# First VPLanet Developers Workshop

## Lesson 8
## Coupling Modules

# Overview

VPLanet has 12 modules currently available, but not all are coupled

Among those that are coupled, not all models are coupled

Many aspects of coupling are unique to the modules involved

Three basic steps to module coupling:

- Write the VerifyMulti function
- Write ForceBehaviorMulti
- Write PropsAuxMulti

Here we assume the modules already exist, next lecture on building a new module

# CONTROL Function Pointer Matrices

```
1747    fnForceBehaviorModule *
1748          *fnForceBehavior; /**< Function Pointers to Force Behaviors */
1749    fnForceBehaviorModule *
1750          *fnForceBehaviorMulti; /**< Function Pointers to Force Behaviors */
1751    int *iNumMultiForce; /**< Number of multi-module ForceBahevior functions */
1752
1753    fnPropsAuxModule *
1754          *fnPropsAux; /**< Function Pointers to Auxiliary Properties */
1755    fnPropsAuxModule *
1756          *fnPropsAuxMulti; /**< Function pointers to Auxiliary Properties for
1757                               multi-module interdependancies. */
1758    int *iNumMultiProps;    /**< Number of Multi-module PropsAux functions */
1759
```

The CONTROL struct contains function pointer matrices for ForceBehavior and PropsAux

We divide them between single module and multi-module members

Single module functions called first

In multi-module functions, you can overwrite single-module instructions

# VerifyModuleMulti

We assume here that no new options are necessary to couple the modules (see Lesson 6 to add them)

Thus, coupling begins in Verify with call to VerifyModuleMulti

```
1740
1741  void VerifyModuleMulti(BODY *body, UPDATE *update, CONTROL *control,
1742                         FILES *files, MODULE *module, OPTIONS *options,
1743                         int iBody, fnUpdateVariable ****fnUpdate) {
1744
1745    int iNumMultiProps = 0, iNumMultiForce = 0;
1746
1747    if (module->iNumModules[iBody] > 0) {
1748      /* XXX Note that the number of elements here is really a permutation,
1749         but this should work for a while. */
1750      control->fnPropsAuxMulti[iBody] =
1751            malloc(2 * module->iNumModules[iBody] * sizeof(fnPropsAuxModule *));
1752      control->fnForceBehaviorMulti[iBody] = malloc(
1753            2 * module->iNumModules[iBody] * sizeof(fnForceBehaviorModule *));
1754    }
1755
```

Note the hack in memory allocation! This approach is not scalable

# VerifyModuleMulti

```
1780
1781    VerifyModuleMultiAtmescEqtide(body, update, control, files, module, options,
1782                                  iBody, &iNumMultiProps, &iNumMultiForce);
1783
1784    VerifyModuleMultiEqtideThermint(body, update, control, files, module, options,
1785                                    iBody, &iNumMultiProps, &iNumMultiForce);
1786
1787    // Always call after VerifyModuleMultiEqtideThermint !!
1788    VerifyModuleMultiAtmescEqtideThermint(body, update, control, files, module,
1789                                          options, iBody, &iNumMultiProps,
1790                                          &iNumMultiForce);
1791
1792    VerifyModuleMultiFlareStellar(body, update, control, files, module, options,
1793                                  iBody, &iNumMultiProps, &iNumMultiForce);
1794
```

VerifyMulti functions called manually in module.c

The order is important!

Note that double, triple, etc. multis are allowed

Always place larger multiples after smaller multiples

- AtmEscEqtideThermint after AtmescEqtide and EqtideThermint

# VerifyModuleMulti

```
1558
1559  void VerifyModuleMultiSpiNBodyDistOrb(BODY *body, UPDATE *update,
1560                                        CONTROL *control, FILES *files,
1561                                        OPTIONS *options, int iBody,
1562                                        int *iModuleProps, int *iModuleForce) {
1563    int iTmpBody;
1564    // This gets done repeatedly, but should be only done once
1565    control->Evolve.bSpiNBodyDistOrb = 0;
1566
1567    // Since the star will not have DistOrb called, only check for planets
1568    for (iTmpBody = 1; iTmpBody < control->Evolve.iNumBodies; iTmpBody++) {
1569      if (body[iTmpBody].bSpiNBody && body[iTmpBody].bDistOrb) {
1570        control->Evolve.bSpiNBodyDistOrb = 1;
1571        // Start with DistOrb, not SpiNBody
1572        control->Evolve.bUsingDistOrb  = 1;
1573        control->Evolve.bUsingSpiNBody = 0;
1574
1575        body[iTmpBody].dMeanL = body[iTmpBody].dMeanA + body[iTmpBody].dLongP;
1576      }
1577    }
1578    if (body[iBody].bSpiNBody && body[iBody].bDistOrb) {
1579      control->fnPropsAuxMulti[iBody][(*iModuleProps)++] =
1580            &PropsAuxSpiNBodyDistOrb;
1581      control->fnForceBehaviorMulti[iBody][(*iModuleForce)++] =
1582            &ForceBehaviorSpiNBodyDistOrb;
1583    }
1584  }
1585
```

Typically you perform some checks and set a few struct parameters
Also can set PropsAuxMulti and ForceBehaviorMulti

