First VPLanet Developers Workshop

Lesson 6
How to Add an Option

Overview

Five steps:

- Determine which module(s) the option applies to
- Define a new integer ID
- Add text to InitializeOption
- Create Read function
- Add Verify step (if necessary)

The framework between outputs and options is similar

- We'll spend some time on Verify in this lecture

We'll go over reading in mass for this lecture

Step 1: Which Module(s)?

Mass is necessary for every module, so its ID goes in options.h

Step 2: Define and Integer ID

```
68
69  #define OPT_MASS 520
70  #define OPT_MASSRAD 525
71  #define OPT_MINVALUE 530
72
```

The syntax is almost identical to output, but OPT replaces OUT

If your new option is module specific, put it in the appropriate header file, e.g. poise.h

Step 3: Update InitializeOptions

As with output, we begin by defining values in the OPTIONS struct

```
4677
4678  void InitializeOptions(OPTIONS *options, fnReadOption *fnRead) {
4679    int iBody, iOpt, iFile, iModule;
4680
4681    /* Initialize all parameters describing the option's location */
4682    for (iOpt = 0; iOpt < MODULEOPTEND; iOpt++) {
4683      memset(options[iOpt].cName, '\0', OPTLEN);
4684      sprintf(options[iOpt].cName, "null");
4685      options[iOpt].iLine      = malloc(MAXFILES * sizeof(int));
4686      options[iOpt].bMultiFile = 0;
4687      options[iOpt].iMultiIn   = 0;
4688      options[iOpt].iType      = -1;
4689      memset(options[iOpt].cDescr, '\0', OPTDESCR);
4690      sprintf(options[iOpt].cDescr, "null");
4691      memset(options[iOpt].cLongDescr, '\0', OPTLONDESCR);
4692      sprintf(options[iOpt].cLongDescr, "null");
4693      memset(options[iOpt].cDefault, '\0', OPTDESCR);
4694      sprintf(options[iOpt].cDefault, "null");
4695      memset(options[iOpt].cValues, '\0', OPTDESCR);
4696      sprintf(options[iOpt].cValues, "null");
4697      memset(options[iOpt].cNeg, '\0', OPTDESCR);
4698      sprintf(options[iOpt].cNeg, "null");
4699      memset(options[iOpt].cDimension, '\0', OPTDESCR);
4700      options[iOpt].dDefault   = NAN;
4701      options[iOpt].iModuleBit = 0;
4702      options[iOpt].bNeg       = 0;
4703      options[iOpt].iFileType  = 2;
4704      options[iOpt].dNeg       = 0;
4705
4706      for (iFile = 0; iFile < MAXFILES; iFile++) {
4707        options[iOpt].iLine[iFile] = -1;
4708        memset(options[iOpt].cFile[iFile], '\0', OPTLEN);
4709        sprintf(options[iOpt].cFile[iFile], "null");
4710      }
4711    }
4712
4713    /* Now populate entries for general options. */
4714    InitializeOptionsGeneral(options, fnRead);
4715
```

Many fields relate to keeping track of if/where the option was found in a file

Others relate to negative units

Others keep track of the cast

Others declare which file(s) the option can exist in

Step 3: Update InitializeOptions

```
4376
4377    sprintf(options[OPT_MASS].cName, "dMass");
4378    sprintf(options[OPT_MASS].cDescr, "Mass");
4379    sprintf(options[OPT_MASS].cDefault, "1 Earth Mass");
4380    sprintf(options[OPT_MASS].cNeg, "Mearth");
4381    sprintf(options[OPT_MASS].cDimension, "mass");
4382    options[OPT_MASS].dDefault   = MEARTH;
4383    options[OPT_MASS].iType      = 2;
4384    options[OPT_MASS].bMultiFile = 1;
4385    options[OPT_MASS].dNeg       = MEARTH;
4386    options[OPT_MASS].iModuleBit = 0;
4387    options[OPT_MASS].bNeg       = 1;
4388    options[OPT_MASS].iFileType  = 1;
4389    fnRead[OPT_MASS]             = &ReadMass;
4390
```

The first set of lines define the strings associated with the option
(Note that inside VPLanet, string prefixes are “c” for char)
First, we set the option name, which must begin with the cast prefix
Then we set the description (pretty simple, no long description needed)
Next is a string describing the default value (Line 4379)
Then the negative unit, if applicable (use astropy conventions!)
Then the dimension (unit), for use with bigplanet

