ECCE Code Registration

Gary Black

Pacific Northwest National Lab

gary.black@pnl.gov

May 15, 2008

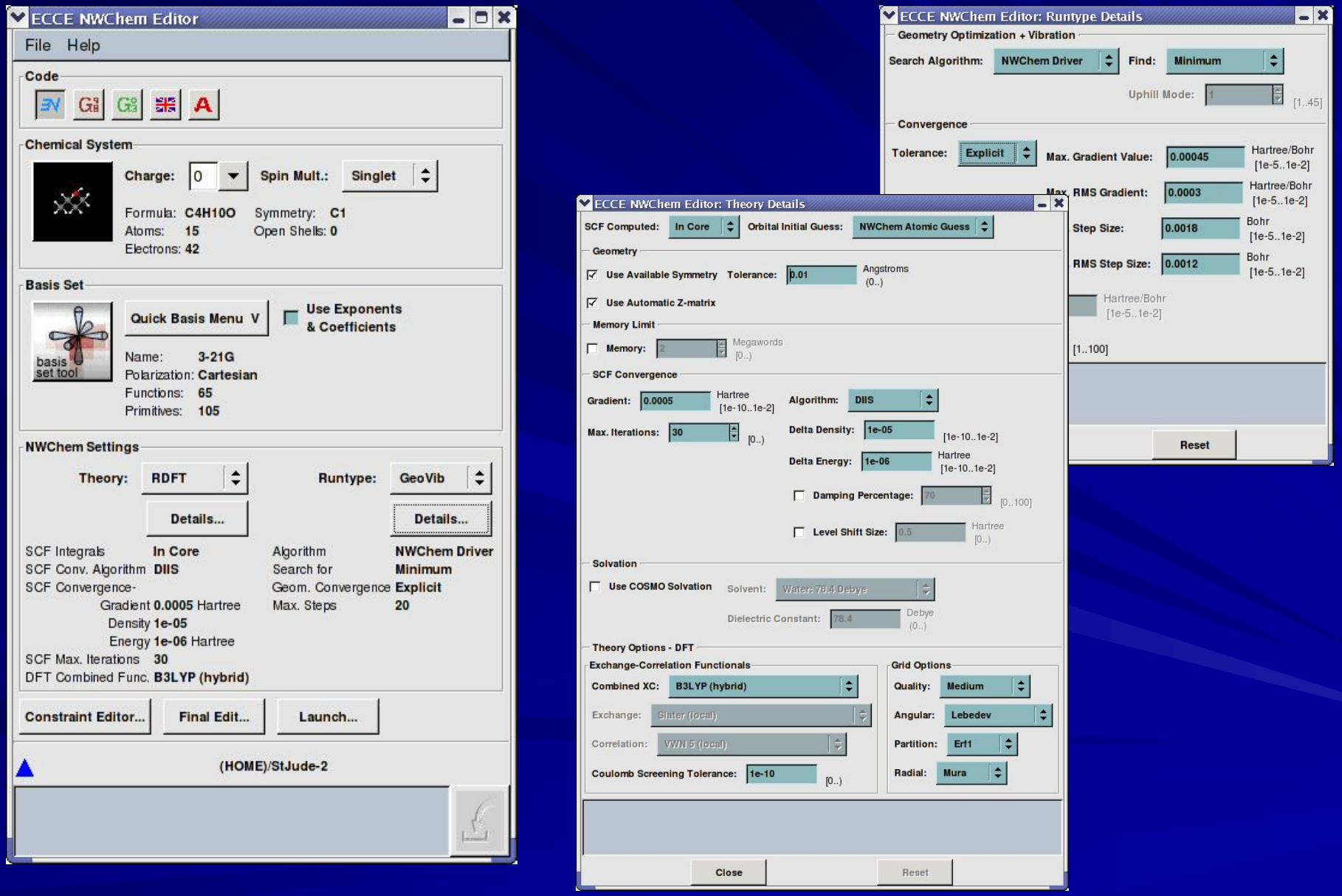
Overview

- Code Registration Manual (update in progress)
  - http://ecce.pnl.gov ➔ Use ECCE ➔ Code Registration link in Related Documents
- Regular ECCE distribution includes everything needed to register a new code or extend support for an existing code
- Distribution includes the complete set of registration files for each ECCE code
  - NWChem, GAMESS-UK, Gaussian 03 and 98™, AMICA
- Simple Python and Perl scripting needed, as well as creating/modifying XML and custom format data files
- Precedent established for how to accomplish different pieces of code registration by existing codes, but design allows other approaches

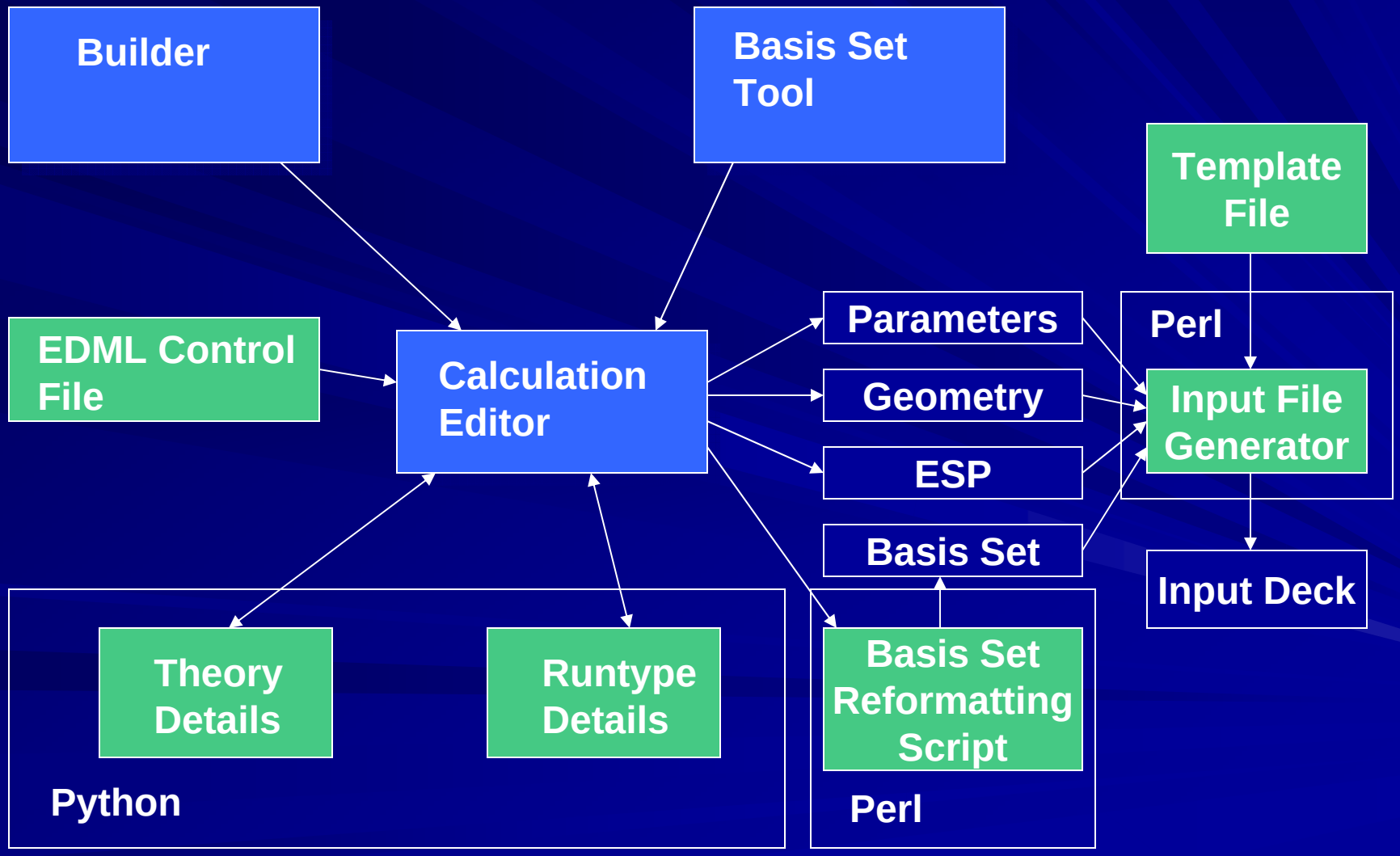
Outline

- Top-level code registration control file (EDML)
- Creating GUI details dialogs
- Input file generation
- Launch preprocessor
- Output property parsing
- Out of presentation scope:
  - Custom GUIs to codes and chemistry domains not fitting the electronic structure model supported by the Calculation Editor
  - Custom Calculation Viewer property GUIs and visualization (currently being reworked for the wxWidgets Calculation Viewer)
  - Code output file importing
  - Job submission file generation (important, but too platform specific)

