

RAFTIMATE Installation Guide

System Requirements

RAFTIMATE requires a Windows computer with Microsoft Excel and MathWorks MATLAB installed. RAFTIMATE has primarily been tested on Windows XP Professional SP2 and MATLAB r2006a. Other versions should work, but have not been tested.

Copying and Unzipping Files

Using WinZip or compatible program, first unzip the program function library (“RAFTIMATE Version 1.5.zip”) to a location that can be read, but not necessarily edited by the users who need to run the program. Windows file servers are a good option.

Next, unzip the Excel HUL file (“F6_MasterComponentsList_r14.zip”) to a location that can be accessed and edited by the user during run time. This file contains the satellite module component information.

Finally, unzip the example input-output directory file (“Demo.zip”) to the location where the program will be run. This location is where input decks can be edited and program output deposited.

RAFTIMATE Execution Guide

RAFTIMATE comes with two modes of execution, the high-fidelity Monte Carlo mode and the lower fidelity Rapid Screening mode. Monte Carlo mode combines time-dependent launcher reliability with individual component modeling, multiple development delay models and survivability modeling to generate a large number of cluster time histories. These time histories are then analyzed to generate present benefit – present cost scatter plots and benefit monetized to generate net present value histograms.

The Rapid Screening version takes the same input deck, but makes several simplifying assumptions to speed up the execution time to assess a cluster. First, all the probability distributions in the model are assessed at a specified confidence level. This eliminates the multiple trials required for a full probabilistic assessment. Second, the perturbations are reduced to only launch vehicle reliability, module reliability and the payload development TRL effect. Launch vehicle reliability is assumed to be constant and the distribution of number of launches is modeled as a Poisson process. Module reliability is calculated by rolling up the component reliabilities for each individual module. Unfortunately, this time saving measure eliminates the ability of the program to model component sharing. The payload TRL effect is analyzed using probability theory to directly calculate the module delay distribution from its constituent payload distributions. These changes allow a full cluster analysis to take place in 1/20th of a second on an Intel CoreDuo Windows XP machine, down from roughly 40 minutes for the Monte Carlo.

Monte Carlo RAFTIMATE

RAFTIMATE HUL's

RAFTIMATE uses hardware lists created in Microsoft Excel with a very specific format. This formatting includes each component type present in a particular module along with information in columns about:

- Redundancy – quantity present and quantity required for module operation or sharing operation
- Mass
- Power
- Shareability – ability to both share and backup this component with an identical component on another module
- Cost model category – CER type to use for costing
- Local sharing criticality – list of WBS items on this module that rely on this component for sharing
- Global sharing criticality – list of WBS items in the cluster that rely on this component for sharing

An example WBS is shown in Figure 1. While it is possible to generate these HUL's manually, it is a labor intensive task.

Category	WBS	Component	Number Installed	Minimum per module	Power (W)	1 year reliability	TRL	Can Publish	Can Subscribe	WBS number sharing affected by failure	WBS number sharing affected by failure globally	Comp Mass (kg)	Total Mass (kg)	Cost Classification
Structure	1.7.2	Frame Assy	1	1			1	N				2.160482627	2.160482627	struc_thermal
	1.7.3	Corner Posts	4	4			1	N				0.489667227	1.958669007	struc_thermal
	1.7.4	Upper Horizontal Panel	1	1			1	N				4.806314071	4.806314071	struc_thermal
	1.7.5	Lower Horizontal Panel	1	1			1	N				1.911004264	1.911004264	struc_thermal
	1.7.6	+Y panel Radiators	1	1			1	N				1.237412683	1.237412683	struc_thermal
	1.7.7	-Y panel Radiators	1	1			1	N				1.237412683	1.237412683	struc_thermal
	1.7.8	+X panel	1	1			1	N				1.237412683	1.237412683	struc_thermal
	1.7.9	-X panel	1	1			1	N				1.237412683	1.237412683	struc_thermal
	1.7.10	LV Adapter-Bus Side	1	0			1	N				4.220081103	4.220081103	struc_thermal
	1.7.11	Sep Bolts	2	0			1	N				0.986212737	1.970425474	struc_thermal
	1.7.12	Misc	1	1			1	N				3.176967271	3.176967271	struc_thermal
Propulsion	2.7.2	Propellant Tank	6	6		0.99912378		N				1.709603893	10.25900399	prop_liquid
	2.7.3	Pressurant Tank	2	1			1	N				3.329175361	6.658350721	prop_liquid
	2.7.4	2 8ft Thruster	8	6		0.990037218		N				0.347682119	2.781456954	prop_liquid
	2.7.5	8 8ft Thruster	1	1		0.990037218		N				0.714415313	0.714415313	prop_liquid
	2.7.6	Thruster Brackets	8	8			1	N				0.141522272	1.132177817	struc_thermal
	2.7.7	Fluid Pressure couplings	4	4			1	N				0.323662076	1.294664302	prop_liquid
	2.7.8	Pressure Regulator	1	1			1	N				0.88111222	0.88111222	prop_liquid
	2.7.9	Latch Valve-Prop	5	5		0.99924895		N				0.367207657	1.796038284	prop_liquid