Step 3: Update InitializeOptions

```
4376
4377    sprintf(options[OPT_MASS].cName, "dMass");
4378    sprintf(options[OPT_MASS].cDescr, "Mass");
4379    sprintf(options[OPT_MASS].cDefault, "1 Earth Mass");
4380    sprintf(options[OPT_MASS].cNeg, "Mearth");
4381    sprintf(options[OPT_MASS].cDimension, "mass");
4382    options[OPT_MASS].dDefault   = MEARTH;
4383    options[OPT_MASS].iType      = 2;
4384    options[OPT_MASS].bMultiFile = 1;
4385    options[OPT_MASS].dNeg       = MEARTH;
4386    options[OPT_MASS].iModuleBit = 0;
4387    options[OPT_MASS].bNeg       = 1;
4388    options[OPT_MASS].iFileType  = 1;
4389    fnRead[OPT_MASS]             = &ReadMass;
4390
```

The next block of lines set the numerical values of the option

Line 4382 sets the default value (MEARTH defined in vplanet.h)

Then an integer for the type or cast:

- Boolean = 0
- Integer = 1
- Double = 2
- String = 3
- For an array, add 10

Step 3: Update InitializeOptions

```
4376
4377    sprintf(options[OPT_MASS].cName, "dMass");
4378    sprintf(options[OPT_MASS].cDescr, "Mass");
4379    sprintf(options[OPT_MASS].cDefault, "1 Earth Mass");
4380    sprintf(options[OPT_MASS].cNeg, "Mearth");
4381    sprintf(options[OPT_MASS].cDimension, "mass");
4382    options[OPT_MASS].dDefault   = MEARTH;
4383    options[OPT_MASS].iType      = 2;
4384    options[OPT_MASS].bMultiFile = 1;
4385    options[OPT_MASS].dNeg       = MEARTH;
4386    options[OPT_MASS].iModuleBit = 0;
4387    options[OPT_MASS].bNeg       = 1;
4388    options[OPT_MASS].iFileType  = 1;
4389    fnRead[OPT_MASS]             = &ReadMass;
4390
```

Next on line 4384, we tell VPLanet this option can exist in multiple files
Then is the conversion factor for negative arguments
The iModuleBit field defines which modules the option applies to
    - Each module has a unique bit; options that apply to all are set to 0
Line 4387 lets VPLanet know that negative options are allowed
The iFileType distinguishes between primary file (0) and body file (1)
Finally, we define the ReadOption function for the function pointer array

Step 4: Write the ReadOption Function

```
2399
2400  void ReadMass(BODY *body, CONTROL *control, FILES *files, OPTIONS *options,
2401                SYSTEM *system, int iFile) {
2402    /* This parameter cannot exist in primary file */
2403    /* Must verify in conjuction with Radius, Density and MassRad */
2404    int lTmp = -1;
2405    double dTmp;
2406
2407    AddOptionDouble(files->Infile[iFile].cIn, options->cName, &dTmp, &lTmp,
2408                    control->Io.iVerbose);
2409    if (lTmp >= 0) {
2410      NotPrimaryInput(iFile, options->cName, files->Infile[iFile].cIn, lTmp,
2411                      control->Io.iVerbose);
2412      if (dTmp < 0) {
2413        body[iFile - 1].dMass =
2414              dTmp * dNegativeDouble(*options, files->Infile[iFile].cIn,
2415                                     control->Io.iVerbose);
2416      } else {
2417        body[iFile - 1].dMass = dTmp * fdUnitsMass(control->Units[iFile].iMass);
2418      }
2419      UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
2420    } else if (iFile > 0) {
2421      body[iFile - 1].dMass = options->dDefault;
2422    }
2423  }
2424
```

ReadOptions must all have the same argument list

lTmp is the line number

This function reads a double, which is placed in dTmp

- other options are bTmp, iTmp and cTmp (plus arrays)