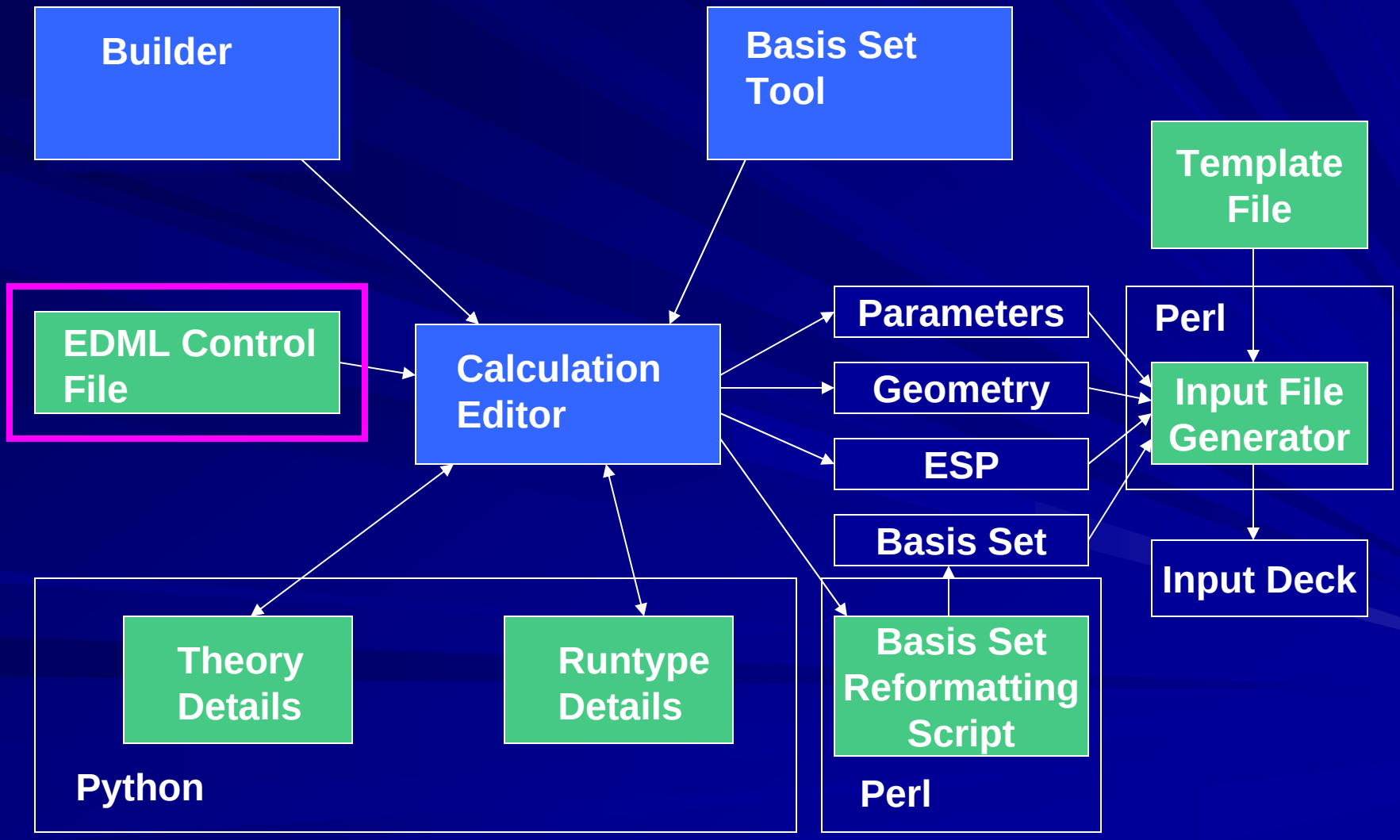
Calculation Editor and Details Dialogs



Code Registration: Calculation Setup



Calculation Setup: EDML File



EDML Control File

- ECCE Data Markup Language – XML based (sorry, we picked a lot of really lame file extensions that have stuck around over the years)
- Lives in $ECCE_HOME/data/client/cap
- IntegrationFiles block lists all primary code registration scripts (more lame extensions):

```
<IntegrationFiles>
        <InputGenerator>ai.nwchem</InputGenerator>
        <Template>nwch.tpl</Template>
        <BasisSetTranslationScript>std2NWChem</BasisSetTranslationScript>
        <LaunchPreprocessor>nwchem.launchpp</LaunchPreprocessor>
        <ParseSpecification>nwchem.desc</ParseSpecification>
        <Importer>NWChem.expt</Importer>
</IntegrationFiles>
```

EDML Control File (cont.)

- DataFiles block gives MIME content types for all code input/output files that will be stored on the Apache data (web) server:

```
<DataFiles>
        <Input type="primary" mimetype="chemical/x-nwchem-input" comment="true"
                commentstring="#">nwch.nw</Input>
        <Output type="primary" mimetype="chemical/x-nwchem-output">
                nwch.nwout</Output>
        <Output type="parse" verifypattern="%begin%input“
                mimetype="chemical/x-ecce-parse">ecce.out</Output>
        <Output type="auxiliary" mimetype="chemical/x-nwchem-mo">
                movec.nw_mo</Output>
        <Output type="property" mimetype="chemical/x-gaussian-cube">
                CUBE</Output>
        <Output type="property" mimetype="chemical/x-nwchem-md-trajectory">
                TRJ</Output>
        ...
</DataFiles>
```

EDML Control File (cont.)

- Editor block contains
  - GUI details dialog script names
  - Theory category/name blocks containing supported runtypes
  - Theory and runtype summary field blocks

```
<Editor theorydialog="nedtheory.py" runtypedialog="nedruntype.py">
        <Theory category="SCF" name="RHF">
                <runtype>Energy</runtype>
                <runtype>Gradient</runtype>
                <runtype>Geometry</runtype>
                <runtype>Vibration</runtype>
                <runtype>GeoVib</runtype>
                <runtype>Property</runtype>
                <runtype noSpherical = "true">ESP</runtype>
        </Theory>
        …
        <Theory category="DFT" name="RDFT">
                <runtype>Energy</runtype>
                <runtype>Gradient</runtype>
                <runtype>Geometry</runtype>
                <runtype>Vibration</runtype>
                <runtype>GeoVib</runtype>
                <runtype>Property</runtype>
        </Theory>
        …
```

