previousPressure;
Double   previousTemperature;
Char     previousAirway[airwaySize + 1];
// save values entered in winds aloft calculation
Double   trueAirspeed;
Double   groundSpeed;
Double   magHeading;
Double   magTrack;
// preserve the waypoint cache
WpCache  waypointCache[waypointCacheSize];
UInt16   nextCacheIndex;       // the next location to be used
// zulu time
Int32    zuluOffset;           // not used for OS 4.0+
// airway type
UInt16   previousAirwayType;
// sort order (last type selected)
UInt16   recentSort;
```

```
   UInt16   flightSort;
   // feature activation codes
   UInt16 activationCode[numActivationCodes];
   UInt16 featureActivated[numActivationFeatures];
   // units for fuel summary form
   UInt16 fuelSummaryUnits1;
   UInt16 fuelSummaryUnits2;
   // unique identifier for external waypoint file
   UInt32 externalWaypointIdentifier;
};
```

The waypoint cache stores all of the most recently used waypoints. The cache is refreshed in the background when the user selects a new flight. If the user then quickly switches to a new application, the waypoint cache refresh may not be complete, i.e. all of the waypoints used in the current flight may not be in the cache. This is not an issue for CoPilot, since the cache refresh will complete when CoPilot is next invoked, but other users of the cache cannot assume that all of the current waypoints are in the cache.