Step 4: Write the ReadOption Function

```
2399
2400  void ReadMass(BODY *body, CONTROL *control, FILES *files, OPTIONS *options,
2401                SYSTEM *system, int iFile) {
2402    /* This parameter cannot exist in primary file */
2403    /* Must verify in conjuction with Radius, Density and MassRad */
2404    int lTmp = -1;
2405    double dTmp;
2406
2407    AddOptionDouble(files->Infile[iFile].cIn, options->cName, &dTmp, &lTmp,
2408                    control->Io.iVerbose);
2409    if (lTmp >= 0) {
2410      NotPrimaryInput(iFile, options->cName, files->Infile[iFile].cIn, lTmp,
2411                      control->Io.iVerbose);
2412      if (dTmp < 0) {
2413        body[iFile - 1].dMass =
2414              dTmp * dNegativeDouble(*options, files->Infile[iFile].cIn,
2415                                     control->Io.iVerbose);
2416      } else {
2417        body[iFile - 1].dMass = dTmp * fdUnitsMass(control->Units[iFile].iMass);
2418      }
2419      UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
2420    } else if (iFile > 0) {
2421      body[iFile - 1].dMass = options->dDefault;
2422    }
2423  }
2424
```

AddOptionDouble searches for the option

If found, the argument goes in dTmp, line number in lTmp

If option not a double, use the appropriate cast (examples to come)

Step 4: Write the ReadOption Function

```
2399
2400  void ReadMass(BODY *body, CONTROL *control, FILES *files, OPTIONS *options,
2401                SYSTEM *system, int iFile) {
2402    /* This parameter cannot exist in primary file */
2403    /* Must verify in conjuction with Radius, Density and MassRad */
2404    int lTmp = -1;
2405    double dTmp;
2406
2407    AddOptionDouble(files->Infile[iFile].cIn, options->cName, &dTmp, &lTmp,
2408                    control->Io.iVerbose);
2409    if (lTmp >= 0) {
2410      NotPrimaryInput(iFile, options->cName, files->Infile[iFile].cIn, lTmp,
2411                      control->Io.iVerbose);
2412      if (dTmp < 0) {
2413        body[iFile - 1].dMass =
2414              dTmp * dNegativeDouble(*options, files->Infile[iFile].cIn,
2415                                     control->Io.iVerbose);
2416      } else {
2417        body[iFile - 1].dMass = dTmp * fdUnitsMass(control->Units[iFile].iMass);
2418      }
2419      UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
2420    } else if (iFile > 0) {
2421      body[iFile - 1].dMass = options->dDefault;
2422    }
2423  }
2424
```

If the option is found, lTmp >= 0, and the first if-then block is entered

NotPrimaryInput checks if the option is in the Primary Input File

- This is not checked automatically, so include if necessary

Step 4: Write the ReadOption Function

```
2399
2400  void ReadMass(BODY *body, CONTROL *control, FILES *files, OPTIONS *options,
2401                SYSTEM *system, int iFile) {
2402    /* This parameter cannot exist in primary file */
2403    /* Must verify in conjuction with Radius, Density and MassRad */
2404    int lTmp = -1;
2405    double dTmp;
2406
2407    AddOptionDouble(files->Infile[iFile].cIn, options->cName, &dTmp, &lTmp,
2408                    control->Io.iVerbose);
2409    if (lTmp >= 0) {
2410      NotPrimaryInput(iFile, options->cName, files->Infile[iFile].cIn, lTmp,
2411                      control->Io.iVerbose);
2412      if (dTmp < 0) {
2413        body[iFile - 1].dMass =
2414              dTmp * dNegativeDouble(*options, files->Infile[iFile].cIn,
2415                                     control->Io.iVerbose);
2416      } else {
2417        body[iFile - 1].dMass = dTmp * fdUnitsMass(control->Units[iFile].iMass);
2418      }
2419      UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
2420    } else if (iFile > 0) {
2421      body[iFile - 1].dMass = options->dDefault;
2422    }
2423  }
2424
```

The next lines assign the member of the BODY struct

Since dMass has negative option, if negative do the conversion

If positive, adjust value based on user-defined units

Note that offset between body number and file number!

Step 4: Write the ReadOption Function

```
2399
2400  void ReadMass(BODY *body, CONTROL *control, FILES *files, OPTIONS *options,
2401                SYSTEM *system, int iFile) {
2402    /* This parameter cannot exist in primary file */
2403    /* Must verify in conjuction with Radius, Density and MassRad */
2404    int lTmp = -1;
2405    double dTmp;
2406
2407    AddOptionDouble(files->Infile[iFile].cIn, options->cName, &dTmp, &lTmp,
2408                    control->Io.iVerbose);
2409    if (lTmp >= 0) {
2410      NotPrimaryInput(iFile, options->cName, files->Infile[iFile].cIn, lTmp,
2411                      control->Io.iVerbose);
2412      if (dTmp < 0) {
2413        body[iFile - 1].dMass =
2414              dTmp * dNegativeDouble(*options, files->Infile[iFile].cIn,
2415                                     control->Io.iVerbose);
2416      } else {
2417        body[iFile - 1].dMass = dTmp * fdUnitsMass(control->Units[iFile].iMass);
2418      }
2419      UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
2420    } else if (iFile > 0) {
2421      body[iFile - 1].dMass = options->dDefault;
2422    }
2423  }
2424
```

UpdateFoundOption records the position of the option in the file

Finally, if the option was not found, assign the default

All options should have a default value; can lead to user error!