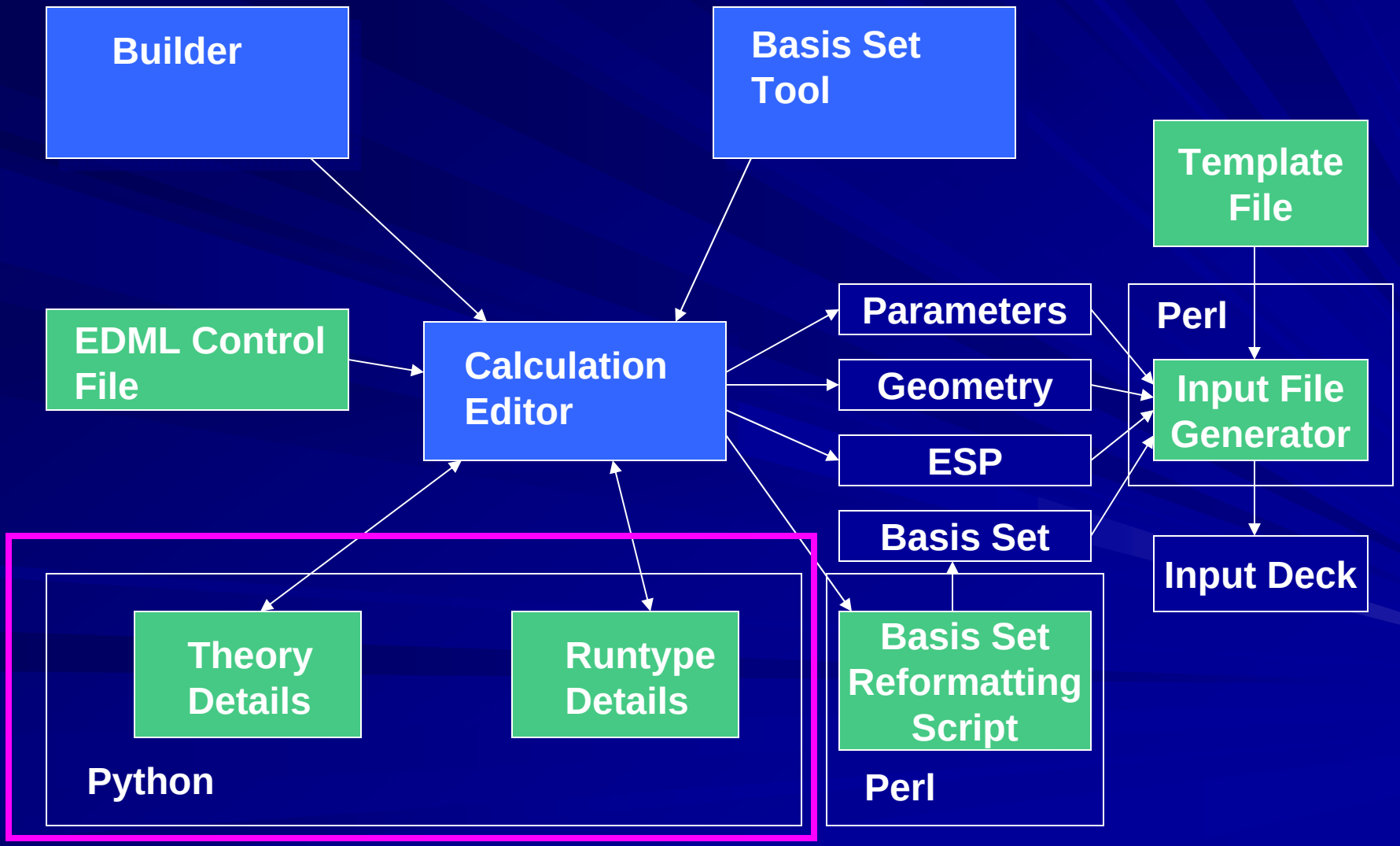
EDML Control File (cont.)

```
...
        <TheorySummary>
                <item key="ES.Theory.SCF.Direct" label="SCF Integrals"></item>
        </TheorySummary>
        <TheorySummary>
                <item key="ES.Theory.SCF.ConvergenceAlgorithm“
                        label="SCF Conv. Algorithm"></item>
        </TheorySummary>
        <TheorySummary topLabel="SCF Convergence-">
                <item key="ES.Theory.SCF.ConvergenceGradient.Value“
                        label="Gradient"></item>
                <item key="ES.Theory.SCF.ConvergenceDensity.Value"
                        label="Density"></item>
                <item key="ES.Theory.SCF.ConvergenceEnergy.Value"
                        label="Energy"></item>
        </TheorySummary>
        ...
        <RuntypeSummary>
                <item key="ES.Runtype.GeomOpt.SearchAlgorithm“ label="Algorithm"></item>
        </RuntypeSummary>
        <RuntypeSummary>
                <item key="ES.Runtype.GeomOpt.SearchFor" label="Search for"></item>
        </RuntypeSummary>
        ...
</Editor>
```

EDML Control File (cont.)

- GaussianBasisSetRules block for controlling Basis Set Tool behavior per code
- GeometryConstraintRules for controlling Calculation Editor Geometry Constraints Toolkit
- MOOrdering block for controlling Calculation Viewer MO computation

Calculation Setup: GUI Details Dialogs



GUI Details Dialogs

- wxPython based – de facto standard GUI toolkit for sophisticated python based GUI development
- Same underlying open source cross-platform C++ GUI toolkit, wxWidgets, used for all core ECCE applications
- ECCE GUI input field classes (widgets) inherit from standard wxPython classes and adds
  - **Streamlined development by combining multiple widgets**
  - **Numeric range validation**
  - **Warning/error messaging**
  - **Communication with Calculation Editor for saving/restoring values (ECCE normally saves key/value pairs when the user has overridden the default value)**
  - **Support for read-only invocations of dialogs prohibiting input field changes**
- Lives in $ECCE_HOME/codereg
- pydi test script to show details dialogs outside ECCE

GUI Details Dialogs (cont.)

- **Boilerplate details dialog code:**

```
from templates import *

class NedTheoryFrame(EcceFrame):
   def __init__(self, parent, title, app, helpURL=""):
      EcceFrame.__init__(self, parent, title)
      panel = NedTheoryPanel(self, helpURL)
      self.Finalize()

class NedTheoryPanel(EccePanel):
   def __init__(self, parent, helpURL):
      EccePanel.__init__(self, parent, helpURL)
      # All dialog input fields created here
      self.AddButtons()

   def CheckDependency(self):
      noop = 0
      # Dependency/constraint logic for dialog here

# main logic
frame = NedTheoryFrame(None,
                  title = "ECCE NWChem Editor: Theory Details",
                  app = app, helpURL = "")
```

GUI Details Dialogs (cont.)

- Input field classes:
  - EcceCheckBox: toggle for binary state input
  - EcceComboBox: drop-down menu for selecting one of several options
  - EcceSpinCtrl: type-in field for integer numbers with range validation plus up/down arrow keys
  - EcceFloatInput: type-in field for floating point numbers with range validation
  - EcceExpInput: Specialization of EcceFloatInput that requires exact powers of 10 to be input
  - EcceTextInput: type-in field for free-form input

GUI Details Dialogs (cont.)

- EcceCheckBox example:

```
self.symmetryTog = EcceCheckBox(self, label = “Use Available Symmetry”,
                                 name = “ES.Theory.UseSymmetry”,
                                 default = True)
```

- EcceComboBox example:

```
qualityChoice = [“Extra Coarse”, “Coarse”, “Medium”, “Fine”, “Extra Fine”]
self.quality = EcceComboBox(self, label = “Quality:”,
                            name = “ES.Theory.DFT.GridDensity”,
                            choices = qualityChoice, default = 2)
```

GUI Details Dialogs (cont.)

- EcceSpinCtrl example:

```
self.maxIter = EcceSpinCtrl(self, label = “Max. Iterations:”,
                            name = “ES.Theory.SCF.ConvergenceIterations”,
                            hardRange = “[0..)”, default = 20)
```

- EcceFloatInput example:

```
self.gradient = EcceFloatInput(self, label = “Gradient:”,
                               name = “ES.Theory.SCF.ConvergenceGradient.Value”,
                               hardRange = “[0..)”, softRange = “[1e-10..1e-2]”,
                               default = 1e-4, unit = “Hartree”)
```

GUI Details Dialogs (cont.)

