This document describes the GRUE-42 language specification in its entirety and why certain pieces of functionality are designed the way they are.
1 GRUE-42 Grammar

The grammar of GRUE-42 is based loosely on other grammars, but in the end is newly derived for the project. It does however tend to follow C scoping rules, and the form of declarations is fairly consistent with C style declarations.

The purpose of GRUE-42 was to allow for the creation of orbital equations using only simple orbital element variables as input. Using these simple inputs a more complex Matlab document is output to a file, which then can be run by the user providing useful data for analysis of orbits.

1.1 Declarations / Statements

The grammar allows for the declaration of different variables unique to orbital elements to define the orbit itself. The idea was to keep GRUE-42 as simple as possible for the user, so the number of different declarations that can be made is somewhat limited. However the same definitions can be used again with different arguments, allowing for a large amount of output from a small declaration set.

*Declarations:*

**Program:**
- is a list of Object

**Object:**
- SatelliteDefinition / PlanetDefinition / Computations.

**SatelliteDefinition:**
- 'Satellite' Name SatelliteDescription 'End'.

**PlanetDefinition:**
- 'Planet' Name PlanetDescription 'End'.

**Computations:**
- 'Computations' CompList 'End'.

**SatelliteDescription:**
- is a list of SatelliteParam

**PlanetDescription:**
- is a list of PlanetParam

**CompList**
- is a list of Computation

**SatelliteParam:**
- OrbitType / SemiMajorAxisDeclaration / SemiMinorAxisDeclaration / LinearEccentricityDeclaration / EccentricityDeclaration / CenterDescription / MassDescription / EpochDescription / LifetimeDeclaration / InclinationDeclaration.

**PlanetParam:**
PlanetType / SemiMajorAxisDeclaration / SemiMinorAxisDeclaration / LinearEccentricityDeclaration / EccentricityDeclaration / CenterDescription / MassDescription / TiltDeclaration.

Computation:
VelocityApoapsisRequest / VelocityPeriapsisRequest / PeriodRequest / ShadowRequest / PositionRequest / DistanceRequest.

OrbitType:
OrbitLEO / OrbitGEO.

SemiMajorAxisDeclaration:
'a' := Number.

SemiMinorAxisDeclaration:
'b' := Number.

LinearEccentricityDeclaration:
'e' := Number.

EccentricityDeclaration:
'e' := Number.

CenterDescription:
CenterPlanet / CenterCoordinate.

MassDescription:
'Mass' := Number.

EpochDescription:
'Epoch' := Number.

LifetimeDeclaration:
'Lifetime' := Number.

InclinationDeclaration:
'I' := Number.

TiltDeclaration:
'Tilt' := Number.

VelocityApoapsisRequest:
'VeLOCITY_A' (Name).

VelocityPeriapsisRequest:
'VeLOCITY_P' (Name).

PeriodRequest:
'P' (Name).

ShadowRequest:
'Shadow' (Name).

PositionRequest:
'Position' (Name).
DistanceRequest:
   'Distance' '(' DistanceObject DistanceObject ')' .

OrbitLEO:
   'Orbit' '=' 'LEO' .

OrbitGEO:
   'Orbit' '=' 'GEO' .

CenterPlanet:
   'Center' '=' PlanetType .

CenterCoordinate:
   'Center' '=' Coordinate .

DistanceObject:
   PlanetType / Name.

Coordinate:
   '(' Number ',' Number ',' Number ')' .

PlanetType:
   'Earth' / 'Moon' / 'Sun' .

Name:
   Identifier .

Number:
   Float .

2 Structural Analysis

The purpose of the structural analyzer is to build the abstract syntax tree (AST) from the user defined input. ELI generates a structural analyzer from the specifications of the
   • AST structure
   • Basic symbol and comments

2.1 The Abstract Syntax Tree

In the course of compiling a program a compiler will complete many different calculations on the AST including analyzing and translating of the source program. Making these computations as simple as possible is the main goal of the AST. The following example input code corresponds to Figure 1, an AST representation.

Example 1: Example of input code.