Step 4: Write the ReadOption Function

```
1643
1644  void ReadDigits(BODY *body, CONTROL *control, FILES *files, OPTIONS *options,
1645                  SYSTEM *system, int iFile) {
1646    /* This parameter can exist in any file, but only once */
1647    int lTmp = -1;
1648    int iTmp;
1649
1650    AddOptionInt(files->Infile[iFile].cIn, options->cName, &iTmp, &lTmp,
1651                 control->Io.iVerbose);
1652    if (lTmp >= 0) {
1653      /* Option was found */
1654      CheckDuplication(files, options, files->Infile[iFile].cIn, lTmp,
1655                       control->Io.iVerbose);
1656      control->Io.iDigits = iTmp;
1657      if (control->Io.iDigits < 0) {
1658        if (control->Io.iVerbose >= VERBERR) {
1659          fprintf(stderr, "ERROR: %s must be non-negative.\n", options->cName);
1660        }
1661        LineExit(files->Infile[iFile].cIn, options->iLine[iFile]);
1662      }
1663      if (control->Io.iDigits > 16) {
1664        if (control->Io.iVerbose >= VERBERR) {
1665          fprintf(stderr, "ERROR: %s must be less than 17.\n", options->cName);
1666        }
1667        LineExit(files->Infile[iFile].cIn, options->iLine[iFile]);
1668      }
1669      control->Io.iDigits = iTmp;
1670      UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
1671    } else {
1672      AssignDefaultInt(options, &control->Io.iDigits, files->iNumInputs);
1673    }
1674  }
1675
```

Reading integer options has similar format

Here we assign the # of digits to be output to the CONTROL struct

Step 4: Write the ReadOption Function

```
1643
1644  void ReadDigits(BODY *body, CONTROL *control, FILES *files, OPTIONS *options,
1645                  SYSTEM *system, int iFile) {
1646    /* This parameter can exist in any file, but only once */
1647    int lTmp = -1;
1648    int iTmp;
1649
1650    AddOptionInt(files->Infile[iFile].cIn, options->cName, &iTmp, &lTmp,
1651                 control->Io.iVerbose);
1652    if (lTmp >= 0) {
1653      /* Option was found */
1654      CheckDuplication(files, options, files->Infile[iFile].cIn, lTmp,
1655                       control->Io.iVerbose);
1656      control->Io.iDigits = iTmp;
1657      if (control->Io.iDigits < 0) {
1658        if (control->Io.iVerbose >= VERBERR) {
1659          fprintf(stderr, "ERROR: %s must be non-negative.\n", options->cName);
1660        }
1661        LineExit(files->Infile[iFile].cIn, options->iLine[iFile]);
1662      }
1663      if (control->Io.iDigits > 16) {
1664        if (control->Io.iVerbose >= VERBERR) {
1665          fprintf(stderr, "ERROR: %s must be less than 17.\n", options->cName);
1666        }
1667        LineExit(files->Infile[iFile].cIn, options->iLine[iFile]);
1668      }
1669      control->Io.iDigits = iTmp;
1670      UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
1671    } else {
1672      AssignDefaultInt(options, &control->Io.iDigits, files->iNumInputs);
1673    }
1674  }
1675
```