- Dialog layout classes:
  - **EcceFrame: top-level window class, 1 instance per dialog**
  - **EccePanel: top-level container/sizer class, 1 instance per dialog**
  - **EcceLineSeparator: horizontal separator line**
  - **EcceLineLabelSeparator: horizontal separator line with embedded label**
  - **EcceBoxSizer: Grid layout of children using specified number of columns before creating a new row, and optionally**
    - **under a labeled separator line**
    - **all inside a framed/labeled box**
  - **EcceHBoxSizer / EcceVBoxSizer: horizontal or vertical layout of all children, no frame/label**
  - **EcceLineLabelHBoxSizer / EcceLineLabelVBoxSizer: horizontal or vertical layout of all children inside framed/labeled box**
  - **EcceTabPanel: tabbed page of notebook for maximizing dialog space usage**

GUI Details Dialogs (cont.)

- **Global variables available to dialogs (referenced as EcceGlobals.<variable>):**
  - **Category (theory category)**
  - **Theory (theory name)**
  - **RunType**
  - **SymmetryGroup**
  - **NumElectrons**
  - **SpinMultiplicity (1=singlet, 2=doublet, etc.)**
  - **NumFrozenOrbs (frozen core)**
  - **NumOccupiedOrbs**
  - **NumVirtualOrbs (excluded virtual)**
  - **NumNormalModes**

GUI Details Dialogs (cont.)

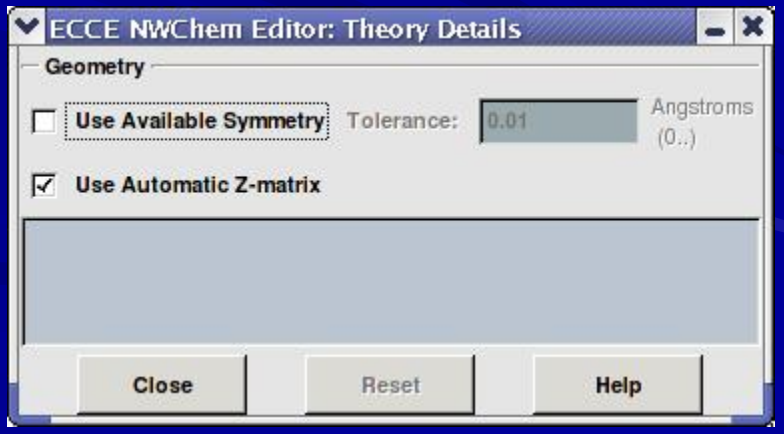
- EccePanel CheckDependency method:
  - **Single point in detail dialog to collect all constraint/dependency logic streamlines development**
    - **No need to define a custom event callback for each bit of constraint logic**
  - **Every change made to an input field value by the user triggers the CheckDependency method**
  - **Number of input fields on dialogs small enough that checking all dependencies for every input field change is not a performance issue**
  - **Commonly used for:**
    - **enabling/disabling input fields based on the current values of other input fields**
    - **Setting default input field values based on the current values of other input fields**
    - **Setting the list of possible choices for an EcceComboBox based on other input field values**
    - **Issuing warning messages that are dependent upon the values of multiple input fields**
  - **The wxPython Bind method can still be used when desired for associating a method with individual constraints**
  - **Previous ECCE details dialog toolkit implementation used more “object oriented” binding of input field changes to individual methods, but time has shown this to increase the level of effort required and complexity of dialog scripts**

GUI Details Dialogs (cont.)

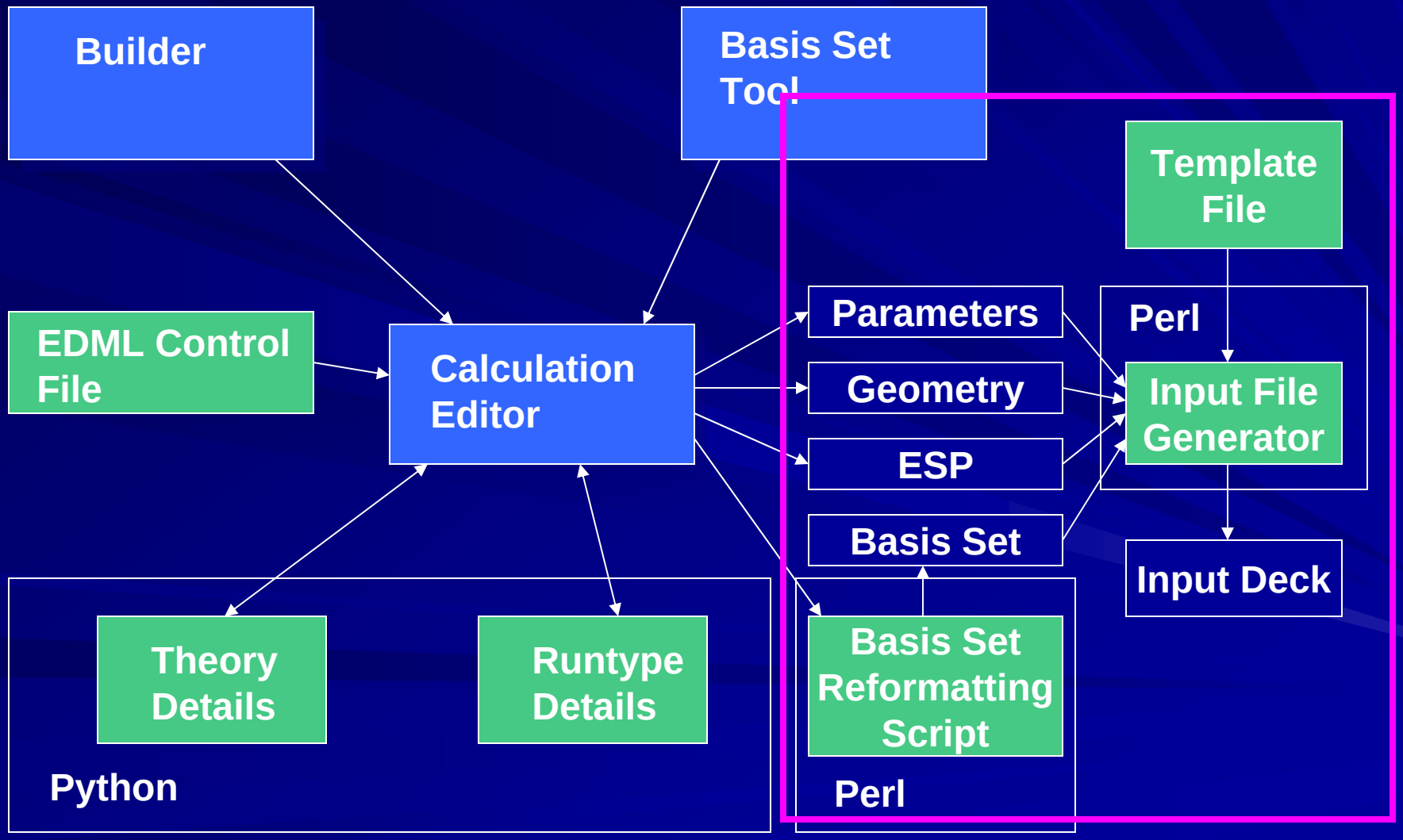
- Putting it all together:

```
from templates import *

class NedTheoryFrame(EcceFrame):
   def __init__(self, parent, title, app, helpURL=""):
      EcceFrame.__init__(self, parent, title)
      panel = NedTheoryPanel(self, helpURL)
      self.Finalize()

class NedTheoryPanel(EccePanel):
   def __init__(self, parent, helpURL):
      EccePanel.__init__(self, parent, helpURL)

      geometrySizer = EcceBoxSizer(self, "Geometry", 2)

      self.symmetryTog = EcceCheckBox(self, label = " Use Available Symmetry",
                                      name = "ES.Theory.UseSymmetry“, default = False)
      geometrySizer.AddWidget(self.symmetryTog)

      self.symmetryTol = EcceFloatInput(self, label = "Tolerance:“,
                                        name = "ES.Theory.SymmetryTol", default = 1e-2,
                                        hardRange = "(0..)", unit = "Angstroms“)
      geometrySizer.AddWidget(self.symmetryTol)

      self.useAutoZ = EcceCheckBox(self, label = " Use Automatic Z-matrix",
                                   name = "ES.Theory.UseAutoZ", default = True)
      geometrySizer.AddWidget(self.useAutoZ)
      self.panelSizer.Add(geometrySizer)

      self.AddButtons()
```

```
def CheckDependency(self):
      self.symmetryTol.Enable(self.symmetryTog.GetValue())

frame = NedTheoryFrame(None,
                       title = "ECCE NWChem Editor: Theory Details",
                       app = app,
                       helpURL =
      "http://www.emsl.pnl.gov/docs/nwchem/nwchem.html")
```