Figure 1 - Example Excel Input Module HUL

HUL Generator

The HUL generator uses a master component list present on the tab “Master Component List”. This database has several satellite bus choices along with the components present on those buses. In addition, at the bottom of the page, component information for various payloads has been saved. This information is used by a HUL generator macro that can be accessed on the “Input Deck” tab of the example HUL Excel file. This input deck is shown in Figure 2.

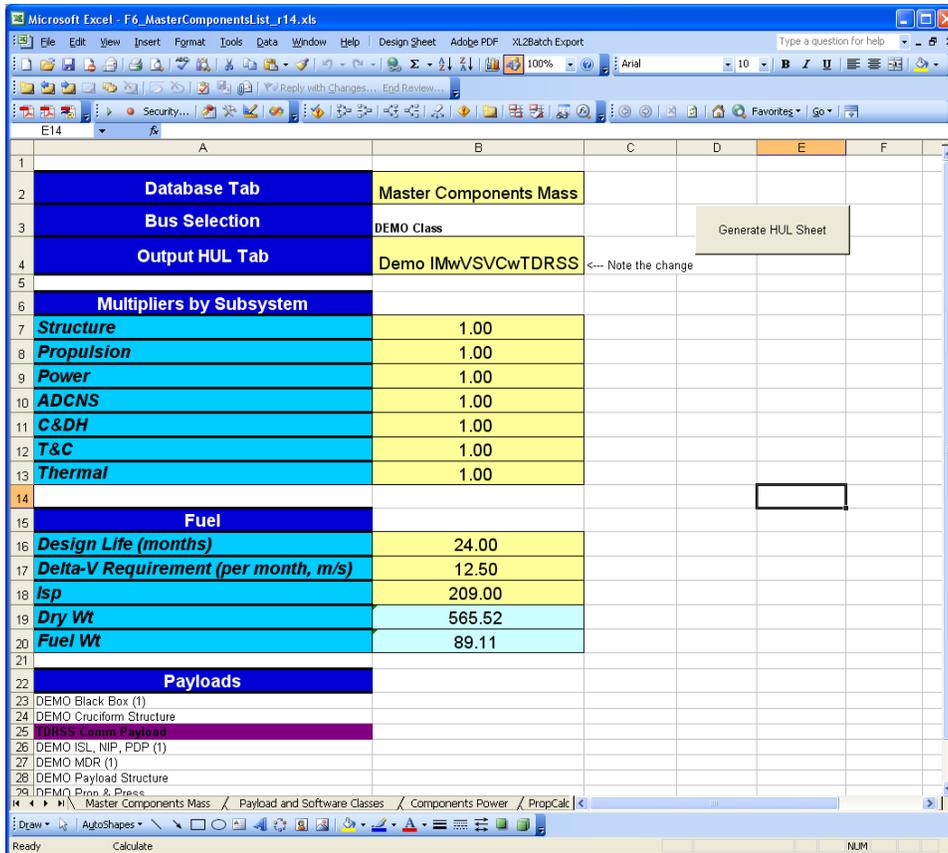


Figure 2 - HUL Generator Input Deck

To operate the generator, first fill in the inputs in cells B2-B4 indicating the tab containing the component database, the HUL set to use and the desired name of the output HUL tab. Inputs B6-B13 set weight multipliers for the components in each subsystem. This is a crude way to scale up the bus hardware and should be used sparingly. In cells B16-B20 are the propellant inputs and summary outputs. The HUL requires a delta-V consumption rate and a design propellant life to calculate the propellant and propellant tank sizes. Finally, cells A23 and below is the list of payloads from the component database that are to be included in the HUL.

Once all the HUL's for a particular cluster have been created, the input decks for RAFTIMATE need to be created based on these HUL's.

RAFTIMATE Input Deck

The RAFTIMATE input deck is written in the form of a MATLAB function that calls functions that reside in the tool.

