IADS Helicopter

Custom Derived Function

January 2013

SYMVIONICS Document SSD-IADS-046

Created: August 15, 2009

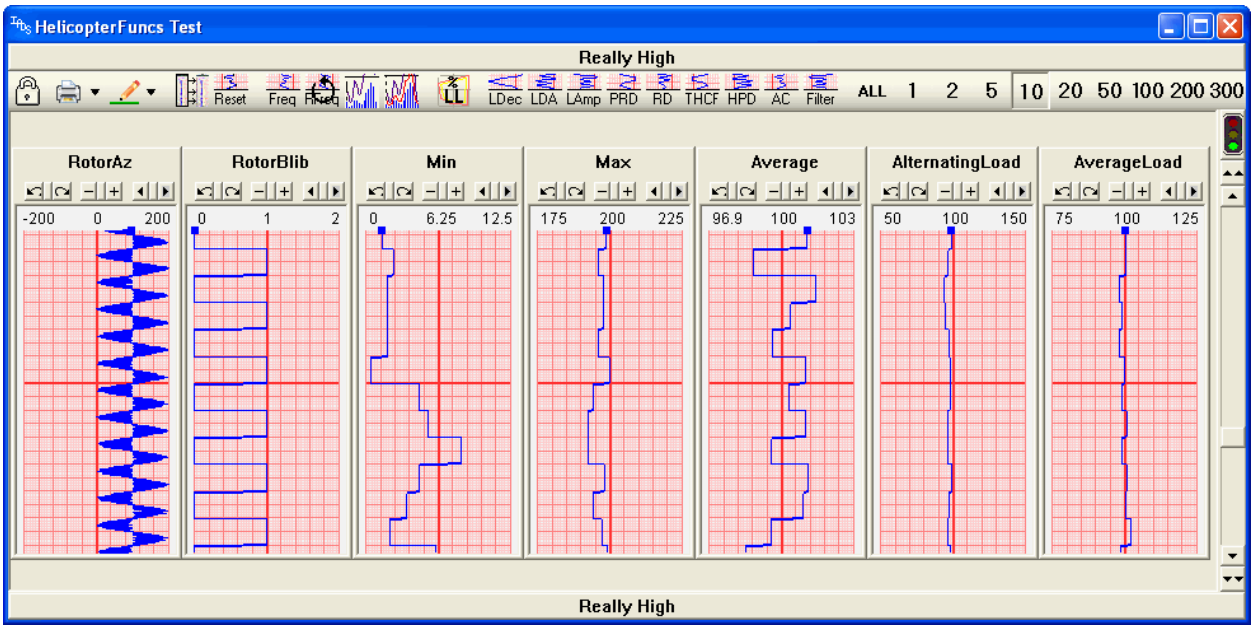
Table of Contents

1. Introduction

This document describes the IADS Helicopter custom derived function, developed to assist in analysis of rotary machines.

1.1. Overview

The IADS Helicopter custom derived function is provided as a Dynamic Link Library (DLL) which contains the five internal functions used in a majority of helicopter testing programs. These internal functions are created in the IADS Configuration Tool and accessed as derived parameters.



2. Installation

Note: *All of the helicopter functions are contained within the "HelicopterFuncs" DLL.*

To register the HelicopterFuncs.dll:

1) Copy the HelicopterFuncs.dll file to a known directory. If you're unsure where to place the it, you might consider the C:\Program Files\IADS directory.
2) In Windows Explorer, navigate to the directory where you copied the file.
3) Right-click on the HelicopterFuncs.dll file and choose **Open With…**
4) Click the **Browse** button.
5) Browse to the **C:\Windows\System32** directory.
6) Select the **regsvr32.exe** file and then click the **Open** button.

*Note: If you select **Always use the selected program to open this kind of file** checkbox in the Open With dialog, from this point forward just double click on any other custom COM dll and it will register.*

7) Click **OK**.

If applicable, the IADS Operator Console can be used to distribute the DLL and register these functions at all workstations.