This option can only be defined once, so we check for duplication

We also check that the argument is in range

LineExit provides a helpful error message

Step 4: Write the ReadOption Function

```
446
447 void ReadMagmOcAtmModel(BODY *body, CONTROL *control, FILES *files,
448                          OPTIONS *options, SYSTEM *system, int iFile) {
449   /* This parameter cannot exist in primary file */
450   int lTmp = -1;
451   char cTmp[OPTLEN];
452
453   AddOptionString(files->Infile[iFile].cIn, options->cName, cTmp, &lTmp,
454                   control->Io.iVerbose);
455   if (lTmp >= 0) {
456     NotPrimaryInput(iFile, options->cName, files->Infile[iFile].cIn, lTmp,
457                     control->Io.iVerbose);
458     if (!memcmp(sLower(cTmp), "petit", 4)) {
459       body[iFile - 1].iMagmOcAtmModel = MAGMOC_PETIT;
460     } else if (!memcmp(sLower(cTmp), "grey", 4)) {
461       body[iFile - 1].iMagmOcAtmModel = MAGMOC_GREY;
462     }
463     UpdateFoundOption(&files->Infile[iFile], options, lTmp, iFile);
464   } else if (iFile > 0) {
465     body[iFile - 1].iMagmOcAtmModel = MAGMOC_GREY;
466   }
467 }
468
```

Reading in strings is slightly more complicated

- C is not good at handling strings

In general strings are converted to integers in a struct

Please #define IDs; no “magic numbers”!

From ReadOptions to VerifyOptions

In the ReadOption function, you can do minimal checking of arguments
- Unphysical values
- Out of bounds
- Undefined strings

Verify is for more complicated checks across multiple options

Verify also initializes many values, couples modules, and finishes setting up structs

At the end of Verify, the options *must* be fully vetted, and an integration is ready to begin

Top-Level Verify Function

```
1103
1104  void VerifyOptions(BODY *body, CONTROL *control, FILES *files, MODULE *module,
1105                     OPTIONS *options, OUTPUT *output, SYSTEM *system,
1106                     UPDATE *update, fnIntegrate *fnOneStep,
1107                     fnUpdateVariable ****fnUpdate) {
1108
1109    int iBody, iModule;
1110
1111    VerifyAge(body, control, options);
1112    VerifyNames(body, control, options);
1113
1114    // Need to know integration type before we can initialize CONTROL
1115    VerifyIntegration(body, control, files, options, system, fnOneStep);
1116    InitializeControlEvolve(body, control, module, update);
1117
1118    // Allocate all memory for the BODY struct, based on selected modules
1119    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1120      for (iModule = 0; iModule < module->iNumModules[iBody]; iModule++) {
1121        module->fnInitializeBody[iBody][iModule](body, control, update, iBody,
1122                                                 iModule);
1123      }
1124    }
1125
1126    // Verify physical and orbital properties
1127    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1128      /* Must verify density first: RotVel requires a radius in VerifyRotation */
1129      VerifyMassRad(&body[iBody], control, options, files->Infile[iBody].cIn,
1130                    iBody);
1131      VerifyRotationGeneral(body, options, files->Infile[iBody + 1].cIn, iBody,
1132                            control->Io.iVerbose);
1133
1134      // If any bodies orbit, make sure they do so properly!
1135      if (control->bOrbiters) {
1136        VerifyOrbit(body, control, *files, options, iBody);
1137      }
1138
1139      VerifyLayers(body, control, files, options, iBody);
1140    }
1141
1142    InitializeUpdate(body, control, module, update, fnUpdate);
1143    InitializeHalts(control, module);
1144
1145    VerifyHalts(body, control, module, options);
1146
```

```
1146
1147    // Now verify the modules' parameters
1148    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1149      for (iModule = 0; iModule < module->iNumModules[iBody]; iModule++) {
1150        module->fnVerify[iBody][iModule](body, control, files, options, output,
1151                                         system, update, iBody, iModule);
1152      }
1153
1154      // Verify multi-module couplings
1155      VerifyModuleMulti(body, update, control, files, module, options, iBody,
1156                        fnUpdate);
1157
1158      for (iModule = 0; iModule < module->iNumManageDerivs[iBody]; iModule++) {
1159        module->fnAssignDerivatives[iBody][iModule](body, &(control->Evolve),
1160                                                    update, *fnUpdate, iBody);
1161      }
1162
1163      /* Must allocate memory in control struct for all perturbing bodies */
1164      if (control->Evolve.iOneStep == RUNGEKUTTA) {
1165        InitializeUpdateBodyPerts(control, update, iBody);
1166        InitializeUpdateTmpBody(body, control, module, update, iBody);
1167      }
1168    }
1169
1170    // Verify multi-module parameters
1171    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1172      if (body[iBody].bEqtide) {
1173        VerifyImK2(body, control, files, options, system, update, iBody);
1174      }
1175    }
1176
1177    // Initialize angular momentum and energy prior to logging/integration
1178    InitializeConstants(body, update, control, system, options);
1179
1180    // Set next output time so logging does not contain a memory leak
1181    // control->Io.dNextOutput = control->Evolve.dTime + control->Io.dOutputTime;
1182
1183    // Finally, initialize derivative values -- this avoids leaks while logging
1184    PropertiesAuxiliary(body, control, system, update);
1185    CalculateDerivatives(body, system, update, *fnUpdate,
1186                         control->Evolve.iNumBodies);
1187
1188    control->Evolve.dTime   = 0;
1189    control->Evolve.nSteps  = 0;
1190    control->Io.dNextOutput = control->Evolve.dTime + control->Io.dOutputTime;
1191  }
1192
```

