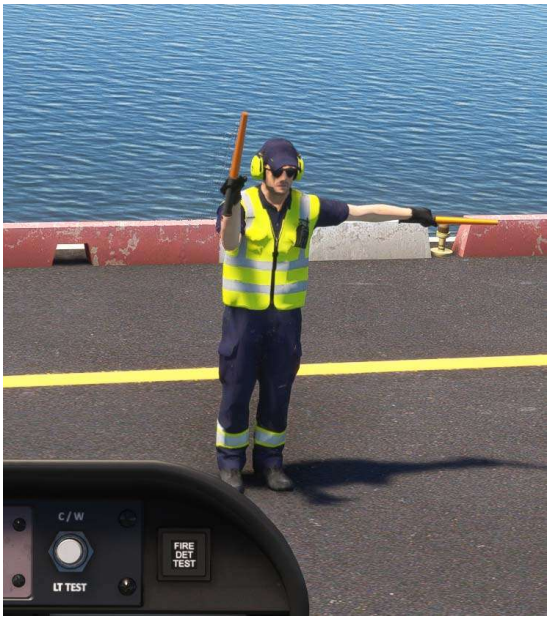
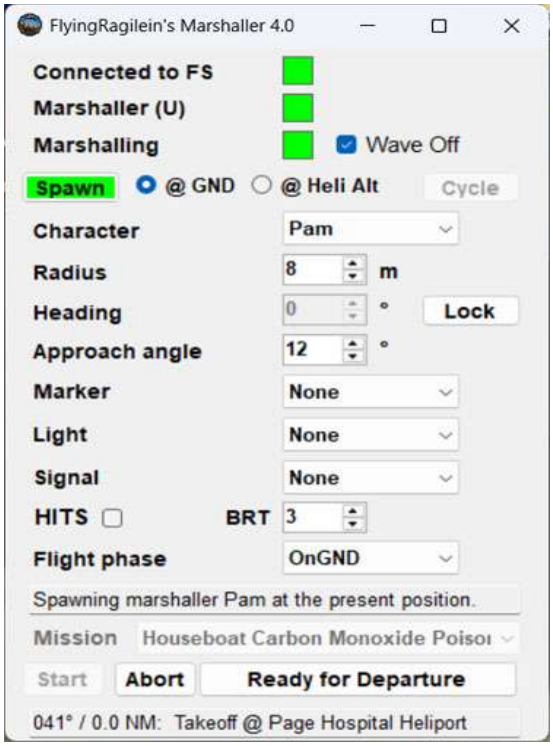
1. Introduction

This add-on provides an animated reactive helicopter Marshaller and a mission framework for FS2020 and FS2024. It consists of two parts:

- A fully animated Marshaller, which shows up in the simulator,



- And a controller app that controls the Marshaller and the mission framework



2. End User License Agreement (EULA)

This software is provided entirely "AS IS". You use this software at your own risk. I give no warranties and deny any liability.

This software is my intellectual property. Publishing or sharing any part of it without my prior written consent, or reverse engineering, is strictly prohibited.

3. Licensing

You can use a subset of the Marshaller's features for free.

To unlock all premium functions on one computer, please give a donation of $8 or more to my project at https://buymeacoffee.com/flyingragilein or https://www.paypal.com/donate?hosted_button_id=H7TJDSACDPZD2

To obtain a license file, please send the computer-based ID displayed during startup of the controller app and a reference to your donation to

gottfriedrazek@yahoo.com

I will send you a license file via email as soon as I can. Please check your spam folder. There are no refunds.

To activate your license, copy the license file into the directory “FlyingRagileins Marshaller App”.

4. Support

For support, questions, and suggestions, please email me at

gottfriedrazek@yahoo.com

I will reply as soon as I can.

If you like the Marshaller, please pass the word! If you encounter any problems, please email me, and I’ll do my best to find a solution.