Calculation Setup: Input File Generation



Input File Generation

- Input file generation command:

  <input_generator> -n <base_file_name> -t <template_file> [-p] [-f] [-b] [-q] [-c]

- The file passed in as template_file must be overwritten with the generated input file (therefore it is a copy of the EDML registered template)
- The base_file_name is used to generate the names of the other files needed as input
- The input file generator command must be invoked in the same directory where the other files needed as input reside
- The other command line options are flags to indicate the existence of specific files:
  - "-p" ➔ <base_file_name>.param contains the details dialogs key/value pairs for both the theory and runtype dialogs
  - "-f" ➔ <base_file_name>.frag contains the chemical system in the ECCE standard MVM format
  - "-b" ➔ <base_file_name>.basis contains the basis set already converted to the format needed by the code
  - "-q" ➔ <base_file_name>.esp contains ESP constraints set in the Calculation Editor Partial Charge Editor
  - "-c" ➔ <base_file_name>.con contains geometry constraints set in the Calculation Editor Geometry Constraint Editor
- The .param file also contains special ECCE values that are needed for input file generation such as the calculation name, user annotation, theory name, runtype, number of electrons, charge, etc.
- Input file generation files live in $ECCE_HOME/scripts/parsers

Input File Generation (cont.)

- Basis set reformatting
  - Existing support is provided for the following codes:
    - NWChem
    - Gaussian 92, 94, 98, 03
    - GAMESS
    - GAMESS-UK
    - ACES II
    - AMICA
    - HONDO
    - MELDEF
    - MOLCAS
    - MOLPRO
    - Supermolecule
    - TX93
  - Typically the reformatting scripts for these codes can be reworked to add support for new codes by starting with the code closest in basis set format to the new code
  - Two Perl scripts are needed:
    - The one specified in the EDML file as the <BasisSetTranslationScript>, by convention named **std2<code>**
    - A script for writing out the basis set format needed by the code from the ECCE standard format, by convention named **wr<code>GBS.pm**
    - The std2<code> script is invoked by the Calculation Editor during input file generation
    - The std2<code> script invokes the wr<code>GBS.pm script
  - Check out the existing scripts such as std2NWChem and wrNWChemGBS.pm in $ECCE_HOME/scripts/parsers to learn more

Input File Generation (cont.)

- ECCE registered codes use a Perl script in combination with a template file to generate input files
- The developer may choose to do input file generation in another way (Python script, FORTRAN executable, no template file, etc.) provided it adheres to the input file generation command conventions listed
- Template file contains "##" delimited keywords that are placeholders for substituting in the input specified by the user for the given calculation
- The "##" keywords correspond to either
  - Python details dialog user input field "name" parameters (or a unique substring match)
  - Subroutine names in the input file generation script
- Python details dialog "name" parameter substitution works well for keyword driven input like NWChem, poorly for more obtuse formats, e.g. Gaussian
- Template file lines containing "##" keywords with neither a corresponding details dialog value (because it's not applicable to the theory/runtype or because the user left the value defaulted) nor an input file generator subroutine (or the subroutine returns an empty string) are removed from the generated input file
- Input file generator subroutine name substitution is needed for more complex input directives rather than a straight mapping of details dialog input fields to input file keywords (e.g., Gaussian route card)
- The subroutine can then take a combination of user input field values and coerce them in unnatural ways to come up with the required input file syntax
- The input file generator is also responsible for composing the other input it is given (fragment file, basis set file, etc.) into a properly formatted code input file

Input File Generation (cont.)

- Template file nwch.tpl snippet:

```
Title "##annotation##“              # ECCE user annotation or calculation name given in .param file

Start  ##title##                    # ECCE calculation name given in .param file

echo

Memory ##MemorySize## mw    # Details dialog ES.Theory.SCF.MemorySize

##Charge##                          # Input generator script subroutine

##chemsys##                         # Input generator script subroutine to write the chemical system

ecce_print ##parseFile##           # ECCE registered parse file name given in .param file

##basis##                           # Input generator script subroutine to write the basis set

scf
  ##SCFTheory##                     # Input generator script subroutine to format theory category/name
  ##SCFDirect##                     # Input generator script subroutine, see next page
  thresh ##SCF.ConvergenceGradient.Value##      # Details dialog ES.Theory.SCF.ConvergenceGradient.Value
  maxiter ##SCF.ConvergenceIterations##         # Details dialog ES.Theory.SCF.ConvergenceIterations
  ##SCFLevel##                                  # Input generator script subroutine,  see next page
end
```

Input File Generation (cont.)

- Input generator script ai.nwchem snippet:

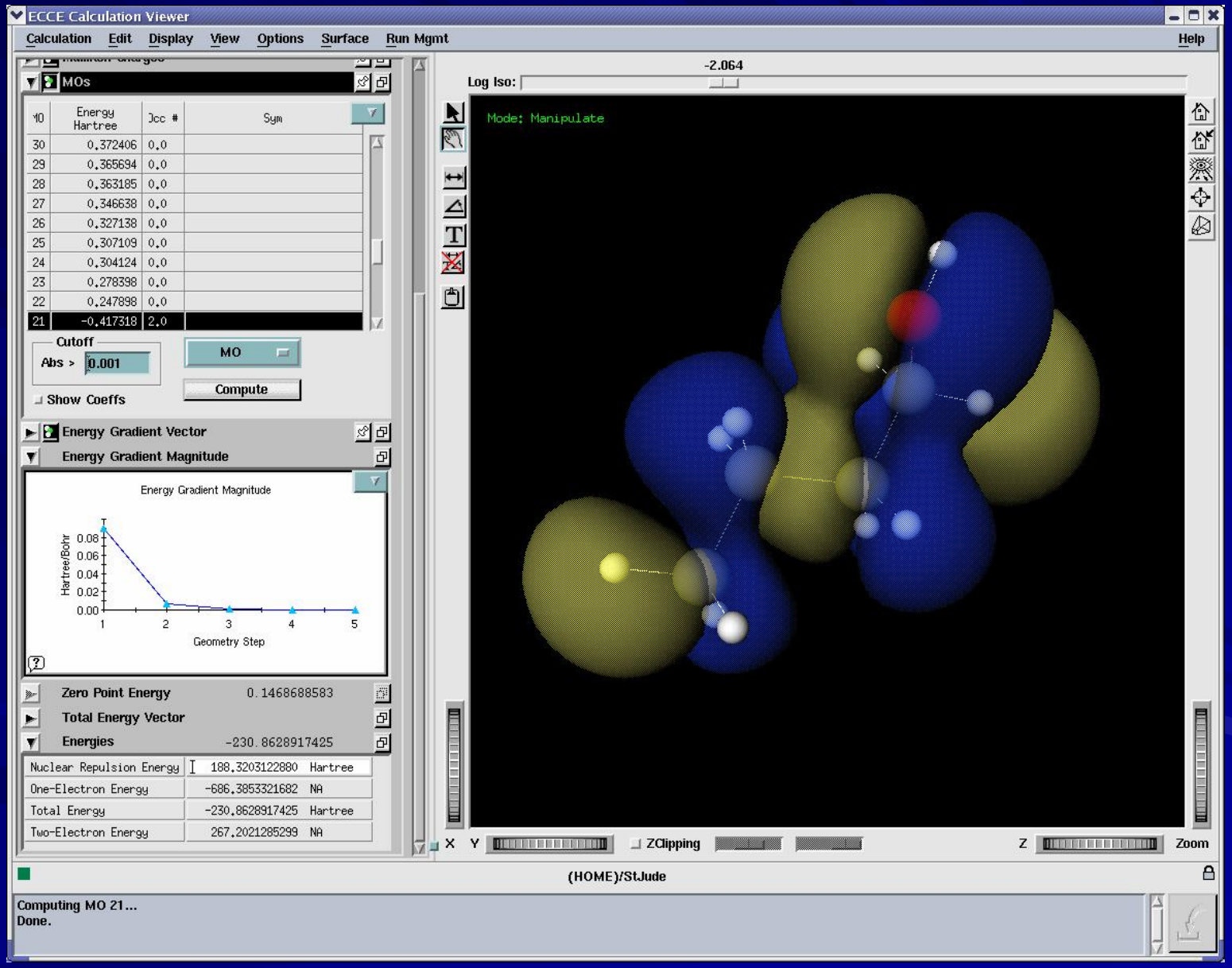
```
#!/usr/bin/env perl

sub SCFDirect {
  local($result);
  $result = $AbiDict{"ES.Theory.SCF.Direct"};

  if ($result eq "" &&
      $AbiDict{"SCF.DiskSize"}) {
    $result = "semidirect filesize ".
              $AbiDict{"SCF.DiskSize"}."000000";
  } elsif ($result eq "Direct") {
    $result = "direct";
  }
  return $result;
}
```

```
sub SCFLevel {
  local($shift1, $shift2, $crossover, $result);
  $shift1 = $AbiDict{"LevelShiftSize"};
  $shift2 = $AbiDict{"NewLevelShiftSize"};
  $crossover =
     $AbiDict{"NewLevelShiftCrossover"};

  if ($shift1) {
    if ($crossover && (! $shift2 )) {$shift2 = 0.0 }
    $result = "level pcg $shift1 $crossover $shift2";
  } else {
    $result = "";
  }
  $_ = $result;
  return $result;
}
```

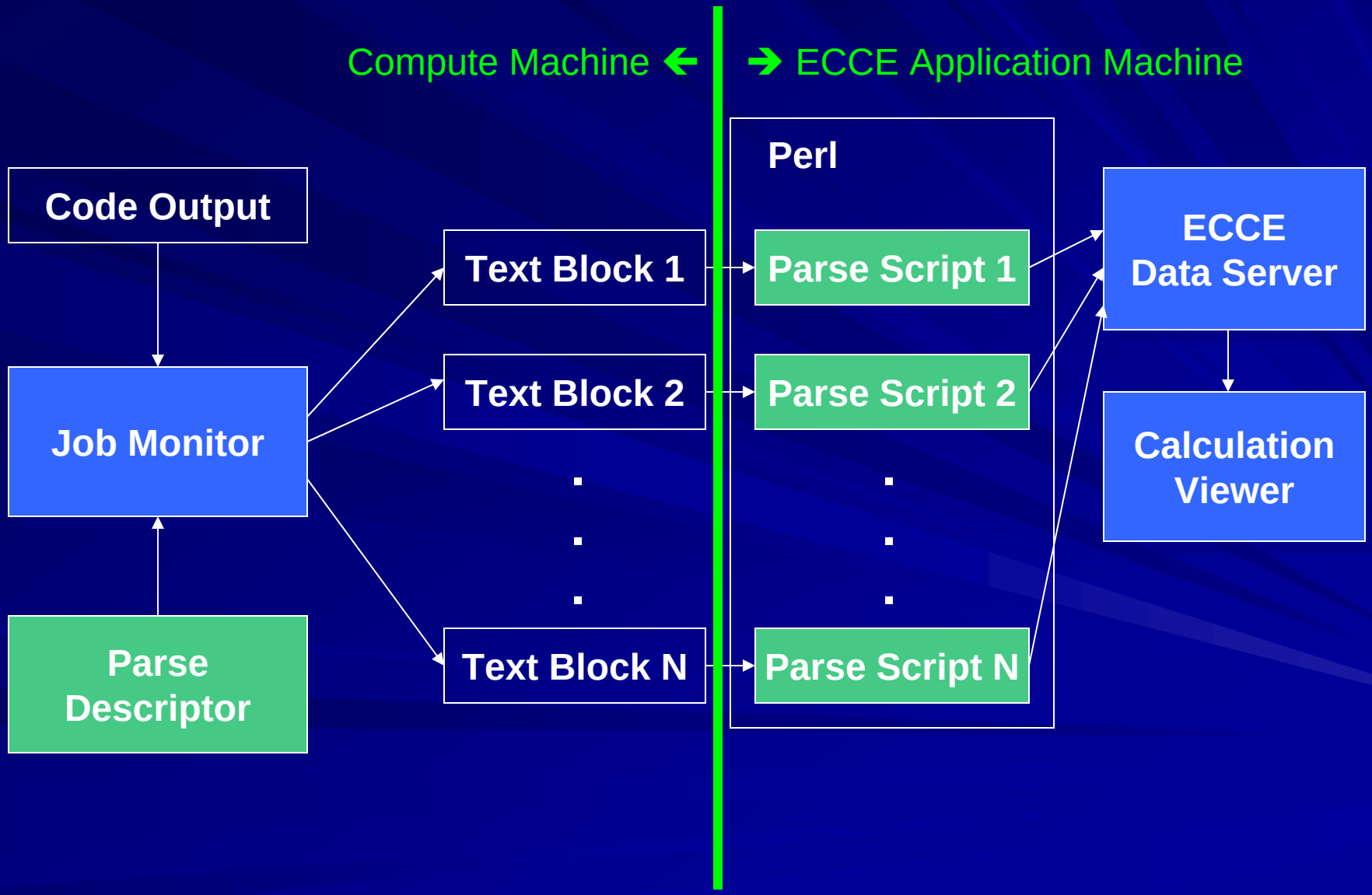
Launch Preprocessor

- Augments the code input file with any directives not known before the job is launched
- Script is invoked with “-p” option specifying the name of a kev/value parameter file
- Parameter file contains Job Launcher GUI settings including run directory, scratch directory, number of nodes and processors selected, etc.
- Currently, NWChem launch preprocessor script only adds the scratch directory to the input file
- Some codes don’t properly divide chemistry calculation input from job submission input making the launch preprocessor critical
- Lives in $ECCE_HOME/scripts/parsers

