# **FMS Implementation Checklist: FlightPlanManager**

This checklist outlines the necessary development tasks to complete the FlightPlanManager module and fully integrate it into the FMS project structure.

### **1. Core State Management**

*This section focuses on enabling the manager to track and handle the "active" flight plan that the FMS is currently using.*

* [ ] **Add Active Plan Attribute:** In FlightPlanManager.\_\_init\_\_, add attributes to hold the active plan and its current state (e.g., self.active\_plan = None, self.current\_leg\_index = 0).
* [ ] **Implement set\_active\_plan:** Create a method that accepts a FlightPlan object and sets it as self.active\_plan. This method should reset the leg index to 0.
* [ ] **Implement clear\_active\_plan:** Create a method to clear the active plan and reset the state variables.

### **2. Live Navigation Interface for Simulink**

*These are the critical methods the MATLAB bridge will call repeatedly during simulation to get real-time navigation guidance data.*

* [ ] **Implement get\_current\_leg:** Create a method that returns the start and end waypoints for the current flight leg based on self.current\_leg\_index.
* [ ] **Implement get\_next\_waypoint:** Create a method to return the FlightPlanWaypoint object for the upcoming waypoint (the destination of the current leg).
* [ ] **Implement advance\_to\_next\_leg:** Create a method that increments self.current\_leg\_index. This will be called by the system when a waypoint passage is detected.
* [ ] **Add End-of-Route Handling:** Ensure the navigation methods (get\_next\_waypoint, advance\_to\_next\_leg) handle the case where the end of the flight plan is reached.

### **3. Advanced Flight Planning Features**

*This section covers enhancing the flight plan's capabilities beyond a simple waypoint list.*

* [ ] **Implement Plan Modification Methods:**
  + [ ] insert\_waypoint(wp\_id, position): Inserts a new waypoint into the active plan at a specific index.
  + [ ] delete\_waypoint(position): Removes a waypoint from the active plan.
  + [ ] modify\_waypoint(position, new\_altitude): Changes data for an existing waypoint (e.g., crossing altitude).
* [ ] **Integrate Airways:** Modify create\_flight\_plan to expand airway identifiers (e.g., "J121") into a sequence of waypoints by querying the navigation database.
* [ ] **Integrate Procedures:** Add methods to fetch and append SIDs (Standard Instrument Departures) and STARs (Standard Terminal Arrival Routes) to the flight plan from the procedures database.

### **4. Module and System Integration**

*This section focuses on connecting the FlightPlanManager to the rest of the FMS ecosystem.*

* [ ] **Update matlab\_python\_bridge.py:**
  + [ ] Import and instantiate the FlightPlanManager.
  + [ ] Refactor existing bridge functions to delegate all flight planning calls to the FlightPlanManager instance.
  + [ ] Add new bridge functions to expose the live navigation methods (get\_next\_waypoint, etc.) to MATLAB.
* [ ] **Connect route\_optimizer.py:**
  + [ ] Create a method in FlightPlanManager called optimize\_active\_plan().
  + [ ] This method should pass self.active\_plan to the optimizer and update it with the returned, optimized route.
* [ ] **Update MATLAB GUIs:** The Flight\_Plan\_Entry.mlapp and FMS\_Control\_Panel.mlapp need to be updated to call the new bridge functions that interact with the manager.
* [ ] **Update Simulink Model:** The Guidance\_Law.slx and Navigation\_Calculations.slx models need to be configured to call the bridge functions for live navigation data on each simulation step.

### **5. Robustness and Testing**

*This section covers professional-grade improvements to ensure the code is reliable and maintainable.*

* [ ] **Implement Logging:** Replace all print() statements for errors and status updates with a proper logging framework (like Python's built-in logging module).
* [ ] **Create Formal Unit Tests:** In tests/unit\_tests/, create a dedicated test file (e.g., test\_flight\_plan\_manager.py) using pytest.
  + [ ] Write tests for all public methods of FlightPlanManager.
  + [ ] Include tests for edge cases (e.g., empty routes, not-found waypoints, loading corrupted files).
* [ ] **Create Integration Tests:** In tests/integration\_tests/, create a MATLAB test script to verify the full loop: MATLAB GUI -> Bridge -> FlightPlanManager -> Navigation Database and back.