For a version history, see https://flyingragilein.com/HeliMarshaller

5. Installation

This software consists of two parts: A Marshaller SimObject that shows up in the simulator, and a control app that controls the Marshaller and the mission framework. Both need to be installed.

To install the Marshaller SimObject, copy the directory “flyingragilein-heli-Marshaller” from the zip file to the “Community” directory of your flight simulator. If you have FS2020 and FS2024, copy it to both “Community” directories.

To install the controller app, copy the directory “FlyingRagileins Marshaller App” from the zip file to any location allowing program execution. Additionally, you can copy a shortcut to “FlyingRagileins Marshaller App.exe” to your desktop for convenience (don’t copy the exe, because it needs other files from the directory, but only a shortcut!).

The mission framework uses Windows Text-to-Speech to play all radio communications. Please ensure that the voices “English (United States)” and “English (United Kingdom)” are installed on your computer. You can install voices via Windows Settings > Time & language > Language & region > Add a language. On my computer, installation took about 30 minutes, so it may take some time.

If an add-on comes with a mission file for my Marshaller, copy it into the directory “FlyingRagileins Marshaller App\Marshallers”. A mission file (ending in .json) can contain Marshaller placements and missions.

6. Startup

Start the controller app and the simulator. The order doesn’t matter, as the controller app waits until the simulator is running.

When both programs are running and connected, you see a green indication at “Connected to FS”.

7. Spawning a Marshaller

You can spawn a Marshaller everywhere and at any time by clicking the “Spawn” button. After spawning a Marshaller, you see a green indication at “Marshaller available”.

If you spawn a Marshaller while you are on the ground, the Marshaller spawns at your present position. You can spawn at ground level or at your present altitude (radio buttons). Ground level is preferred, except for positions above ground level, like oil platforms.

If you spawn a Marshaller while you are airborne, the Marshaller spawns at the nearest helipad of the nearest airport within 5000 meters. In this case, you can spawn at ground level or at pad altitude (radio buttons). Again, ground level is preferred, except for pads above ground level. If the airport has multiple helipads, you can cycle through them via the “Cycle” button.

If you spawn a Marshaller while you are on the ground, the Marshaller aligns with your helicopter for takeoff. Once you depart, the Marshaller begins to align with the wind for landing. If you spawn a Marshaller while you are airborne, the Marshaller immediately aligns to the wind.

When you spawn a Marshaller, any other Marshaller within 500 m is removed to prevent conflicts and performance issues. Manually spawned Marshallers exist, as long as the controller app is running. If you want to place Marshallers permanently, you can do so via JSON files (see chapter 12).

8. Takeoff & Departure

After spawning, the Marshaller shows the “start engine” signal if the engine isn’t already running. Otherwise, he stands idle.

Once you add a little collective, the Marshaller performs the “move upward” signal. After you climb to about 65 ft, the Marshaller performs the “depart” signal, and above 130 ft, he returns to standing idle.

When the Marshaller performs a marshalling signal, you see a green indication at “Marshalling”.

9. Approach & Landing

As soon as you are within 500 m of the Marshaller and below 130 ft, the Marshaller performs the “face me” signal. Below 65 ft, he starts marshalling you to the pad. During this time, he performs one of the “move down”, “move left”, “move right”, “move ahead” or “move back” signals. The strength of the first three signals depends on your deviation from the approach profile.

The default vertical profile angle is 12°. You can change the angle by the “Approach angle” control.

If your approach becomes unsafe (sink rate too high, sink rate vs airspeed too high, or speed vs altitude too high), the Marshaller waves you off. The status bar at the bottom of the controller app shows the reason for the wave off, and the app background turns red. In this case, you have to start the approach anew. You can disable this feature by unchecking “Wave Off”.

As soon as you are in a position for landing, the Marshaller performs the “land” signal.

After landing, he performs the “cut engines” signal and returns to standing idle. Then you can repeat the cycle by adding a little collective.

Tip: The Marshaller also looks great at night with his orange glowing marshalling wands.