Satellite CX
   a = 25.7
   b = 23.0
   e = 1214.8

   Center = (55.0, 56.0, 66.0)
   Mass = 456.0
I = 23.5
Lifetime = 5.0
Epoch = 3298743298.0
End

Computations
Va (CX)
Vp (CX)
Position (CX)
End

The AST for GRUE-42 was designed to help keep the computations section as simple as possible when dealing with PTG. Since the major portion of the conversion to Matlab is dealt with by the computations section, one of the goals was to keep that section as simple as possible.

Figure 1 shows how the AST is broken down into different categories, where the three major ones are SatelliteDef, PlanetDef, and Computations. The tree in Figure 1 does not display the category PlanetDef due to the simplified input source. These categories help control the data as the compiler will eventually have to access the information at each terminal node. The categories SatelliteDef and PlanetDef organize the information based on the fact that the information the user provides is specific to an individual satellite or planet, unless the information is categorized as a computation.

Computations are defined as using the properties of a satellite or planet, stored within their respective definition. The computations then use name checking to get the correct information when the actual Matlab computations are being done, as a C style scope is used, where once a variable is declared it can be used by anything below it. Thus each computation will make sure to use the correctly named variables to avoid errors in the computations.

The reason for organizing in this pattern is that each declared computation, other than the distance computation, is specific to only one category at a time. Breaking up the information into these categories reduces the work in retrieving the information as NameUse is used in a computation, and is compared to the different NameDefs. The corresponding NameDef has all of the required information for the computation, either being a SatelliteDef or a PlanetDef.
Figure 1: AST diagram generated from user input file.
Example 2: LIDO specification.

RULE: Program LISTOF Object END;
RULE: Object ::= PlanetDef END;
RULE: Object ::= SatelliteDef END;
RULE: Object ::= Computations END;

RULE SatDefRule: SatelliteDef ::= 'Satellite' NameDef SatelliteDesc 'End' END;
RULE: SatelliteDesc LISTOF SatParam END;
RULE: SatParam ::= OrbitType END;
RULE: SatParam ::= SemiMajorAxisDecl END;
RULE: SatParam ::= SemiMinorAxisDecl END;
RULE: SatParam ::= LinEccentDecl END;
RULE: SatParam ::= EccentricityDecl END;
RULE: SatParam ::= CenterDesc END;
RULE: SatParam ::= MassDesc END;
RULE: SatParam ::= EpochDesc END;
RULE: SatParam ::= LifetimeDecl END;
RULE: SatParam ::= InclDecl END;
RULE: OrbitType ::= 'Orbit' '=' 'LEO' END;
RULE: OrbitType ::= 'Orbit' '=' 'GEO' END;

/* Planet Rules */
RULE PlanetDefRule: PlanetDef ::= 'Planet' NameDef PlanetDesc 'End' END;
RULE: PlanetDesc LISTOF PlanetParam END;
RULE: PlanetParam ::= PlanetType END;
RULE: PlanetParam ::= SemiMajorAxisDecl END;
RULE: PlanetParam ::= SemiMinorAxisDecl END;
RULE: PlanetParam ::= LinEccentDecl END;
RULE: PlanetParam ::= EccentricityDecl END;
RULE: PlanetParam ::= CenterDesc END;
RULE: PlanetParam ::= MassDesc END;
RULE: PlanetParam ::= TiltDecl END;
RULE: PlanetType ::= 'Earth' END;
RULE: PlanetType ::= 'Moon' END;
RULE: PlanetType ::= 'Sun' END;

/* Orbit Parameter Info*/
RULE: EpochDesc ::= 'Epoch' '=' Number END;
RULE: CenterDesc ::= 'Center' '=' PlanetType END;
RULE: CenterDesc ::= 'Center' '=' Coordinate END;
RULE: MassDesc ::= 'Mass' '=' Number END;
RULE: SemiMajorAxisDecl ::= 'a' '=' Number END;
RULE: SemiMinorAxisDecl ::= 'b' '=' Number END;
RULE: LinEccentDecl ::= 'c' '=' Number END;
RULE: EccentricityDecl ::= 'e' '=' Number END;
RULE: Coordinate ::= ('Number', 'Number', 'Number') END;
RULE: LifetimeDecl ::= 'Lifetime' '=' Number END;
RULE: InclDecl ::= 'I' '=' Number END;
RULE: TiltDecl ::= 'Tilt' '=' Number END;

/* Computation Rules */
RULE: Computations ::= 'Computations' CompList 'End' END;
2.2 Mapping Phrases to Nodes

Usually this is done by defining a complete concrete grammar, however in the case of GRUE-42 it simplified things by having the phrases contained within the AST. This means that each rule goes has the parsing already built into it which can be seen in above in Example 2.