# VerifyModuleMulti

```
1490
1491  void VerifyModuleMultiFlareStellar(BODY *body, UPDATE *update, CONTROL *control,
1492                                     FILES *files, MODULE *module,
1493                                     OPTIONS *options, int iBody,
1494                                     int *iModuleProps, int *iModuleForce) {
1495
1496    if (body[iBody].bFlare) {
1497      if (!body[iBody].bStellar) {
1498        fprintf(stderr,
1499                "ERROR: Must include module STELLAR to run module FLARE.\n");
1500        LineExit(files->Infile[iBody + 1].cIn,
1501                 options[OPT_MODULES].iLine[iBody + 1]);
1502      } else {
1503        control->fnPropsAuxMulti[iBody][(*iModuleProps)++] =
1504              &PropsAuxFlareStellar;
1505      }
1506    }
1507  }
1508
```

Regardless of which modules selected, *every* VerifyMulti function called
If a module depends on another, you can check it here
You can also change variables based on the selected models

Verify is hard, VerifyMulti is even harder — be careful!

# PropsAuxMulti

```
1854
1855  void PropsAuxEqtideDistorb(BODY *body, EVOLVE *evolve, IO *io, UPDATE *update,
1856                             int iBody) {
1857    body[iBody].dEccSq = body[iBody].dHecc * body[iBody].dHecc +
1858                         body[iBody].dKecc * body[iBody].dKecc;
1859  }
1860
1861  void PropsAuxEqtideDistRot(BODY *body, EVOLVE *evolve, IO *io, UPDATE *update,
1862                             int iBody) {
1863    if (body[iBody].bCalcDynEllip) {
1864      body[iBody].dDynEllip = CalcDynEllipEq(body, iBody);
1865    }
1866  }
1867
```

PropsAuxMulti function behaves just like normal PropsAux

Calculate/define parameters necessary for the integration

dEccSq is necessary for EqTide

When EqTide+DistRot coupled, change the shape of the body to assume hydrostatic equilibrium

PropsAuxMulti functions called *after* single-module PropsAux

Can also add code for different model choices here

# ForceBehaviorMulti

```
2122
2123  void ForceBehaviorEqtideAtmesc(BODY *body, MODULE *module, EVOLVE *evolve,
2124                                 IO *io, SYSTEM *system, UPDATE *update,
2125                                 fnUpdateVariable ***fnUpdate, int iBody,
2126                                 int iModule) {
2127
2128    // We think there is an envelope, but there isn't!
2129    if (body[iBody].bEnv &&
2130        (body[iBody].dEnvelopeMass <= body[iBody].dMinEnvelopeMass)) {
2131      if (io->iVerbose >= VERBPROG) {
2132        fprintf(stderr, "%s's envelope lost at t = %.2e years!\n",
2133                body[iBody].cName, evolve->dTime / YEARSEC);
2134      }
2135      body[iBody].bEnv     = 0;
2136      body[iBody].dImK2Env = 0;
2137      // Adjust Im(k_2)
2138      body[iBody].dImK2 = fdImK2Total(body, iBody);
2139    }
2140
```

Example: If a planet loses its envelope, its tidal Q should change

- Gas giants have Q ~1e6, terrestrial have Q ~ 100

Users can specify these values, but code needs to know when to change

This complicated if-then structure is an ideal ForceBehavior feature

ForceBehaviorMulti called after the single-module ForceBehaviors

(This functionality works, validated, but still no example!)

# Update Module Compatibility

After coupling two (or more) existing modules, you probably need to modify VerifyModuleCompatibility

```
1610
1611  /** Verify that selected modules are compatable */
1612
1613  void VerifyModuleCompatability(BODY *body, CONTROL *control, FILES *files,
1614                                 MODULE *module, OPTIONS *options, int iBody) {
1615
1616    // Binary
1617    if (body[iBody].bBinary) {
1618      if (body[iBody].bSpiNBody) {
1619        if (control->Io.iVerbose >= VERBERR) {
1620          fprintf(stderr, "ERROR: Modules Binary and SpiNbody cannot be applied "
1621                          "to the same body.\n");
1622        }
1623        LineExit(files->Infile[iBody + 1].cIn,
1624                 options[OPT_MODULES].iLine[iBody + 1]);
1625      }
1626      if (body[iBody].bPoise) {
1627        if (control->Io.iVerbose >= VERBERR) {
1628          fprintf(stderr, "ERROR: Modules Binary and Poise cannot be applied to "
1629                          "the same body.\n");
1630        }
1631        LineExit(files->Infile[iBody + 1].cIn,
1632                 options[OPT_MODULES].iLine[iBody + 1]);
1633      }
1634    }
1635
```

# Multi-Module Primary Variables

We've encountered one case in which coupling modules requires a new primary variable (STELLAR + EqTide)

- BINARY and POISE probably needs it, too, for example

VPLanet also has a framework for adding them, too

```
595
596  void AddModulesMulti(BODY *body, CONTROL *control, MODULE *module, int iBody,
597                       int *iModule) {
598
599    if (body[iBody].bEqtide && body[iBody].bStellar) {
600      module->fnAssignDerivatives[iBody][*iModule] =
601            &AssignEqtideStellarDerivatives;
602      module->fnNullDerivatives[iBody][*iModule] = &NullEqtideStellarDerivatives;
603      module->iaEqtideStellar[iBody]             = *iModule;
604      (*iModule)++;
605    }
606  }
607
```

From here, it's mostly like adding a simple primary variable, but put the functions in module.c