10. Customizing your Landing Zone

You can customize your landing zone in many ways:

With the “Character” control, you can select a Marshaller character.

With the “Radius” control, you can select the radius at which the Marshaller rotates around the landing spot.

With the “Lock” button, you can lock the heading of the Marshaller and set a heading with the “Heading” control.

With the “Marker” control, you can select landing zone markers (“None”, “Cone”, “Padlight D&N”, “Padlight N”, “Stick”, or “VS-21”). Four markers mark the landing zone, and a fifth marker is placed slightly upwind.

With the “Light” control, you can select additional lighting (“None” or “Flashlight”). The additional lighting only shows up at night.

With the “Signal control”, you can select a signal (“None”, “Flare” or “Smoke Flare”). Signals greatly facilitate the identification of landing zones in missions.

For the currently displayed Marshaller, you can change all settings at any time.

11. HITS (Highway In The Sky)

The HITS displays green frames that guide you to the pad during final approach from about 500 ft AGL. Clicking “HITS” turns the HITS on/off. The path of the frames depends on the heading of the Marshaller and the currently selected vertical profile angle. You can adjust the brightness of the HITS frames by the “BRT” control.



12. Placing Marshallers via JSON Files

Placed Marshallers spawn automatically, as soon as you come within 5000 m of them. This enables Marshallers in missions, where the pilot searches for the mission site and therefore can’t spawn the Marshaller via the “Spawn” button.

Placing Marshallers is very simple and doesn’t require the scenery SDK. Marshallers are placed by providing Marshaller files in simple JSON format. The following “flyingragilein-heli-Marshaller.json” is already in the directory and shows the possible parameters for placing Marshallers. All optional parameters are premium functions and are ignored in the free version. Exception: The parameter “Character”, which is essential for missions.

```
{
  // this sample shows the placing of 3 Marshallers

  "Marshallers":                // one root-element “Marshallers”
   [                            // an array […] of any number of Marshallers

    // Marshaller Mike at Newark Liberty (KEWR) Helipad 1 with minimum parameters
    {
      "Lat": 40.7044147309674,   // latitude in degrees
      "Lon": -74.1680740385394,  // longitude in degrees
    },

    // Marshaller Pam at a meadow in Central Park, New York
    {
      "Lat": 40.7716827311327,
      "Lon": -73.9757390926147,
      "Character": "Pam",        // optional Marshaller character; default: "Mike"
      "Radius": 15.0,            // optional radius of Marshaller; default: 8.0
      "ApproachAngle": 10.0,     // optional approach angle in degrees; default: 12.0
      "LockedHeading": 70.0,     // optional heading in degrees; default: aligns with wind
      "Marker": "Cone",          // optional LZ marker; default: no marker
      “Signal”: “Smoke Flare”,   // optional signal; default: no signal
      “Light”: “Flashlight”      // optional light; default: no light
    },

    // Marshaller Mia at the rooftop helipad VOLC5 in Paris (FS2024 only)
    {
      "Lat": 48.8893489845186,
      "Lon": 2.23900770075355,
      "Character": "Mia",
      "Alt": 234.921             // optional altitude of elevated pad in meters;
                                 // default: Marshaller snaps to the ground
    }
  ]
}
```

To install a Marshaller file, copy it into the directory “FlyingRagileins Marshaller App\Marshallers”. To prevent conflicts, add-on creators shall name their Marshaller files exactly like their package – i.e. “company-package.json”. To prevent conflicts between Marshallers, their minimum distance is 500 m. When a Marshaller is placed, it removes all Marshallers within 500 m. The Marshaller files are processed in alphabetical order. Therefore, Marshallers in files beginning with “z” may remove Marshallers in files beginning with alphabetically lower letters. If a placed Marshaller doesn’t show up, please check the log file in “c\Users\<User>\AppData\Local\FlyingRagilein\MarshallerApp”.