Top-Level Verify Function

```
1103
1104  void VerifyOptions(BODY *body, CONTROL *control, FILES *files, MODULE *module,
1105                     OPTIONS *options, OUTPUT *output, SYSTEM *system,
1106                     UPDATE *update, fnIntegrate *fnOneStep,
1107                     fnUpdateVariable ****fnUpdate) {
1108
1109    int iBody, iModule;
1110
1111    VerifyAge(body, control, options);
1112    VerifyNames(body, control, options);
1113
1114    // Need to know integration type before we can initialize CONTROL
1115    VerifyIntegration(body, control, files, options, system, fnOneStep);
1116    InitializeControlEvolve(body, control, module, update);
1117
1118    // Allocate all memory for the BODY struct, based on selected modules
1119    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1120      for (iModule = 0; iModule < module->iNumModules[iBody]; iModule++) {
1121        module->fnInitializeBody[iBody][iModule](body, control, update, iBody,
1122                                                 iModule);
1123      }
1124    }
1125
1126    // Verify physical and orbital properties
1127    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1128      /* Must verify density first: RotVel requires a radius in VerifyRotation */
1129      VerifyMassRad(&body[iBody], control, options, files->Infile[iBody].cIn,
1130                    iBody);
1131      VerifyRotationGeneral(body, options, files->Infile[iBody + 1].cIn, iBody,
1132                            control->Io.iVerbose);
1133
1134      // If any bodies orbit, make sure they do so properly!
1135      if (control->bOrbiters) {
1136        VerifyOrbit(body, control, *files, options, iBody);
1137      }
1138
1139      VerifyLayers(body, control, files, options, iBody);
1140    }
1141
1142    InitializeUpdate(body, control, module, update, fnUpdate);
1143    InitializeHalts(control, module);
1144
1145    VerifyHalts(body, control, module, options);
1146
```

The order of these functions is important!

The first half of Verify is pretty straight-forward

Initializing CONTROL, filling out members of structs, etc.

Top-Level Verify Function

Things get more complicated in the second half

VPLanet must ensure module combinations are OK

VerifyImK2 ensures interiors are self-consistent (tidal Q depends on temperature, but temperature depends on tidal Q)

At the end, calculate AuxProps and derivatives

Finally, set first step of the integration

```
1146
1147    // Now verify the modules' parameters
1148    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1149      for (iModule = 0; iModule < module->iNumModules[iBody]; iModule++) {
1150        module->fnVerify[iBody][iModule](body, control, files, options, output,
1151                                         system, update, iBody, iModule);
1152      }
1153
1154      // Verify multi-module couplings
1155      VerifyModuleMulti(body, update, control, files, module, options, iBody,
1156                        fnUpdate);
1157
1158      for (iModule = 0; iModule < module->iNumManageDerivs[iBody]; iModule++) {
1159        module->fnAssignDerivatives[iBody][iModule](body, &(control->Evolve),
1160                                                    update, *fnUpdate, iBody);
1161      }
1162
1163      /* Must allocate memory in control struct for all perturbing bodies */
1164      if (control->Evolve.iOneStep == RUNGEKUTTA) {
1165        InitializeUpdateBodyPerts(control, update, iBody);
1166        InitializeUpdateTmpBody(body, control, module, update, iBody);
1167      }
1168    }
1169
1170    // Verify multi-module parameters
1171    for (iBody = 0; iBody < control->Evolve.iNumBodies; iBody++) {
1172      if (body[iBody].bEqtide) {
1173        VerifyImK2(body, control, files, options, system, update, iBody);
1174      }
1175    }
1176
1177    // Initialize angular momentum and energy prior to logging/integration
1178    InitializeConstants(body, update, control, system, options);
1179
1180    // Set next output time so logging does not contain a memory leak
1181    // control->Io.dNextOutput = control->Evolve.dTime + control->Io.dOutputTime;
1182
1183    // Finally, initialize derivative values -- this avoids leaks while logging
1184    PropertiesAuxiliary(body, control, system, update);
1185    CalculateDerivatives(body, system, update, *fnUpdate,
1186                         control->Evolve.iNumBodies);
1187
1188    control->Evolve.dTime   = 0;
1189    control->Evolve.nSteps  = 0;
1190    control->Io.dNextOutput = control->Evolve.dTime + control->Io.dOutputTime;
1191  }
1192
```