Once complete, these functions are now registered and are available for use within the IADS derived equation engine (please see the IADS On-line Help System for additional information using derived equations). The following table shows the function names (Program IDs) that are used for reference when creating derived functions:

ProgId	Description
HelicopterFuncs.Minimum	Calculate the minimum for all points during a user defined cycle
HelicopterFuncs.Maximum	Calculate the maximum for all points during a user defined cycle
HelicopterFuncs.Average	Calculate the average for all points during a user defined cycle
HelicopterFuncs.AlternatingLoad	Calculate the alternating load for all points of a user defined cycle
HelicopterFuncs.AverageLoad	Calculate the average load for all points of a user defined cycle

Table 2-1 HelicopterFuncs DLL Installed Functions

3. Instructions for Use

This section will explain how to create the derived functions within IADS. Figure 3-1 shows all of the functions used to create the IADS derived parameters in the Configuration Tool.

Generally, it is recommended that you run these functions as "IAP" parameters and compute them in the real time environment on a server. At this point in time, the functions do have some issues while jumping randomly around in time as they are based off of previous data. The function will need to be upgraded to make them fully compliant in a post test “random access” environment. Until that point in time, it is recommended that you process these parameters as “IAP” types on a server PC in real-time. This will alleviate any of the random access issues and will also have other benefits (as you will read below).

If you do not run the parameters as IAP types (i.e. DataSourceType = Derived), you may notice some strange results while scrolling back, jumping around in time, or flipping between tabs of strip charts. For the most part it will be perfectly fine for use in testing as the values will show the correct results after a short amount of recovery time. If these data issues are of immediate concern, please do run the parameters as IAP types. For more information on running the parameters as IAP types, please consult the help manual.

Note: *For quick setup, simply run and test all of your equations as type "Derived" (try to refrain from scroll back usage) and then transition to type "IAP" before flight test or data playback tests.*

To create a derived parameter that uses a helicopter function:

1) In IADS, on the Dashboard click the **Configuration** button.
2) Open the **Data** folder, select the **Parameter Defaults** table.
3) Copy and paste an existing row of data for a parameter that is similar to the one you are creating.
4) Enter a unique parameter name in the *Parameter* column.
5) In the *DataSourceType* column, select **Derived** *(see note above*).
6) Enter the desired helicopter function in the *DataSourceArgument* column (see Figure 3-1).
7) Click the **Save** button.
8) Repeat as necessary for each function.

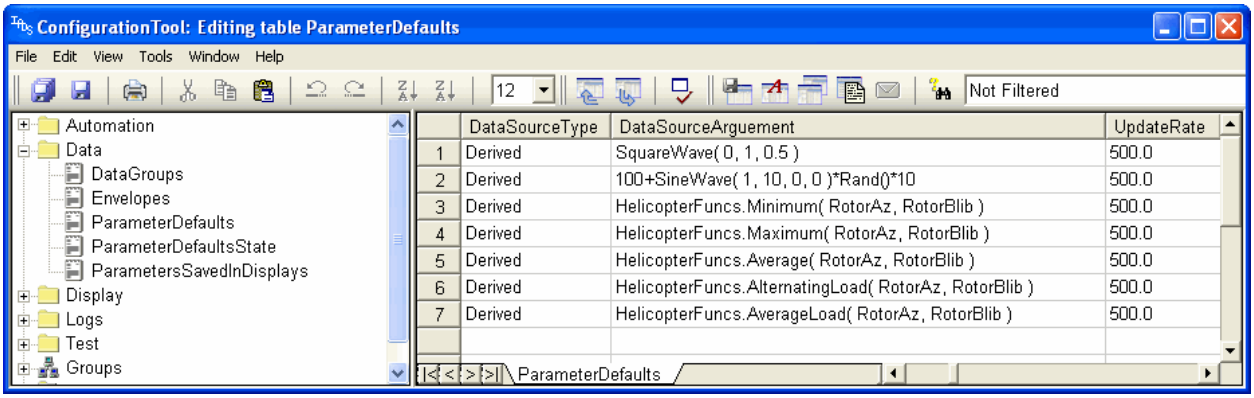


Figure 3-1 Derived Parameters in the IADS Configuration Tool

4. Helicopter Function Reference

4.1. Overview

The helicopter functions calculate the **Min**, **Max**, **Average**, **Alternating Load** (**Oscillatory**) and **Average Load** per rotor turn given an input parameter and a "rotor blib" parameter. The first argument within each function "InputParam" can be any measurement you would like to analyze within the cycle. The cycle (period of time) for each calculation is defined by the second parameter "RotorBlibParam". For a more in depth discussion of how the blib input parameter defines a calculation cycle, see the remarks section below.