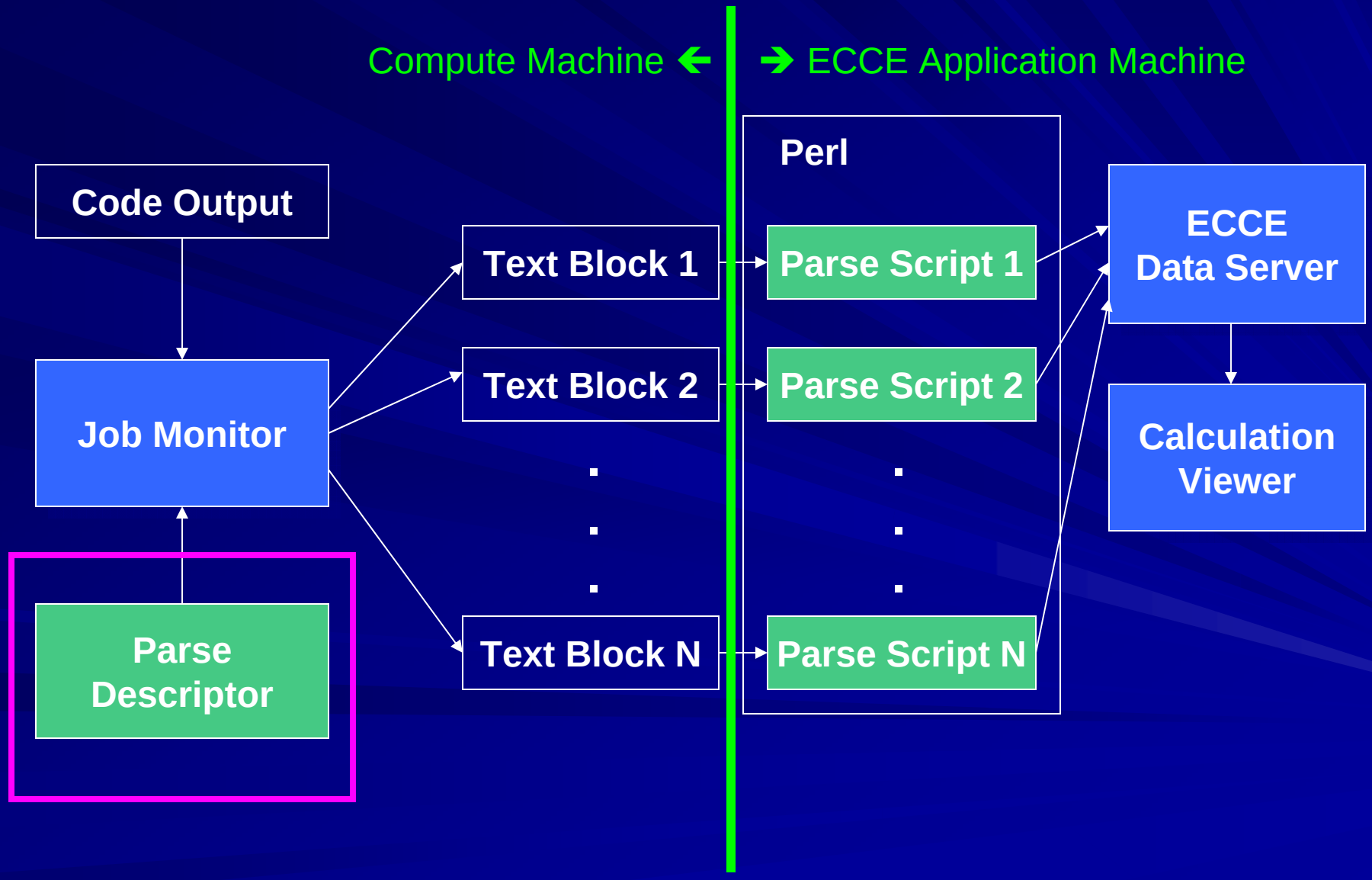
Calculation Viewer



Code Registration: Output Parsing



Output Parsing: Parse Descriptor



Parse Descriptor

- During job execution, eccejobmonitor perl script scans the output file for matches on strings specified by parse descriptor file
- When a parse descriptor is matched, buffer data until the end marker of the descriptor is found
- Data block is shipped via ssh from the compute machine to the eccejobstore process on the ECCE application machine
- Parse descriptor file lives in $ECCE_HOME/scripts/parsers

To be continued…

Parse Descriptor (cont.)

- Parse Descriptor nwchem.desc snippet:

```
[TEVEC]
Script=nwchem.te
Begin=task_gradient%begin%total energy
Frequency=all
End=task
[END]

[DIPOLE]
Script=nwchem.dipole
Begin=begin%total dipole
Frequency=all
Lines=2
[END]

[ESPCHARGE]
Script=nwchem.esp
File=##CalcName##.q
Begin=##CalcName##.q
[END]
```

Properties File

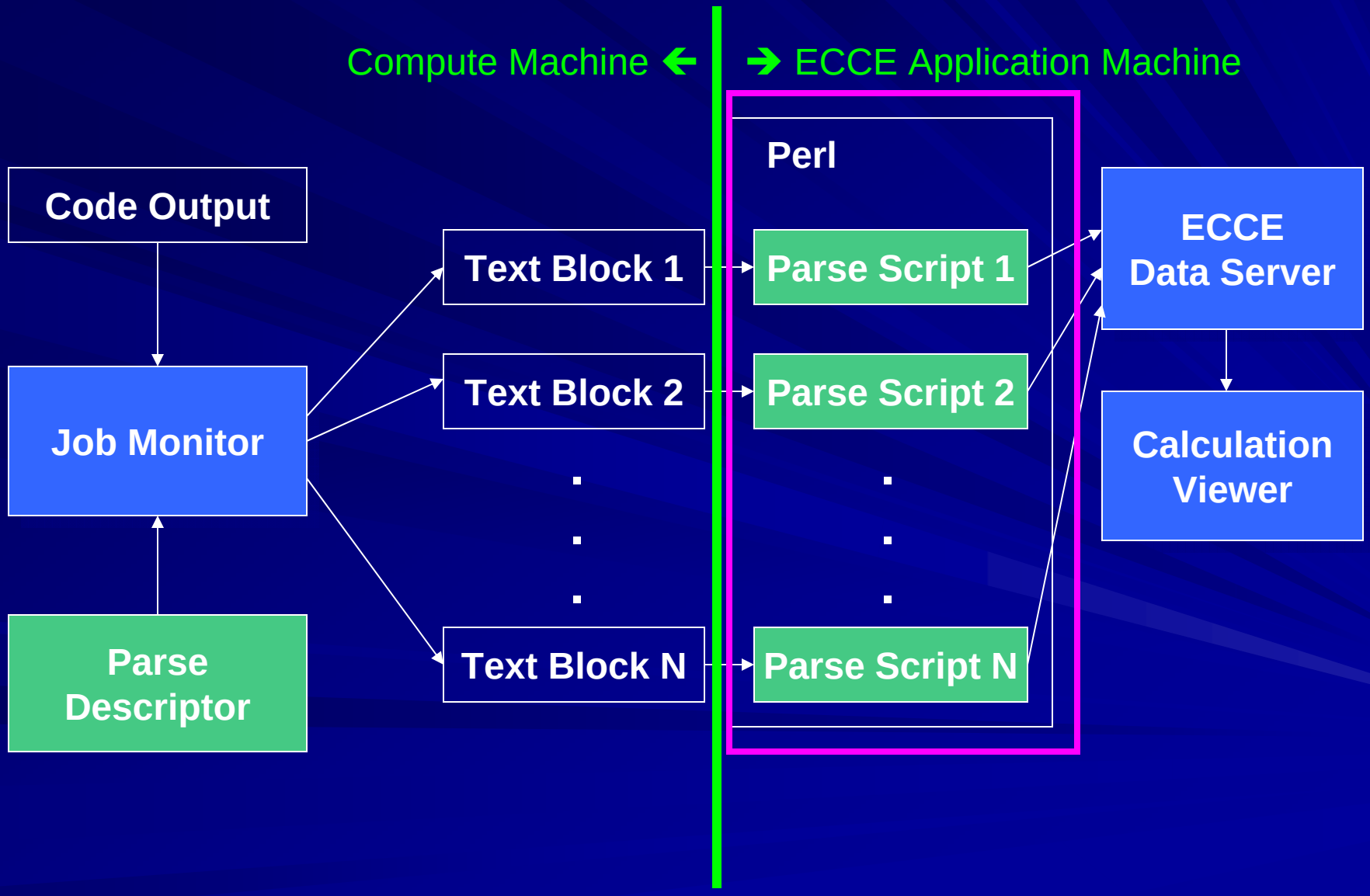
- Maps property names (keys) to data representation based on dimension and type of data per dimension
- Calculation Viewer uses the data representation attribute to determine the property panel GUI and whether/how to visualize, graph, or display raw data for the property
- Descriptive label and units are also properties file attributes used by the Calculation Viewer property panel GUIs
- Current ~250 properties are listed – best strategy is to find one close to the new one
- Lives in $ECCE_HOME/data/client/config

Properties File (cont.)