2.3 Basic Symbols

GRUE-42 only has two types of symbols, of which both are specified in the gla file. The two symbols are a of type CString, and a custom definition for floating point numbers. The gla file can be seen below in Example 3.

Example 3: GRUE-42.gla file.

Identifier: C_IDENTIFIER [mkidn]

Num: $[-0-9][0-9]*\.?[0-9]+ [mkidn]

3 Semantic Analysis

3.1 Name Analysis

GRUE-42 supports the choice of several different computations that may be done on the supported Objects. This requires the user to specify the name of the object (be it a Satellite or a Planet) on which a specific computation is to be performed. Since computations can only be performed on already-defined Objects, the name of the Object must appear in a definition statement prior to the computation call.

In order to verify that the Object name was in fact already defined, the symbol NameUse is used in the computation statement (e.g. Va <NameUse>, to apply the Va computation). NameUse inherits IdUseEnv and ChkIdUse, thus using the build-in ELI roles to make sure the name has already been defined and does exist.
somewhere in the tree. This works also due the fact that when the name was originally defined (in the Object definition), the program used NameDef for the definition, which inherits IdDefScope to make sure the name is being defined.

3.2 Error Checking

For proper functionality of the requested computations, certain values must be assigned in order for a computation to have enough parameters to perform its calculations. The assignment of those values is checked by using a CHAIN that goes through the tree and resets each value to the StringTable whenever one of these variables is defined. After all assignments have been read in, the indexes of the required variables are checked. If a required variable’s index still maintains the zero (the equivalent of NULL) that it was assigned during the CHAIN creation, then an error message is printed, notifying the user the required information is missing.

GRUE-42 allows the user to reassign values to defined variables. Since the StringTable is updated using the CHAINs, every time a variable is redeclared, the new assigned value is added to the StringTable, and the index of the variable is changed to that new location in the Table. If the user is unsure of a value, they may enter -1 for any of any of the values of $a$, $b$, $c$, or $e$.

The required assignments change, depending on the Object. In Table 1 below are the variables that require assignments.
Table 1: Required assignments.

<table>
<thead>
<tr>
<th>Object</th>
<th>Variable Name</th>
<th>Variable Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satellite</td>
<td>( a )</td>
<td>Semi Major Axis</td>
</tr>
<tr>
<td></td>
<td>( b )</td>
<td>Semi Minor Axis</td>
</tr>
<tr>
<td></td>
<td>( c )</td>
<td>Linear Eccentricity</td>
</tr>
<tr>
<td></td>
<td>( e )</td>
<td>Eccentricity</td>
</tr>
<tr>
<td></td>
<td>Center</td>
<td>Center Point</td>
</tr>
<tr>
<td></td>
<td>Mass</td>
<td>Mass</td>
</tr>
<tr>
<td></td>
<td>Epoch</td>
<td>Epoch</td>
</tr>
<tr>
<td></td>
<td>Lifetime</td>
<td>Lifetime</td>
</tr>
<tr>
<td></td>
<td>( I )</td>
<td>Inclination</td>
</tr>
<tr>
<td>Planet</td>
<td>( a )</td>
<td>Semi Major Axis</td>
</tr>
<tr>
<td></td>
<td>( b )</td>
<td>Semi Minor Axis</td>
</tr>
<tr>
<td></td>
<td>( c )</td>
<td>Linear Eccentricity</td>
</tr>
<tr>
<td></td>
<td>( e )</td>
<td>Eccentricity</td>
</tr>
<tr>
<td></td>
<td>Center</td>
<td>Center Point</td>
</tr>
<tr>
<td></td>
<td>Mass</td>
<td>Mass</td>
</tr>
<tr>
<td></td>
<td>Tilt</td>
<td>Tilt</td>
</tr>
<tr>
<td>Computation</td>
<td></td>
<td>None Required</td>
</tr>
</tbody>
</table>