Each function, its arguments, and algorithm are described below.

4.2. Min

Description: Calculates the minimum of all data points in InputParam during a cycle.

Syntax: **HelicopterFuncs.Minimum(InputParam, RotorBlibParam)**

Algorithm: Min(all data points during cycle)

4.3. Max

Description: Calculates the maximum of all data points in InputParam during a cycle.

Syntax: **HelicopterFuncs.Maximum(InputParam, RotorBlibParam)**

Algorithm: Max(all InputParam data points during cycle)

4.4. Average

Description: Calculates the average of all data points in InputParam during a cycle.

Syntax: **HelicopterFuncs.Average(InputParam, RotorBlibParam)**

Algorithm: Average(all data points during cycle)

4.5. AlternatingLoad (Oscillatory)

Description: Calculates the average of the difference between the maximum and minimum of all data points in InputParam during a cycle.

Syntax: **HelicopterFuncs.AlternatingLoad(InputParam, RotorBlibParam)**

Algorithm: Max(all points during cycle) - Min(all points during cycle)) / 2.0

4.6. AverageLoad

Description: Calculates the average of the sum between the maximum and minimum of all points in InputParam during a cycle.

Syntax: **HelicopterFuncs.AverageLoad(InputParam, RotorBlibParam)**

Algorithm: Max(all points during cycle) + Min(all points during cycle)) / 2.0

4.7. Remarks

The rotor blib is simply a parameter that toggles between 0 and 1 for each complete rotor turn. For instance, if you watched the rotor blib parameter in a strip chart, you would see it transition from 0 to 1 to 0 and continue this pattern indefinitely, where each transition point marks the next complete rotor turn. A number of groups have rotor blib output parameters that don't fit this profile. In that case, you will need to create a derived parameter which converts your rotor blib equation into a 0..1..0 transition type. Once the derived parameter is created, you can use it as your blib input to the equations. This is not a problem in most cases and has been done for a number of test programs. If you need some assistance on converting your blib parameter, please contact customer support or post the question to our Google group: http://groups.google.com/group/iads. Also, if you wish to compute the results at rates faster then the normal blib transition (i.e. more than once per rotation or say at each 15 degree increment), you'll simply use this same technique and create a derived blib parameter that calculates the higher transition frequencies based of the original blib signal.

5. Comments on Previous Versions

During previous versions of the Helicopter Function dll, there was growing discussion and confusion among a number of groups regarding the exact interpretation of the blib parameter. If you're un-aware of these issues, you can simply disregard this discussion. As for previous users that might be questioning the issues still, please read on.

The question was posed: Did the exact data point at the transition of the blib value mark the start of a new cycle (thus we should include it's value in the current cycle's calculation) or did it mark the end of the previous cycle (thus we should not include it's value in the current cycle)?

The consensus developed as we began testing different systems and starting examining the results. Also, everyone involved had more time to discuss the issues and view the input parameter data, blib transitions, and output results side by side. The consensus was that all the values of the input parameter that occurred on a given RotorBlib value were to be used in the same calculation. It seems simple when you say it in that manner, but at the time there was some confusion.

The bottom line is that if you have a blib signal like this:

0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0

And input parameter values like this:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Because the first six values of the blib signal are all zeros -> the first six values of the input parameter will be used to calculate the results of that particular "cycle" of the rotor. Likewise, because the next six values of the blib signal are all ones -> the next six values of the input parameters will define the second cycle (and so on). In other words, the blib parameter doesn't actually define the

“transition” between cycles as much as it defines the inclusion identification. Values that line up in time (or ordinal number if the sample rate of the blib parameter is equal to the sample rate of the azimuth parameter) and numeric content belong together in the same calculation. 1..6 belong in the same calculation cycle, 7..12 in the next cycle, and 13..18 the last cycle.

In the previous version of the functions, there was a third argument attached to each function called “BlibTransitionDefinesStartOfCycleVsEnd”. At the time it was recommended that you set the value to TRUE. In this latest version of the dll, the argument was removed. To keep backward compatibility, its value is still allowed to be used, but it will default to TRUE if not supplied. Setting this value to FALSE is not recommended as it will cause the shift of the values in cycle calculation by one data point. Both settings produce nearly the same output, but they are not identical.