Payload Menu

The payload menu consists of a list of declarations of payload objects. Payload objects can be any name, as long as the obj.type corresponds to one of the following types:

- Transmitter

- ReallySimpleTransmitter
- SimpleEO
- ReallySimpleEO
- GenericPL1 ... GenericPL8
- Processor

Each object type has a distinct set of input requirements. These requirements can be inferred from the function “satellite_menu.m”. An example set of inputs is shown in Figure 3.

```

1 function [cluster, launch_menu] = generate_simple_cluster(moduledata, launchtable)
2
3 %Special Trade Study Variables Version
4
5 %build PL menu
6
7 %Really simple transmitter inputs
8 rsto.type = 'ReallySimpleTransmitter';
9 rsto.goal = 2;
10
11 %Really simple EO inputs
12 rseo.type = 'ReallySimpleEO';
13 rseo.goal = 2;
14
15 gp.type = 'GenericPL1';
16
17 %Processing inputs
18 pr.type= 'processor';
19 pr.ncards = 2;
20 pr.flops = pr.ncards * 8e8;
21 pr.Wt = 2 * 5;
22 pr.Pwr = 2 * 20;
23

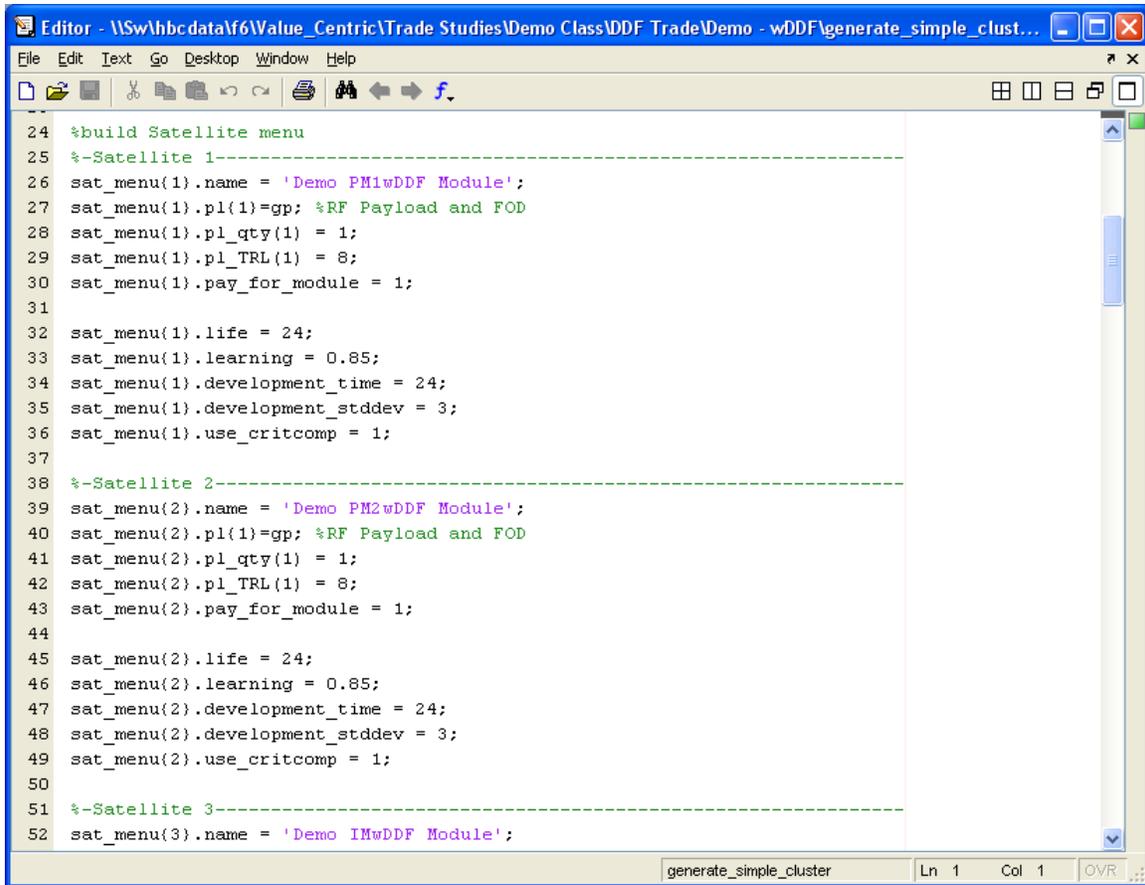
```

Figure 3 - Example Payload Inputs

Module Menu

The module menu populates a cell structure in the model called “sat_menu”. Each cell is a module that can be included in the cluster. Each module needs a name that corresponds to a HUL tab in the Excel HUL input file, a list of payload assignment objects from the payload menu, the quantity of each type of payload, the TRL of each payload and a flag the determines whether the module costs need to be accounted for in the analysis.