That’s it. Happy landings & have fun with your new Marshaller!

13. Missions

The mission framework guides you through missions, verifies your progress, and plays realistic radio communications. Each mission consists of a series of tasks, such as “Takeoff”, “Land”, “Overfly”, “WinchRescue”, and “Search”. As soon as you complete a task, the framework advances to the next one until the mission is accomplished.

The missions are provided as JSON files. Copy all mission files into the FlyingRagileins Marshaller App\Marshallers directory. You can easily adapt these files or author your own missions (see Chapter 14).

The Mission dropdown lists all missions that start *at your current parking position*. Select a mission and click “Start” to begin. Click “Abort” to abort a running mission.

The large multifunction button lets you communicate certain mission-specific messages, such as “Ready for Departure” or “In Sight”. You'll hear a chime when the button becomes enabled.

The field at the bottom shows the heading and distance to the waypoint associated with the current task, except during “Search” tasks. This avoids lengthy radio calls to communicate mission coordinates.

The mission framework also generates a flight plan for each mission in:

C:\Users\<User>\AppData\Local\FlyingRagilein\FlightPlans

The flight plans are written when you start the controller app. To ensure that the flight plans are available in the simulator, start the controller app first.

Personally, I find it more interesting to search for mission locations visually, although using the GPS is probably more realistic. The choice is yours. The simulator renames user waypoints in avionics displays, and the framework shows renamed user waypoints in (...).

Although the radio communications may mention actions to perform on the GPS, the framework cannot perform these actions automatically.

14. Authoring Missions

This chapter is of a technical nature and is only intended for users who want to author their own missions or modify existing missions. We will cover a lot of ground here, and I'll try to keep it as short and painless as possible.

14.1 Mission Files

Missions are defined by JSON files, a simple data format readable by both humans and computers.

To prevent naming conflicts, authors must name their mission files exactly like their package – i.e., "company-package.json".

You only need to understand two JSON structures to author missions:

- Objects {...} – Used to group information. An object can contain any number of properties written as "name": "value" pairs – e.g., a task object:
  {"ID": "TakeoffFromPage", "WaypointID": "3AZ3", ...}
- Arrays [...] – Used for lists. An array contains a comma-separated list of items of the same kind – e.g., an array of task objects:
  [{"ID": "TakeoffFromPage"}, {"ID": "OverflyRockCreekBay"}, ...]

Objects and arrays can be nested inside each other as required.

That is all you need to know about JSON to get started! If you want to learn more about JSON, you can visit https://www.w3schools.com/js/js_json.asp

The best idea is to open the mission file "flyingragilein-glenn-canyon-missions.json" from my "Glenn Canyon Missions" package now. It contains many comments to help guide you. Here, I'll give you some high-level information that helps you understand what you see.

Each mission file consists of a JSON object with three array properties: "Marshallers", "Waypoints", and "Missions".

- "Marshallers" holds the marshallers used in the missions (see Chapter 12),
- "Waypoints" holds the waypoints used in the missions, and
- "Missions" holds the missions.

14.2 Waypoints

The property “Waypoints” holds an array of waypoints. Waypoints define locations where mission tasks happen.

Each waypoint must have the following properties:

- A “WaypointType” (“HospitalHeliport”, “RescueSite”, or “Location”),
- Coordinates “Lat” (latitude in degrees) and “Lon” (longitude in degrees),
- an “ID”, which is used to reference the waypoint in tasks,
- a “Name”, which is used to refer to the location in radio communications, and
- a “RadioType” (“Uncontrolled” or “None”).

Additionally, a waypoint can have an “InsightName” which is used for in-sight reports, like “smoke flare” that produces “report smoke flare” or “smoke flare in sight”.

If the “RadioType” is “Uncontrolled”, a waypoint also must have a “Callsign” and a “Frequency”.

A waypoint becomes invisible to the user if it is used in a “Search” task. Invisible waypoints don’t show up in flight plans, and no navigation advice is provided for them.

