3.2 Objectives and Acceptance Criteria

3.2.1 Pre-DLL Build

Inspect existing code. If the code has been migrated from one platform to another or from one language to another, a detailed inspection of conditional statements, equations and the use of brackets/parentheses should be performed.

Recommendation
The code should be evaluated for integrators or areas where a number is added/subtracted to itself. This should be modified to use a value from the previous time step. The previous time step value is then set to the current value in the PostDynamicUpdateCalculation function. It is suggested that the above technique is used to model integrators, regardless as to what other approaches are used for the convergence loop.

Any inconsistencies/anomalies should be corrected and then documented for debrief purposes. It is recommended that the code inspection process is formally signed off as being accepted.

The sign off should be undertaken by at least a Principal Engineer.

[redacted] may wish to establish a standard process for this as the implementation of the code and the subsequent j2 Developer item underpins all future work.

3.2.2 Post DLL Build

A further check of the complete code should be performed to check that all variables required for input and output have been included in the Developer Description. Do all parameters have the correct Unit Type and Unit Scale. Is the Counter included in the output?

Ensure model structures have been created correctly for aerodynamics & engines. Any calculations that are contained in code that is not in the model are included in the Calculate method.

Are all relevant files included for Source, Headers, Assemblies, and References? The DLL builds. Test the DLL is in the correct format using the TestDeveloperLoad utility.

Any inconsistencies/anomalies should be corrected and then documented for debrief purposes. It is recommended that the code inspection process is formally signed off as being accepted.

The sign off should be undertaken by at least a Principal Engineer.

Checking of the j2 Developer item as part of the initial model check and data mapping process is critical as this underpins the entire process.

3.3 Building the DLL

3.3.1 Process Summary

In building the DLL, there are several stages that need to be performed and checked prior to moving onto the next step. These are:

- Code Inspection

  Code should be full investigated prior to the compilation into a Windows DLL. This investigation will identify the interface boundary, I/O parameters, and integrators/filters.

- Generate Developer Item DLL

  The main body of the code is combined with template interface files and compiled into a Windows™ Dynamic Link Library (DLL). This DLL will be linked through an additional interface component right into the j2 environment.

  This Interface DLL can be created directly from a template and addresses the DLL generated from the original code.

- Test the DLL

  This is the process of running the test utility to establish that the compiled DLL will be recognised by the j2 Universal Tool-Kit before attempting to attach it to a model.

3.3.2 Code Inspection

The process rules and guidelines for a full and complete code inspection of the original code are contained in the [redacted] Process document ...]

In addition to the formal [redacted] code inspection, there are some additional process steps that are required to enable the development of the DLL and the subsequent integration with the j2 Universal Tool-Kit to progress correctly. These are:

1. Understand the Code, identify where the Interface breakpoint is to be.
2. Identify Inputs/Outputs (number, name, units)
   a. Aerodynamic Outputs should include gross coefficients, major contributions to coefficients, and data table independent variables.
   b. Engine Outputs should include total thrust, thrust in each axis, and thrust contributions, torque and torque contributions, engine speeds, propeller speeds, temperatures and pressures, mass flow, propwash speeds, and engine coefficients.
3. Document all Inputs and Outputs and generate the code needed to add them into the I/O arrays.

Recommendation
An output from the DLL should be added that includes the DLL version number. This can then be tracked within analyses for further quality control checks.

4. Axis System. Consider what Axis System the Coefficients are in and what Axis System(s) for the input parameters are required.

Note
The aircraft model should be set to the same Axis System as the Coefficients.

5. Identify and document Objects/Structural Items that will be required to generate input values for the code. This can include control surfaces, internal equipment, locations etc.

6. Check for integrators/filters.
    a. Update code to move integrator loop to setting the previous value in a new function. This will be called by the `PostUpdateCalculations` method in j2. This approach will not impact on the operation of the simulator and it has been agreed that as such it will remain in the code for the simulator to maintain consistency between both environments.
    b. Parameters that are adding to themselves need to be updated to add to their value from the previous time step. The previous time step values are then updated in a separate "`PostUpdate`" function at the end of the calculation.
7. Convert all C files (*.c) to C++(*.cpp).
8. "Sign Off" the code inspection process and documentation.

3.3.3 Generate Developer Item DLL

Following the code inspection, the next stage is to generate a Windows DLL that is able to integrate into the model as a Developer Item. The stages involved include:

1. Create a new VS2017 Project using the[redacted] ircraft Developer Item template.
2. Add the aircraft (aerodynamics and engine) Header Files (*.h), Source Files (*.cpp), and table files (*.gh) into the project. (§3.1.4)
3. Include the headers containing the aero and engine data structures into the DeveloperDescription.h (§3.1.4.1)
4. Update the `DeveloperDescription Constructor` Definition to include references to the new data structures in DeveloperDescription.h (§3.1.4.1)
5. Update the `DeveloperDescription Constructor` Implementation to include the new data structures in DeveloperDescription.cpp (§3.1.4.6)
6. From the documented list of Inputs and Outputs, create the Inputs and Outputs arrays within the `Developer Description Constructor`. (§3.1.4.6)
7. Include the headers containing the aero and engine data structures in DeveloperInterface.h (§3.1.4.2)
8. Add the static references to the aerodynamic and engine structures in DeveloperInterface.h (§3.1.4.2)
9. Add the headers for the[redacted] external functions to be called into DeveloperInterface.cpp (§3.1.4.7)
10. Add the external global structures for aerodynamic and engine data into DeveloperInterface.cpp (§3.1.4.7)
11. Update the `DeveloperInterface Constructor` to pass the pointers to the aerodynamic and engine structures to the `DeveloperDescription Constructor`. (§3.1.4.7)
12. Update the `DeveloperInterface Destructor` to delete the aerodynamic and engine structures. (§3.1.4.7)
13. Update the `DeveloperInterface Reset` method to assign the global structure pointers to the `DeveloperInterface` static pointer values. (§3.1.5)
14. Update the `DeveloperInterface Reset` method to call the initialisation functions that only need to be called once within the `firstPass` condition. (§3.1.5)
15. Update the `DeveloperInterface Reset` method to initialise any values that are reset at the beginning of each analysis. (§3.1.5)
16. Update the `DeveloperInterface Reset` method to initialise the previous time step integrator values to the current values. (§3.1.5)

17. Update the `DeveloperInterface PostDynamicUpdateCalculation` method to initialise the previous time step integrator values to the current values. (§0)
18. Update the `DeveloperInterface Calculate` method perform any calculations that are required outside the Main Calculation. (§3.1.7)
19. Within the Main Calculation Step in the `DeveloperInterface Calculate` method (when all inputs have converged) add the function calls to the main calculations within the [redacted] code. (§3.1.7)
20. Within the Main Calculation Step in the `DeveloperInterface Calculate` method (when all inputs have converged) and if the analysis is Steady State (`m_Trimming==true`) add the function calls to the post calculation functions that set the values from the previous time steps to the current values within the [redacted] code. (§3.1.7)
21. Build the DLL
22. Move the DLL to the C:\j2Aircraft\Developer folder on the server.
23. Run the Test Load Utility and check that the DLL is recognised as a j2 Developer Item.

Recommendation:
When the DLL is updated, change aircraft model description to latest DLL date. This will update the aircraft model version number and avoid mixing data sets that use different DLLs.

Recommendation
Ensure that the DLL is finalized before starting the J2 matching process. The code should be reviewed by multiple people since any issues can result in many, many hours lost.