The second input block contains the nominal propellant life, the module production learning factor, the module baseline development time, the development time standard deviation and a flag that determines whether or not the Excel HUL file is used for input. The Excel HUL file is required for Monte Carlo analysis. An example set of modules is shown in Figure 4.



```
24 %build Satellite menu
25 %--Satellite 1-----
26 sat_menu{1}.name = 'Demo PM1wDDF Module';
27 sat_menu{1}.pl{1}=gp; %RF Payload and FOD
28 sat_menu{1}.pl_qty(1) = 1;
29 sat_menu{1}.pl_TRL(1) = 8;
30 sat_menu{1}.pay_for_module = 1;
31
32 sat_menu{1}.life = 24;
33 sat_menu{1}.learning = 0.85;
34 sat_menu{1}.development_time = 24;
35 sat_menu{1}.development_stddev = 3;
36 sat_menu{1}.use_critcomp = 1;
37
38 %--Satellite 2-----
39 sat_menu{2}.name = 'Demo PM2wDDF Module';
40 sat_menu{2}.pl{1}=gp; %RF Payload and FOD
41 sat_menu{2}.pl_qty(1) = 1;
42 sat_menu{2}.pl_TRL(1) = 8;
43 sat_menu{2}.pay_for_module = 1;
44
45 sat_menu{2}.life = 24;
46 sat_menu{2}.learning = 0.85;
47 sat_menu{2}.development_time = 24;
48 sat_menu{2}.development_stddev = 3;
49 sat_menu{2}.use_critcomp = 1;
50
51 %--Satellite 3-----
52 sat_menu{3}.name = 'Demo IMwDDF Module';
```

Figure 4 - Example Module Inputs

Launch Menu

The launch menu consists of “packages” cells that are attached to the object launch menu. Packages only need to be defined if more launchers than one rank one launcher per defined module is required, as these launchers are automatically created later in the code. Each launch package definition has a:

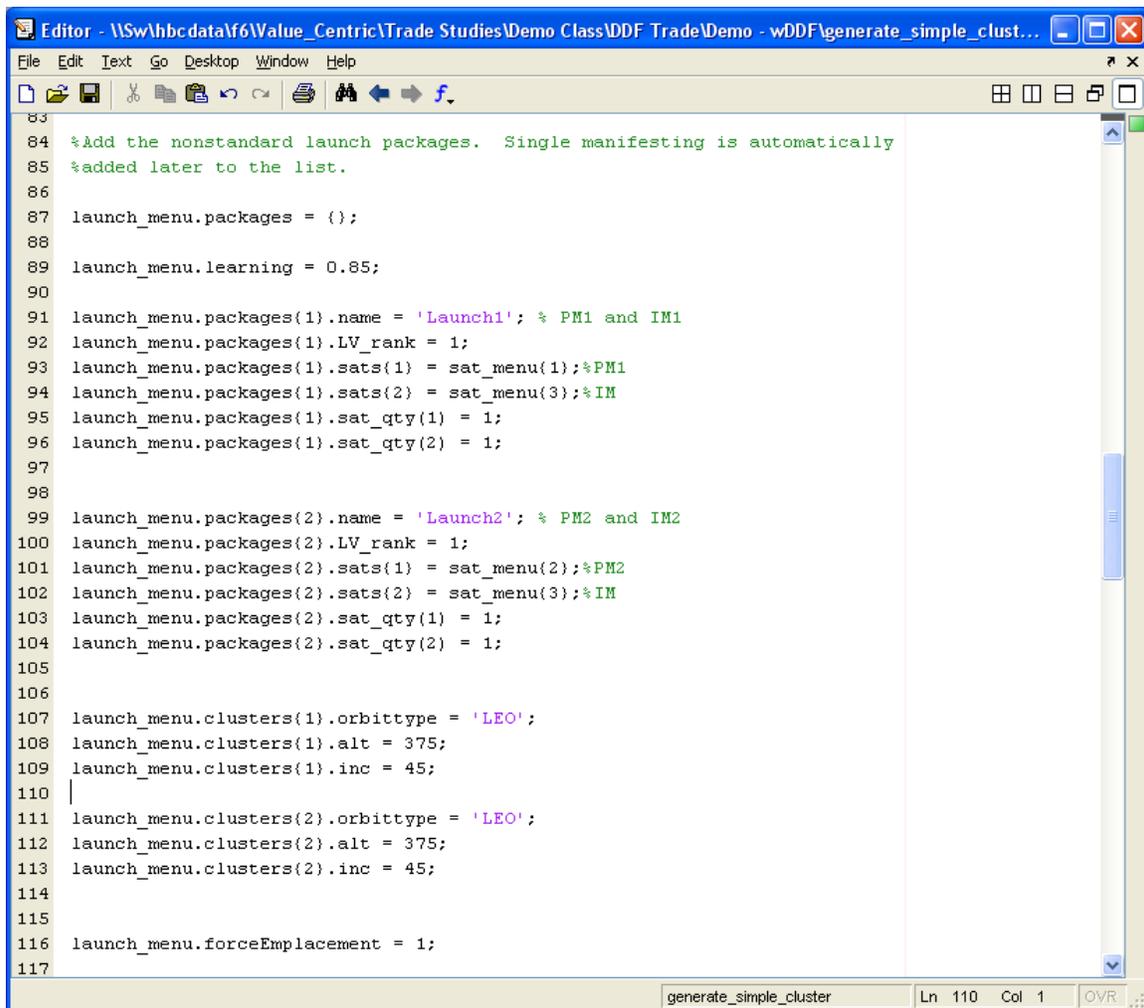
- Name
- LV_rank – this is the cost rank of the launch to select. For example, a rank 2 package will select the second cheapest launcher in the table that can perform the requested mission.
- Sats cells – these cells contain satellite objects from the satellite menu for the satellites in the launch package.
- Sat_qty cells – number of each satellite type