4 Sample Input and Output

4.1 Input

Example 4: Example input file, entitled input.
Satellite CX
\( a=-1 \)
\( b = 23.0 \)
\( c=-1 \)
\( e = 0.7 \)
Center = (55.0, 56.0, 66.0)
Mass = 456.0
\( I = 23.5 \)
Lifetime = 5.0
Epoch = 3298743298.0
Satellite ThreeCS
b = 500.0
a = 33.3
c = 10
e = -1

Center = Earth
Mass = 0.0
I = 12.0
Lifetime = 0.0
Epoch = 3398743010.0

End

Planet Mars
a = 0.0
b = -1
e = 0.0
c = -1
Center = (0.0, 0.0, 0.0)

Tilt = 20.0
Mass = 83289734987.0

End

Computations
Va (CX)
Vp (CX)
P (CX)
Shadow (CX)
Position (CX)
Distance (CX, Mars)

End

Example 5: Example input file, entitled input2
Satellite CX
a = 25.7
b = 23.0
c = -1
c = -1

Center = (55.0, 56.0, 66.0)
Mass = 456.0
I = 23.5
Lifetime = 5.0
Epoch = 3298743298.0

End

Computations
Va (CX)
Vp (CX)
4.2 Output

Example 6: GRUE-42 Matlab output from input.
close all
clear all
cle

c
%Define the Gravitational Constant
G=6.67259E-11;

%Define the Planets, then the Satellites and their parameters
CX_a = -1;
CX_b = 23.0;
CX_c = -1;
CX_e = 0.7;
CX_Center = [55.0, 56.0, 66.0];
CX_Center_Mass = 456.0;
CX_Inclination = 23.5;
CX_Lifetime = 5.0;
CX_Epoch = 3298743298.0;

CX_name = 'CX';
%Define mu (G*M)
CX_mu = G*CX_Center_Mass;

%Determine which of the two variables was declared, and calculate the remaining two
[CX_a,CX_b,CX_c,CX_e]=IfStatement(CX_a,CX_b,CX_c,CX_e);

ThreeCS_b = 500.0;
ThreeCS_a = 33.3;
ThreeCS_c = 10;
ThreeCS_e = -1;
ThreeCS_Center = [0.0, 10.0, 0.2];
ThreeCS_Center_Mass = 0.0;
ThreeCS_Inclination = 12.0;
ThreeCS_Lifetime = 0.0;
ThreeCS_Epoch = 3398743010.0;

ThreeCS_name = 'ThreeCS';
%Define mu (G*M)
ThreeCS_mu = G*ThreeCS_Center_Mass;

%Determine which of the two variables was declared, and calculate the remaining two
[ThreeCS_a,ThreeCS_b,ThreeCS_c,ThreeCS_e]=IfStatement(ThreeCS_a,ThreeCS_b,ThreeCS_c,ThreeCS_e);

Mars_a = 0.0;
Mars_b = -1;
Mars_e = 0.0;
Mars_c = -1;
Mars_Center = [0.0, 0.0, 0.0];
Mars_Tilt = 20.0;
Mars_Center_Mass = 83289734987.0;

Mars_name = 'Mars';
%Epoch for a planet is 0
Mars_Epoch=0;
%Inclination for a planet is 0
Mars_Inclination=0;
%Define mu (G*M)
Mars_mu = G*Mars_Center_Mass;

%Determine which of the two variables was declared, and calculate the remaining two
[Mars_a,Mars_b,Mars_c,Mars_e]=IfStatement(Mars_a,Mars_b,Mars_c,Mars_e);

fprintf('Results:
')
fprintf('---------------------------------------

%Now that we know everything, start doing the calculations
%Finding the Velocity at Apoapsis
CX_ra=CX_a+CX_c;
CX_Va=1000*sqrt(CX_mu/1E9)*((2000/CX_ra)-(1000/CX_a))^(1/2);
fprintf('****The Velocity of %s at Apoapsis is %.3d m/s\n',...
CX_name,CX_Va)