Departure and destination waypoints must have a property “Alt” (altitude in feet) which is used in flight plans.

14.3 Missions

A mission is something the user selects to do within the mission framework. Most missions involve searching for and retrieving a patient from a rescue site and transferring him to a hospital.

The property “Mission” holds an array of missions.

Each mission has a property “Name” and a property “HeliCallsign” that defines the helicopter callsign used in the mission. A mission also holds a briefing dialog (property “Briefing”).

Dialogs are used in many places and define radio communications. Each dialog is an array of lines (remember: enclosed in [...]). Each line is a string in the form of “<Speaker>: <Spoken Text>”. <Speaker> can be: “Pilot”, “Disp” (dispatch), “Medic”, “Doc”, or “Ground” (ground station). <Spoken Text> is arbitrary text and can contain so-called tokens like [Heli.Callsign]. The framework contains many tokens, and they are very powerful. Tokens are either replaced by text or perform an action. For instance, the token [Heli.Callsign] will be replaced by the actual helicopter callsign. And the token [Pause(3000)] will pause the spoken dialog for 3000 milliseconds. Tokens are also enclosed in […] but are not arrays.

Furthermore, each mission holds an array of tasks, which the user has to perform (property “Tasks”).

14.4 Tasks

Tasks define what the user has to do to accomplish a mission.

Each task has a property “TaskClass” that defines what the user has to do to complete the task – e.g., “Takeoff”, “Land”, “Overfly”, “Arrival”, “WinchRescue”, or “Search”. The property “WaypointID” references the waypoint where the task happens. Each task usually contains multiple steps with associated dialogs that guide the user through the task. When the user completes a task, the framework advances to the next task. After completing the last task, the mission is completed.

“Takeoff” tasks have a property “HasPatientPickup” (default: false). If you set it to true, the task will perform a patient pickup. “Land” tasks have a property “HasPatientDropoff”, (default: false). If you set it to true, the task will perform a patient dropoff. WinchRescue tasks always perform a patient pickup. Patient pickups and patient dropoffs maintain a mission-level flag “PatientIsOnBoard”. This flag can be used to generate different dialog variants (more on this later).

Usually the active task begins to play dialogs as soon as the user reaches a certain distance to the tasks waypoint. By adding an “Arrival” task before a “Land”, “WinchRescue”, or “Search” task, the user must overfly the waypoint of the “Arrival” task first. An “Arrival” task differs from an “Overfly” task in that it does not play its own position reporting. Instead, the position reporting is incorporated into the following task's arrival dialog, resulting in a single, compact check-in dialog.

If you want to author a mission using only the default dialogs, you are done. Define some waypoints, add some tasks to the mission, and you are good to test-fly. And I would recommend doing this now, before digging deeper! If the mission doesn’t do what it is supposed to do, check the log file in “c\Users\<User>\AppData\Local\FlyingRagilein\MarshallerApp” for error messages. Probably some JSON typos. Been there, done that. When you start a mission, the framework writes the whole mission to the log file, including raw and expanded versions of all dialogs. This is great for debugging.