There are a few top level inputs in the launch menu as well. The “learning” field contains the launcher production learning factor. The “forceEmplacement” field tells the model whether to precisely follow the prescribed launch packages (=1) or to launch the modules in the fastest order possible (=0).

Once the launch package requests have been defined, the location(s) for the launch menu need(s) to be defined. RAFTIMATE accepts multiple orbit requests using the most stressing orbit location as the basis for launcher selection. To enter a location, create a cell vector called “clusters” attached to the “launch_menu” object. Each cell in the vector is a separate launch location. Each location needs to have the following fields:

- Orbittype – This is set to either ‘LEO’ or ‘GEO’ and changes the location of the launch table lookup. ‘GEO’ is assumed to be direct injection into geostationary orbit. ‘LEO’ launch capacities are only valid to about 1000km altitude and up to a polar inclination.
- Alt – This is the altitude in km of the orbit location
- Inc – This is the inclination in degrees of the orbit location

An example launch menu is shown in Figure 5.

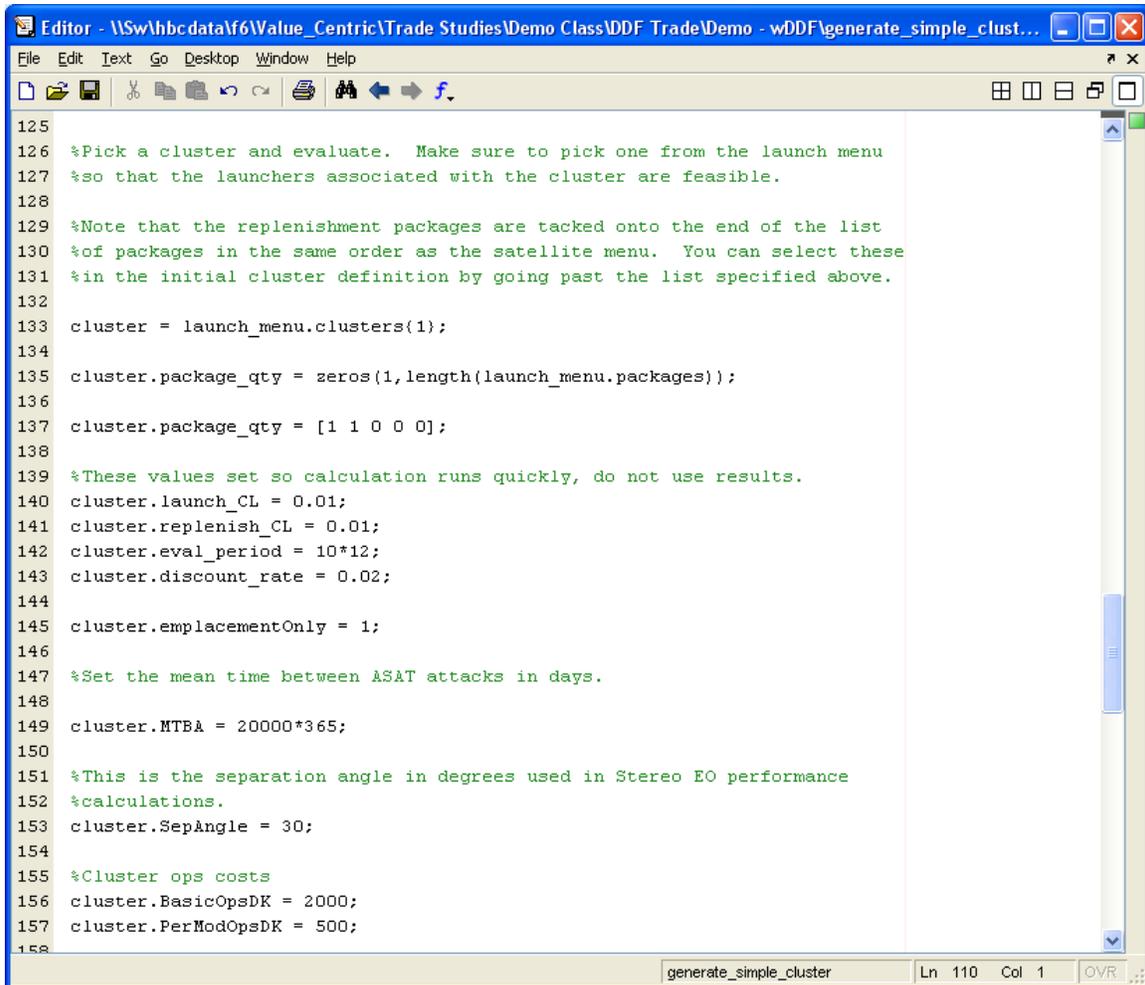


```
83
84 %Add the nonstandard launch packages. Single manifesting is automatically
85 %added later to the list.
86
87 launch_menu.packages = {};
88
89 launch_menu.learning = 0.85;
90
91 launch_menu.packages(1).name = 'Launch1'; % PM1 and IM1
92 launch_menu.packages(1).LV_rank = 1;
93 launch_menu.packages(1).sats(1) = sat_menu(1);%PM1
94 launch_menu.packages(1).sats(2) = sat_menu(3);%IM
95 launch_menu.packages(1).sat_qty(1) = 1;
96 launch_menu.packages(1).sat_qty(2) = 1;
97
98
99 launch_menu.packages(2).name = 'Launch2'; % PM2 and IM2
100 launch_menu.packages(2).LV_rank = 1;
101 launch_menu.packages(2).sats(1) = sat_menu(2);%PM2
102 launch_menu.packages(2).sats(2) = sat_menu(3);%IM
103 launch_menu.packages(2).sat_qty(1) = 1;
104 launch_menu.packages(2).sat_qty(2) = 1;
105
106
107 launch_menu.clusters(1).orbittype = 'LEO';
108 launch_menu.clusters(1).alt = 375;
109 launch_menu.clusters(1).inc = 45;
110 |
111 launch_menu.clusters(2).orbittype = 'LEO';
112 launch_menu.clusters(2).alt = 375;
113 launch_menu.clusters(2).inc = 45;
114
115
116 launch_menu.forceEmplacement = 1;