%Finding the Velocity at Periapsis
CX_rp=CX_a-CX_c;
CX_Vp=1000*sqrt(CX_mu/1E9)*((2000/CX_rp)-(1000/CX_a))^(1/2);
fprintf('****The Velocity of %s at Periapsis is %.3d m/s\n',...
CX_name,CX_Vp)

%Finding The Orbital Period
CX_P=2*pi*sqrt(CX_a^3/CX_mu);
fprintf('****The Period of %s is %.3d seconds\n',...
CX_name,CX_P)

%Finding the Shadow Time, must also compute the period
CX_radius = 6.378E6;
CX_A_shad = CX_radius*CX_a;
CX_A = pi*CX_a*CX_b;
CX_P=pi*sqrt(CX_a^3/CX_mu);
CX_Shadow = (CX_A_shad/CX_A)*CX_P;
fprintf('****The Shadow time for %s is %.3d seconds\n',...
CX_name,CX_Shadow)

%Finding the position, period must be calculated first
CX_Time =CX_Epoch+100000;
CX_P=pi*sqrt(CX_a^3/CX_mu);
[ x y z ] = CalcPosition (CX_Time, CX_Inclination, CX_Epoch, CX_P, CX_a, CX_e);
CX_Position = CX_Center + [ x y z ];
%Calculating the distance between CX and Mars
%Finding the position of CX
CX_Time = CX_Epoch + 100000;
CX_P = 2*pi*sqrt(CX_a^3/CX_mu);
[x y z] = CalcPosition(CX_Time, CX_Inclination, CX_Epoch, CX_P, CX_a, CX_e);
CX_Position = CX_Center + [x y z];

%Finding the position of Mars
Mars_Time = Mars_Epoch + 100000;
Mars_P = 2*pi*sqrt(CX_a^3/CX_mu);
[x y z] = CalcPosition(Mars_Time, Mars_Inclination, Mars_Epoch, Mars_P, Mars_a, Mars_e);
Mars_Position = Mars_Center + [x y z];

%Finding the Distance between both objects
CX_Mars_Distance = sqrt((CX_Position(1)+Mars_Position(1))^2 + (CX_Position(2)+Mars_Position(2))^2 + (CX_Position(3)+Mars_Position(3))^2);
fprintf('****The Distance from %s to %s is %.3d meters
', CX_name, Mars_name, CX_Mars_Distance);

function [x y z] = CalcPosition(Time, Inclination, Epoch, P, a, e)
%Find the x, y, and z values
angle = Inclination;
percent = mod(Time-Epoch, P)/P;
nu = 2*pi*percent; %true anomaly
r = a*(1-e^2)/(1+e*cos(nu));
x = r*cos(nu)*cos(angle);
y = r*sin(nu);
z = r*cos(nu)*sin(angle);
end

function [a b c e] = IfStatement(a, b, c, e)
%Determine which of the two variables was declared, and calculate the remaining two
if a == -1
  if b == -1
    c = sqrt(a^2-b^2);
    e = c/a;
  elseif c == -1
    e = c/a;
    b = a*sqrt(1-e^2);
  else
    b = a*sqrt(1-e^2);
    c = e*a;
  end
endif
elseif b == -1
  if c == -1
    a = sqrt(b^2+c^2);
  else
    c = a*sqrt(1-b^2);
    b = a; e = 0;
  endif
endif
\begin{verbatim}
e=c/a;
else
    a=b/sqrt(1-e^2);
c=e*a;
end
else
    a=c/e;
b=a*sqrt(1-e^2);
end

Example 7: Matlab Output from input.
Results:
---------------------------------------
****The Velocity of CX at Apoapsis is 1.291e-005 m/s
****The Velocity of CX at Periapsis is 7.317e-005 m/s
****The Period of CX is 6.584e+006 seconds
****The Shadow time for CX is 5.811e+011 seconds
****The Position of CX is ( 5.440e+001 m, 5.692e+001 m, 5.638e+001 m )
****The Distance from CX to Mars is 9.684e+001 meters