14.5 The simplest mission possible

Here is the simplest mission possible. Just two waypoints. A flight from a hospital to a rescue site and back. You can use this sample as a basis for your own first mission. The framework provides default dialogs for all tasks, so you don't have to define every radio exchange yourself. The waypoints are close to each other, making this mission quick to test-fly. You can read the mission definition from top to bottom. No complex GUI is involved, and no programming is required. So you can focus on what matters: Designing and authoring your missions.

```
{
 // a very simple test mission with 2 waypoints and 4 tasks resulting in 2 flights

 "Waypoints":
 [
  // array of waypoints
  {
     "WaypointType": "HospitalHeliport",    // this is a hospital waypoint
     "Lat": 37.52910312004555,              // latitude in degrees
     "Lon": -110.71878639545493,            // longitude in degrees
     "ID": "K3DG0",                         // ID for referencing this waypoint in tasks
     "Name": "Bullfrog Clinic",             // name of this waypoint used in dialogs
     "RadioType": "Uncontrolled",           // the radio type (has a radio and is uncontrolled)
     "Callsign": "Bullfrog Clinic Helipad", // the radio callsign
     "Frequency": "123.05"                  // the frequency
  },
  {
     "WaypointType": "RescueSite",          // this is a rescue site waypoint
     "Lat": 37.47217454420279,
     "Lon": -110.7126419339948,
     "ID": "ASITE",
     "Name": "Accident Site",
     "RadioType": "Uncontrolled",
     "Callsign": "Ranger 6",
     "Frequency": "122.9"
  }
 ],

 "Marshallers":
 [
  // array of marshallers

  // Rita at waypoint ASITE
  {
     "Lat": 37.47217454420279,
     "Lon": -110.7126419339948,
     "Character": "Rita",
     "Marker": "Cone",
  }
 ],
```

```
"Missions":
  [
    // array of missions (here just one mission)
    {
       "Name": "Simple Mission Sample",      // name that shows up in mission selection
       "HeliCallsign": "SkyCare 9",          // callsign of helicopter used in dialogs
       "Briefing":                           // the mission briefing
       [
          // array of dialog lines in form "<Speaker>: <Text>"
          "Disp: [Heli.Callsign], This is a simple test mission",
             // [Heli.Callsign] will be replaced by "SkyCare 9"
          "Pilot: Copy the simple test mission, [Heli.Callsign].",
          "Disp: [Heli.Callsign], Advise when airborne.",
          "Pilot: Will advise airborne, [Heli.Callsign]."
       ],
       "Tasks":
       [
          // array of 4 tasks
          {
             // takeoff from K3DG0
             "ID": "TakeoffFromK3DG0",        // arbitrary task ID of your choice
             "TaskClassID": "Takeoff",        // this is a "Takeoff" task
             "WaypointID": "K3DG0"            // this task happens at waypoint with ID "K3DG0"
          },
          {
             // landing at ASITE
             "ID": "LandAtASITE",
             "TaskClassID": "Land",           // this is a "Land" task
             "WaypointID": "ASITE"            // this task happens at waypoint with ID "ASITE"
          },
          {
             // takeoff from ASITE
             "ID": "TakeoffFromASITE",
             "TaskClassID": "Takeoff",
             "WaypointID": "ASITE",
             "HasPatientPickup": true         // this task picks up a patient
          },
          {
             // landing at K3DG0
             "ID": "LandAtK3DG0",
             "TaskClassID": "Land",
             "WaypointID": "K3DG0",
             "HasPatientDropoff": true,       // this task drops off the patient
             "DialogMods":
             [
                // dialog modifications for this task (covered in chapter 14.7)
                {
                   "ID": "PreAlert",          // modifies the dialog "PreAlert"
                   "PropertyMods":
                   [
                      // set "PatientInfo"
                      "PatientInfo: We have a 19 year old male with chest and abdominal trauma
following a traffic accident. Patient is on the vent with stable vital signs."
                   ]
                }
             ]
          }
       ]
    }
  ]
}
```

14.6 Default Dialogs

After your first mission is running, you might want to modify the default dialogs. Before you can do that, you need to know a few things about the default dialogs. Instead of producing tons of documentation that tends to get outdated, I made the default dialogs self-documenting by exposing them as a JSON file. So, now is the time to open the file “FlyingRagileins Marshaller App\TaskClasses.json”. You *should not modify* this file, as this would affect all other missions.

In TaskClasses.json, you see a property “TaskClasses” that holds all task classes of the framework. Each task class has an “ID”, like “Takeoff”, “Landing”, etc., and some other properties.

The property “Properties” holds properties of the task class, such as: ["BeginDepartureDist: 0.2", ...]. This defines that the dialog “BeginDeparture” is played when the helicopter is 0.2 nautical miles from the task's waypoint.