Digging into Verify

```
520 void VerifyMassRad(BODY *body, CONTROL *control, OPTIONS *options, char cFile[],
521                   int iBody) {
522   int iFile = iBody + 1, iVerbose;
523
524   iVerbose = control->Io.iVerbose;
525
526   /* !!!!! --- Mass and Radius ARE ALWAYS UPDATED AND CORRECT --- !!!!! */
527
528   /* First see if mass and radius and nothing else set, i.e. the user input the
529    * default parameters */
530   if (options[OPT_MASS].iLine[iFile] > -1 &&
531       options[OPT_RADIUS].iLine[iFile] > -1 &&
532       options[OPT_DENSITY].iLine[iFile] == -1 &&
533       options[OPT_MASSRAD].iLine[iFile] == -1) {
534     return;
535   }
536
537   /* Was anything set> */
538   if (options[OPT_MASS].iLine[iFile] == -1 &&
539       options[OPT_RADIUS].iLine[iFile] == -1 &&
540       options[OPT_DENSITY].iLine[iFile] == -1) {
541     if (iVerbose >= VERBERR) {
542       fprintf(stderr, "ERROR: Must set at least one of %s, %s, and %s.\n",
543               options[OPT_MASS].cName, options[OPT_RADIUS].cName,
544               options[OPT_DENSITY].cName);
545     }
546     exit(EXIT_INPUT);
547   }
548
549   /* Were all set? */
550   if (options[OPT_MASS].iLine[iFile] > -1 &&
551       options[OPT_RADIUS].iLine[iFile] > -1 &&
552       options[OPT_DENSITY].iLine[iFile] > -1) {
553     VerifyTripleExit(options[OPT_MASS].cName, options[OPT_RADIUS].cName,
554                      options[OPT_DENSITY].cName, options[OPT_MASS].iLine[iFile],
555                      options[OPT_RADIUS].iLine[iFile],
556                      options[OPT_DENSITY].iLine[iFile], cFile, iVerbose);
557     exit(EXIT_INPUT);
558   }
559
560   /* Was mass set? */
561   if (options[OPT_MASS].iLine[iFile] > -1) {
562
563     /* Can only set 1 other */
564     if (options[OPT_RADIUS].iLine[iFile] > -1 &&
565         options[OPT_MASSRAD].iLine[iFile] > -1) {
566       VerifyTwoOfThreeExit(options[OPT_MASS].cName, options[OPT_RADIUS].cName,
567                            options[OPT_MASSRAD].cName,
568                            options[OPT_MASS].iLine[iFile],
569                            options[OPT_RADIUS].iLine[iFile],
570                            options[OPT_MASSRAD].iLine[iFile], cFile, iVerbose);
571     }
572
```

The user can set the following:

- Mass
- Radius
- Density
- A mass-radius relationship

The code only keeps track of mass and radius

VerifyMassRad converts 4 options into 2 BODY struct members

Note functions like TripleLineExit and VerifyTwoOfThreeExit

Function continues…

Finishing Up Your New Option

As with outputs, please write unit tests so your code will continue to work

Note that for coverage statistics, exit lines should be excluded:

```
if (dTmp < 0) {
  if (control->Io.iVerbose >= VERBERR) { // LCOV_EXCL_LINE
    fprintf(stderr,"ERROR: %s must be >= 0.\n",options->cName); //LCOV_EXCL_LINE
  }
  LineExit(files->Infile[iFile].cIn,lTmp); // LCOV_EXCL_LINE
}
```

or

```
if (dTmp < 0) {
  // LCOV_EXCL_START
  if (control->Io.iVerbose >= VERBERR) {
    fprintf(stderr,"ERROR: %s must be >= 0.\n",options->cName);
  }
  LineExit(files->Infile[iFile].cIn,lTmp);
  // LCOV_EXCL_STOP
}
```