117
```

Figure 5 - Example Launch Menu Inputs

Cluster Definition

The definition of a cluster begins with the selection of an orbit location from the launch menu. This is done as shown in Figure 6. Next, the elements of the cluster need to be selected. This is done by requesting quantities of launch packages in the “package_qty” field of cluster. The beginning elements of the vector correspond in order of the launch packages defined by the user for the launch menu. The second part of the vector corresponds to the modules as launched individually. Because this is always needed for replenishment, single manifested packages are created automatically regardless of user input. An example is shown in Figure 6.



```
125
126 %Pick a cluster and evaluate. Make sure to pick one from the launch menu
127 %so that the launchers associated with the cluster are feasible.
128
129 %Note that the replenishment packages are tacked onto the end of the list
130 %of packages in the same order as the satellite menu. You can select these
131 %in the initial cluster definition by going past the list specified above.
132
133 cluster = launch_menu.clusters(1);
134
135 cluster.package_qty = zeros(1,length(launch_menu.packages));
136
137 cluster.package_qty = [1 1 0 0 0];
138
139 %These values set so calculation runs quickly, do not use results.
140 cluster.launch_CL = 0.01;
141 cluster.replenish_CL = 0.01;
142 cluster.eval_period = 10*12;
143 cluster.discount_rate = 0.02;
144
145 cluster.emplacementOnly = 1;
146
147 %Set the mean time between ASAT attacks in days.
148
149 cluster.MTBA = 20000*365;
150
151 %This is the separation angle in degrees used in Stereo EO performance
152 %calculations.
153 cluster.SepAngle = 30;
154
155 %Cluster ops costs
156 cluster.BasicOpsDK = 2000;
157 cluster.PerModOpsDK = 500;
158
```

Figure 6 - Cluster Definition Inputs

The next block of inputs corresponds to econometric assumptions:

- “launch_CL” and “replenishment_CL” refer to confidence level analyses on the launch and module reliabilities respectively. For complex clusters, these analyses can take a long time to calculate, so it is recommended that unless this output is going to be used, the confidence levels be set to 1%. This will not affect the Monte Carlo simulation in any way.