The property “Dialogs” holds the dialogs of the task class. Each dialog has an “ID”, “Properties” (we will talk about dialog properties in the next chapter), and “Variants”. Each variant has “Patterns” and “Lines”.

The structure of “Lines” is the same as for the “Briefing” dialog, which we already covered.

You might notice that “Lines” is not a direct property of “Dialogs”, but that there is an intermediate property, “Variants”. Each variant has a property "Patterns" and a property "Lines". Variants enable the generation of different dialogs depending on your concrete mission definition. The general rule is that all variants play that have a matching pattern. A pattern matches if all the values given in the pattern match – e.g., the pattern {"WaypointType": "RescueSite", "WaypointIsVisible": true} matches if the waypoint is a rescue site and the waypoint is visible. Variants without a pattern always play. Furthermore, all variants play in their specified order.

14.7 Modifying predefined properties

The easiest way to modify dialogs is to modify their predefined properties. This is a simple yet very effective way to make your dialogs unique.

If you look at the dialog “BeginDeparture” in the task class “Takeoff”, you can see a token: [PatientInfo]. This token correspond to the property “PatientInfo”, which is empty by default. You can provide a value for this token by adding a “DialogMods” property to your task.

A “DialogMods” property holds an array of dialog modifications for the task. Each dialog modification applies to the default dialog with the matching “ID”. To modify the properties of a dialog, add a property “PropertyMods” to its dialog modification and set its value to an array of property modifications, like: […, "PatientInfo: We've got the patient on board. She's unresponsive with stable vitals and receives high-flow oxygen.",...]. This replaces the token [PatientInfo] in the dialog with the given text.

You can also modify the distances at which dialogs are played by adding a property “PropertyMods” at the task level. This property can hold an array of property modifications, like: ["BeginArrivalDist: 2.0", ...]. This changes the distance at which the “BeginArrival” dialog is played to 2.0 nautical miles from the task’s waypoint.

And now it is time to open the file “flyingragilein-glenn-canyon-missions.json” to see how this all works and maybe make your first dialog modification!

14.8 Modifying Dialog Lines

You've followed me so far, excellent! If you want to take complete control of the dialogs in your own missions – read on!

You can either add lines before and/or after the existing lines of a default dialog or override its lines completely.

To add lines before a default dialog, add a property “PrependLines” to its dialog modification and set its value to an array of dialog lines.

To add lines after a default dialog, add a property AppendLines”.

To replace the lines of a default dialog completely, add a property “ReplaceLines”. This gives you complete control over the dialog, but also requires you to specify a complete dialog.

The mission file “flyingragilein-glenn-canyon-missions.json” contains an example of “AppendLines”.

14.9 Triggered Dialogs

Triggered dialogs offer a powerful way to define additional dialogs that play between tasks. To define triggered dialogs, add a property “TriggeredDialogs” to your mission object and set its value to an array of triggered dialogs.

Triggered dialogs have a property “ID”, a property “Trigger” that defines *when* the dialog is played, and a property “Lines” that holds the dialog lines.

Each trigger has the following properties:

- “Type” specifies the type of the trigger. Currently, the only type is “Distance”.
- CompletedTaskID” specifies a task that must have been completed before the dialog will play.

A trigger of type “Distance” has the following additional properties:

- WaypointID” specifies a waypoint.
- “Value” specifies a distance from the waypoint at which the dialog is played.
- “Reference” specifies if the dialog is played when flying to the waypoint (“ToWaypoint”) or from the waypoint (“FromWaypoint”).

Each triggered dialog only plays once during a mission.

The mission file “flyingragilein-glenn-canyon-missions.json” contains several good examples of triggered dialogs.

That’s it ;) If you want to author a mission first add your waypoints and tasks keeping all the defaults. Fly the missions with the defaults and then modify everything to your needs. Usuallly just a few modifications are required to make a mission unique. Good luck and have fun!