- properties file snippet:

#class	user_label	prop_key	units	representation
######	##########	########	#####	###############
#				
Energy	Zero Point Energy	EZEROPT	Energy	PropValue
Energy	Total Energy Vector	TEVEC	Energy	PropTSVector<Geometry Step>
Energy	Kinetic Energy Vector	KEVEC	Energy	PropTSVector<Geometry Step>
Energy	Potential Energy Vector	PEVEC	Energy	PropTSVector<Geometry Step>
Energy	SCF Energy Vector	ESCFVEC	Energy	PropTSVector<Geometry Step>
Energy	DFT Energy Vector	EDFTVEC	Energy	PropTSVector<Geometry Step>
Energy	MP2 Energy Vector	EMP2VEC	Energy	PropTSVector<Geometry Step>
#				
Charge	Mulliken Charges	MULLIKEN	Charge	PropVector<Atom>
Charge	Mulliken Shell Charges	MLKNSHELL	Charge	PropVector<Unknown>
Charge	Electron Density at Nuclei	EDENS	EFieldGrad	PropVector<Atom>
Charge	Fermi Contact	FERMI	MHertz	PropVector<Atom>
Charge	ESP Charge Fitting	ESPCHARGE	Charge	PropTable<Atom,Charge>
#				
Derived	S**2	S2	NA	PropValue
Derived	Dipole Moment	DIPOLE	Dipole	PropVector<Coordinate>
Derived	MP2 Dipole Moment	MP2DIPOLE	Dipole	PropVector<Coordinate>
Derived	Quadrupole Moment	QUADPOLE	Quadrupole	PropVector<Unknown>

Output Parsing: Parse Scripts



Parse Scripts

…Output parsing continued

- eccejobstore process invokes the proper parse script with command line arguments on the data block
- Parse script reformats the data based on its properties file data representation
- eccejobstore uploads reformatted property data to ECCE data server
- eccejobstore sends out JMS message notifying interested applications of the new property
- Calculation Viewer(s) in context of calculation process JMS message, retrieve, and display/visualize property
- Parse scripts live in $ECCE_HOME/scripts/parsers

Parse Scripts (cont.)

- Parse script invocation command:

  <parse_script> <key> <runtype> <theory_category> <theory_name> <open_shells> **>**parseOutFile **<**parseInFile

- Note that the parseInFile and parseOutFile are specified with file redirection and parse scripts read from stdin and write to stdout

- By convention, <key> is "." as a placeholder and parse scripts know or determine the key they need to use for the parseOutFile (it is left for "historical reasons")

- Parse scripts typically load only the command line arguments needed and most need none.

Parse Scripts (cont.)

- The parse output file format is tied to the data representation with four basic output formats:
  - Scalar data
    key:
    size:
    values:
    END
  - Table data
    key:
    size:
    rowLabels:
    columnLabels:
    values:
    END
  - Vector data
    key:
    size:
    columnLabels:
    values:
    END
  - Vector of Tables data
    key:
    size:
    vectorLabels:
    rowLabels:
    columnLabels:
    values:
    END

Parse Scripts (cont.)

- Parse script nwchem.dipole:
  - **Need to reformat this data block extracted using the parse descriptor file from the ecce.out file:**

task_optimize driver task_gradient scf%begin%total dipole%3%double
4.38894415454724e-01 3.91321140865017e-01 -4.38827528892410e-01

  - **into this format specified by the properties file:**

key: DIPOLE
size:
3
rowLabels:
x y z
values:
1.02648373484354 1.1658016598219 -1.07813172811085
units:
Debye
END

Parse Scripts (cont.)

- Parse script nwchem.dipole:

```
#!/usr/bin/env perl

# Force output to be flushed
$| = 1;

$label = <STDIN>;
chop($label);
$line = <STDIN>;
chomp($line);
@values = split(/ +/,$line);
#########################################################
#  convert atomic units to debye
#########################################################
$size = @values;
for ($i = 0; $i < $size; $i++) {
  $values[$i] = 2.541766*$values[$i];
}
#########################################################
# for mp2, both the mp2 and scf dipoles are output
#########################################################
$label =~ /(\S+)%begin/;
$prebegin = $1;  # might be task_<tasktype> or a theory type

if ($prebegin =~ /mp2/) {
  $key = "MP2DIPOLE";
} else {
  $key = "DIPOLE";
}
```

```
 print "key: $key\n";
print "size:\n3\n";
print "rowlabels:\n";
print "x y z\n";
print "values:\n";
foreach $i (@values) { print " $i"; }
print "\nunits:\nDebye\n";
print "\nEND\n";
```

Parse Scripts (cont.)

- Parse script nwchem.esp:
  - **Need to reformat this <CalcName>.q file specified by the parse descriptor file:**

```
 3  3
O   0.000000   0.000000  -0.012125  -0.824086  -0.823655  -1.638648
H  -0.078299   0.000000   0.048499   0.412455   0.412240   0.000000
H   0.078299   0.000000   0.048499   0.411631   0.411415  -1.618905
```

  - **into this format specified by the properties file:**

```
key: ESPCHARGE
size:
3 3
rowLabels:
1 2 3
columnLabels:
ESP RESP CRESP2
values:
-8.2408600000000000e-01 -8.2365500000000000e-01 -1.6386480000000000e+00 4.1245500000000000e-01
    4.1224000000000000e-01 0.0000000000000000e+00 4.1163100000000000e-01 4.1141500000000000e-01 -
    1.6189050000000000e+00
units:
e
END
```

Parse Scripts (cont.)

- Parse script nwchem.esp:

```
#!/usr/bin/env perl

# Force output to be flushed
$| = 1;

# read the data from stdin:
#
$line = <STDIN>;
$line =~ s/^\s*//;
$line =~ s/\s*$//;
($natom,$ncol) = split(/ +/,$line);
$icnt = 0;
@charges = ();
 while (<STDIN>) {
   last if ($icnt >= $natom);
   $line = $_;
   $line =~ s/^\s*//;
   $line =~ s/\s*$//;
   @values = ();
   @values = split(/ +/,$line);
   for ($i = 4; $i<4+$ncol; $i++) {
     push(@charges, $values[$i]);
   }
   $icnt++;
}
#
# Print out the data in standard format.
#
print "key: ESPCHARGE\n";
print "size:\n";
print $natom . " $ncol\n";
print "rowlabels:\n";

for ($i=1;$i<=$natom;$i++) {
  print "$i ";
  if ($i % 12 == 0 &&  $i < $natom - 1) { print "\n";}
}
if ($ncol == 5) {
  print "\ncolumnlabels:\nESP CESP RESP CRESP CRESP2\n";
} elsif ($ncol == 4) {
  print "\ncolumnlabels:\nESP CESP RESP CRESP\n";
} elsif ($ncol == 3) {
  print "\ncolumnlabels:\nESP RESP RESP2\n";
} elsif ($ncol == 2) {
  print "\ncolumnlabels:\nESP CESP\n";
} elsif ($ncol == 1) {
  print "\ncolumnlabels:\nESP\n";
}
print "values:";
$icnt = 0;
for ($i=0; $i<= $#charges; $i++) {
  if ($icnt % $ncol == 0) {print "\n";}
  printf("%.15e ",$charges[$icnt]);
  $icnt++;
}
print "\nunits:\ne\n";
print "END\n";
```