- Next is the simulation analysis period “eval_period”. This is the window of time over which the simulation should assess the cluster in months.
- “discount_rate” is the discount rate to be used to calculate present benefit and present costs.
- “emplacementOnly” is a flag that tells RAFTIMATE whether or not to reconstitute the cluster in the event of module failures.
- “MTBC” is the mean time between collisions in days and defines the frequency of the random survivability events in RAFTIMATE.
- “SepAngle” is the separation angle in degrees of any sparse aperture modules and is only used in “StereoEO” payload performance calculations.
- “BasicOpsDK” is the Ops cost of operating the first module in the cluster in thousands of 2008 dollars.
- “PerModOpsDK” is the Ops cost of operating subsequent modules that are added to the cluster.
- “clustertype” is used to define which tab in the Excel HUL to look at to get the list of cluster critical shareable components. This list defines which components can be shared across the cluster and only need to exist a specified number of times. A component that is needed a specified number of times per module should not be included on the cluster critical list.

Cluster Perturbation Switches

Perturbations can be turned on, off or overridden using the list of switches in this section. This is useful for identifying perturbation drivers of system response. The “onoff” switches activate and deactivate the perturbations while the “direct” switches tell the code to call a user-defined helper function in place of the existing perturbation. The names of the user-defined helper functions are as follows:

- Dev - direct_satellite_availability.m
- DevDelay - direct_module_dev_disruption.m
- ReqChange - direct_cluster_req_change_disruption.m
- collision - direct_collision.m
- Components - direct_compfail.m
- Launch_failure - direct_launchsuccess.m

It is recommended that these user defined functions reside in the run input-output directory rather than the main code library. The ability to override perturbations was included primarily for vignette analysis and is an advanced RAFTIMATE feature. Use this feature only if you are very familiar with the code.

Cluster Scoring Inputs

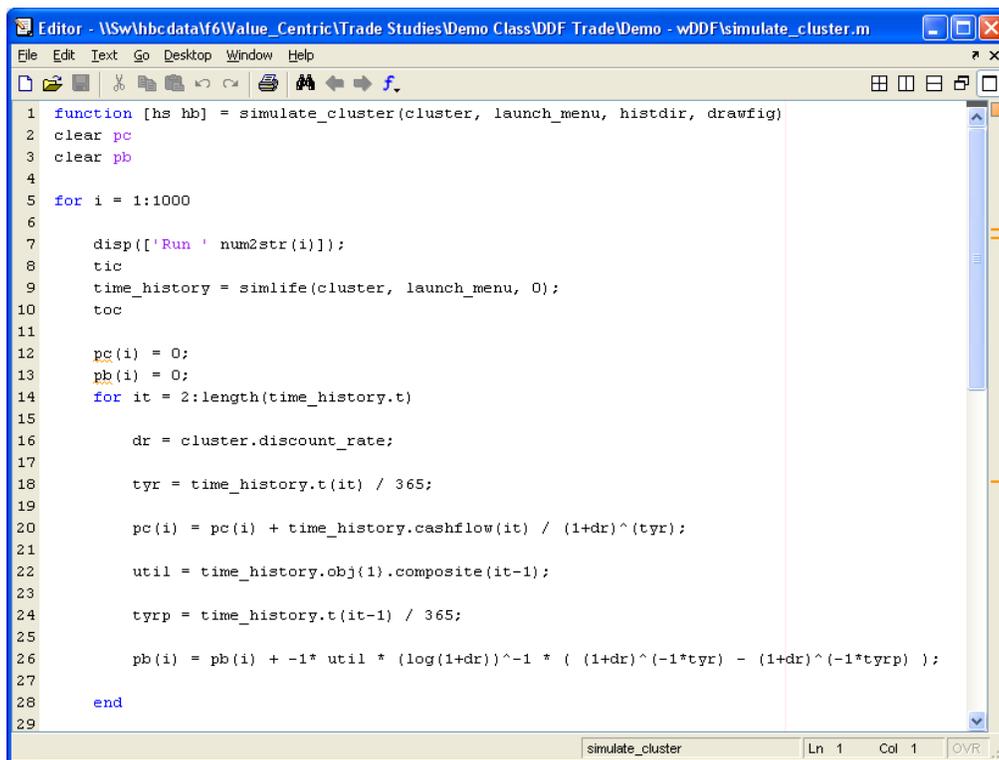
To obtain the cluster benefit score, the final set of inputs are needed. Multiple scores can be calculate using different “obj” cells. Each “obj” cell corresponds to a different scoring criteria. The fields for an obj cell are:

- Weighting – this is a vector of the weightings given to each of the scoring criteria.