Example 8: GRUE-42 Matlab output from input2.
close all
clear all
clc

%Define the Gravitational Constant
G=6.67259E-11;

%Define the Planets, then the Satellites and their parameters
CX_a = 25.7;
CX_b = 23.0;
CX_c = -1;
CX_e = -1;
CX_Center = [55.0, 56.0, 66.0];
CX_Center_Mass = 456.0;
CX_Inclination = 23.5;
CX_Lifetime = 5.0;
CX_Epoch = 3298743298.0;

CX_name = 'CX';
%Define mu (G*M)
CX_mu = G*CX_Center_Mass;

%Determine which of the two variables was declared, and calculate the remaining two
[CX_a,CX_b,CX_c,CX_e]=IfStatement(CX_a,CX_b,CX_c,CX_e);
\end{verbatim}
fprintf('Results:
')
fprintf('---------------------------------------

% Now that we know everything, start doing the calculations
% Finding the Velocity at Apoapsis
CX_ra=CX_a+CX_c;
CX_Va=1000*sqrt(CX_mu/1E9)*((2000/CX_ra)-(1000/CX_a))^(1/2);
fprintf('****The Velocity of %s at Apoapsis is %.3d m/s',...
  CX_name,CX_Va)

% Finding the Velocity at Periapsis
CX_rp=CX_a-CX_c;
CX_Vp=1000*sqrt(CX_mu/1E9)*((2000/CX_rp)-(1000/CX_a))^(1/2);
fprintf('****The Velocity of %s at Periapsis is %.3d m/s',...
  CX_name,CX_Vp)

% Finding the position, period must be calculated first
CX_Time =CX_Epoch+100000;
CX_P=2*pi*sqrt(CX_a^3/CX_mu);
[x y z] = CalcPosition (CX_Time, CX_Inclination, CX_Epoch, CX_P, CX_a, CX_e);
CX_Position = CX_Center + [x y z];
fprintf('****The Position of %s is ( %.3d m, %.3d m, %.3d m )',...
  CX_name,CX_Position(1),CX_Position(2),CX_Position(3))

function [x y z] = CalcPosition (Time, Inclination, Epoch, P, a, e)
% Find the x, y, and z values
angle = Inclination;
percent = mod(Time-Epoch,P)/P;  % true anomaly
nu = 2*pi*percent;  
        r=a*(1-e^2)/(1+e*cos(nu));
x=r*cos(nu)*cos(angle);
y=r*sin(nu);
z=r*cos(nu)*sin(angle);

function [a b c e]=IfStatement(a, b, c, e)
% Determine which of the two variables was declared, and calculate the remaining two
if a==-1
  if b==-1
    c=sqrt(a^2-b^2);
    e=c/a;
  elseif c==-1
    e=c/a;
    b=a^sqrt(1-e^2);
  else
    b=a^sqrt(1-e^2);
    c=e*a;
  end
else
  if b==-1
    a=sqrt(b^2+c^2);
    e=c/a;
  else
    a=b^sqrt(1-e^2);
    c=e*a;
end

function [a b c e]=IfStatement(a, b, c, e)
% Determine which of the two variables was declared, and calculate the remaining two
if a==-1
  if b==-1
    c=sqrt(a^2-b^2);
    e=c/a;
  elseif c==-1
    e=c/a;
    b=a^sqrt(1-e^2);
  else
    b=a^sqrt(1-e^2);
    c=e*a;
  end
else
  if b==-1
    a=sqrt(b^2+c^2);
    e=c/a;
  else
    a=b^sqrt(1-e^2);
    c=e*a;
end 
else 
a=c/e; 
b=a*sqrt(1-e^2); 
end 

Example 9: Matlab Output from input2. 
Results: 
--------------------------------------- 
****The Velocity of CX at Apoapsis is 2.129e-005 m/s 
****The Velocity of CX at Periapsis is 5.560e-005 m/s 
****The Position of CX is ( 5.412e+001 m, 5.791e+001 m, 5.188e+001 m )