- Thresholds – these are the values that are to be used as a floor for scores. Criteria below the threshold will count as zero.
- Goals - the criteria goals are scored in most schemes at a value of 1. Whether or not this is the ceiling for the scoring criteria is determined by the “single_limit” variable.
- single_limit – This is the maximum score that can be attained by any one criteria. A value of 1 corresponds to the criterion goal.
- Type – this determines the type of scoring to be used. Valid fields are ‘minterm’, ‘product’ and ‘sum’. ‘minterm’ uses the minimum score from the two fields to determine the score of the cluster. ‘product’ multiplies the scores together while ‘sum’ adds the scores together.
- Val_loc – This is a cell vector containing string names for the fields in the cluster object to be used as the criteria. Any numerical field in the cluster object can be used as a criteria.

RAFTIMATE Monte Carlo Control Script

The file for controlling the Monte Carlo simulation is “simulate_cluster.m” that should reside in the input-output directory. This function is typically called from the “runtrade.m” script, also in the input-output directory.



```

1 function [hs hb] = simulate_cluster(cluster, launch_menu, histdir, drawfig)
2 clear pc
3 clear pb
4
5 for i = 1:1000
6
7     disp(['Run ' num2str(i)]);
8     tic
9     time_history = simlife(cluster, launch_menu, 0);
10    toc
11
12    pc(i) = 0;
13    pb(i) = 0;
14    for it = 2:length(time_history.t)
15
16        dr = cluster.discount_rate;
17
18        tyr = time_history.t(it) / 365;
19
20        pc(i) = pc(i) + time_history.cashflow(it) / (1+dr)^(tyr);
21
22        util = time_history.obj(1).composite(it-1);
23
24        tyrp = time_history.t(it-1) / 365;
25
26        pb(i) = pb(i) + -1* util * (log(1+dr))^-1 * ( (1+dr)^(-1*tyr) - (1+dr)^(-1*tyrp) );
27
28    end
29

```

Figure 7 - Example Monte Carlo Control Script

The number of Monte Carlo trials is controlled by the “for” loop in simulate_cluster.m as seen in Figure 7. A random time history is generated by calling simlife.m and sending it a cluster object, a launch menu object and a flag to determine the level of text output

desired. If the flag is set to 0, then only the run number and run time will be displayed at each trial. If the flag is set to 1, then a description of each event that occurs during the trial will be scrolled across the screen.

Once the timeline has been generated, the next code block discounts the cost and benefits. Cost is discounted as discrete cash flows corresponding to each event. Benefit is integrated continuously and is assumed to be constant between events.

RAFTIMATE Trade Study File

The master trade study script to execute RAFTIMATE is typically like the example “runtrade.m” provided in the example inputs. This script can be modified to change variables in “generate_simple_cluster.m” to facilitate automated trade analysis.

Path Inputs

RAFTIMATE requires the path to both the Excel HUL file and function library in order to execute. The function library needs to be added to the MATLAB path while the excel path is stored in the “moduledata” variable that is sent to “generate_simple_cluster.m”. Also, the name of the launcher table to be used is stored in the variable “launchtable”. This table should be present in the input-output directory as shown in the example files. If there is no table present in the input-output directory, then the default value of “launcher_table” will use the copy present in the function library.

Calling draw_football.m script

Once the trade script has run the Monte Carlo, the “draw_football.m” script can be run to copy the present benefit and present cost data into Excel as well as fit a multivariate normal distribution to the 2-D data. Once the distribution has been fit, the script outputs 1, 2 and 3-sigma outlines for ease of display.

RAFTIMATE Monetization

Monetizing the Reference Case

To monetize a reference case using the Boeing comparison algorithm developed for F6, simply execute the script “monetizer.m” after the Monte Carlo and “draw_football.m” has been executed. This will edit the “football.xls” and add the exchange rate and the net present value for each Monte Carlo trial, along with a histogram plot of net present value.

Monetizing Comparison Cases

To monetize a case using the exchange rate from another run, execute the script “monetizer_compare.m”. This script will bring up a user prompt for the location of a set of reference data that contains exchange rate information generated by “monetizer.m”. Once this case has been located, the script will update the “football.xls” file with the exchange rate and net present value information.