Webinar: Migration of acsI X Model Code to Magnolia

SOT BMSS

January 18, 2019
Webinar Outline

• Overview of Magnolia UI and workflow
• Migration of model (CSL) code
• Summary of command scripting language (CMD) differences between acslX and Magnolia
• Advanced scripting using Python
What’s Magnolia?

- Magnolia is an environment for modeling systems whose behavior can be described by systems of differential equations. Magnolia provides the tools for developing models using an equation-based modeling language, scripting the execution of simulations using either the Python programming language or a simple command-based language, and for interactively exploring model behavior using an intuitive user interface.

- Modeling language heavily influenced by ACSL/acsIx

- [http://www.magnoliasci.com](http://www.magnoliasci.com), team@magnoliasci.com
Overview of Magnolia UI and workflow

• Workspaces/projects are no longer used; all files are organized using filesystem folders according to user’s preferences
• Models are translated and compiled silently and automatically as needed (this usually takes only a fraction of a second, even for large models)
• Running a loaded model automatically creates a default plot from which the model behavior may be interactively explored
• A command prompt is provided, but uses the Python language instead of the M language
• Multiple models may be loaded and active simultaneously
• Scripts may be developed using either the CMD language or the Python language
Magnolia UI and Workflow Demo
ACSL/acslX Model Code Migration
Why change the language?

• Simplify language by unifying syntax constructs which all do the same thing
  • E.g., code block delimiters: “end” vs “end if” versus “label:continue”

• Remove legacy syntax constructs which would interfere with planned future language extensions
  • E.g., planned object-orientation makes use of “dot” token in things like “.AT.” and “.GT.” problematic

• Make the language more consistent with modern, popular programming languages
  • MATLAB, C/C++, Java, Python

• Replace syntax constructs which could lead to ambiguities
  • E.g., function call versus array notation
Replace the PROGRAM keyword

Old Syntax
program This is a model

New Syntax
model butadiene

This needs to be a valid variable name
Replace old-style comment delimiters

**Old Syntax**

‘This is a comment’

$This is a comment

**New Syntax**

! This is a comment

$This is a comment
Combine datatype declarations and DIMENSION statements

• Datatype specification (REAL, INTEGER, LOGICAL, CHARACTER) is now included in the DIMENSION statement and array declaration uses square brackets. CHARACTER variables no longer have to be assigned a size.

**Old Syntax**

```plaintext
dimension x(2, 3)
dimension flag(4)
    logical flag
character*25 title
```

**New Syntax**

```plaintext
dimension x[2, 3]
dimension logical flag(4)
    character title
dimension Exponential m1
```
Update array declarations and references

- Array elements are now addressed using square bracket notation.

**Old Syntax**

`dimension x(2, 3)`

`x(i, j) = 10.0*k(i)*y(j)`

**New Syntax**

`dimension x[2, 3]`

`X[i, j] = 10.0*k[i]*y[j]`
Update TABLE statements

- The data used to populate a TABLE statement is now specified in two or more CONSTANT arrays.

**Old Syntax**
```
table Bodyweight, 1, 10 /0, 1, ... 15, 20/
```

**New Syntax**
```
constant ageData = 0, 1, 2...
constant bwData = 0.1, 0.13...
table Bodyweight = ageData, bwData
```
Update SCHEDULE statements

• The AT/XZ/XP/XN qualifiers are no longer delimited by periods.

**Old Syntax**

```
schedule on .at. t + 0.5
schedule Bounce .xn. x
```

**New Syntax**

```
schedule on at t + 0.5
schedule Bounce xn x
```
Migrate old-style DO loops

- DO loops are replaced with FOR loops, which are delimited by an END statement, not a CONTINUE statement.

**Old Syntax**

```plaintext
do 10 i = 1, 2
    do 20 j = 1, 3
        dx(i, j) = k(i, j)*x(i, j)
    20: continue
10: continue
```

**New Syntax**

```plaintext
for i = 1, 2
    for j = 1, 3
        dx[i, j] = -k[i, j]*x[i, j]
    end
end
```
Migrate IF/THEN/ELSE statements

- Conditionals no longer use the THEN keyword, and the end block is delimited by END, not ENDIF.

**Old Syntax**

```plaintext
if (t .LT. 1.0) then
   x = t^2
else if ( t .LT. 1.35) then
   x = exp(t)
else
   x = sqrt(t)
end if
```

**New Syntax**

```plaintext
if (t < 1.0)
   x = t^2
else if ( t < 1.35)
   x = exp(t)
else
   x = sqrt(t)
end
```
Update relational and logical Operators

• FORTRAN-style relational and logical operators have been replaced by the C- and Java-style counterparts: ==, <, >, <=, >=, &,&, |. The only exception is the logical negation operator, which uses the MATLAB-style ~ instead of ! to avoid conflicts with CSL comments.

**Old Syntax**

```
termt(t .GE. tstop)
if(a .LT. 0.0 .OR. a .GT. 1.0)
```

**New Syntax**

```
termt(t >= tstop)
if(a < 0.0 || a > 1.0)
```
Remove (comment) ALGORITHM

• Presently, the default ODE solver in Magnolia is a fast solver which is appropriate for stiff systems
• Other solvers may be added in the future, at which point the ALGORITHM statement will be reintroduced

! algorithm ialg = 2
Example Ported CSL Model
Command Scripting Language (CMD)
General Differences

• Command abbreviation not presently supported
  • E.g., “d” cannot be used instead of “display”

• Command flags are now denoted with an “@” symbol instead of a “/”
  • E.g., “display @all” instead of “display /all”

• CMD scripts are now compiled instead of interpreted
  • Python is now used as the interactive scripting language

• Commands have been extended/added for parameter estimation, Monte Carlo analysis, and MCMC

• The “start” command no longer requires the /nocallback flag
Commands for which syntax has not changed

• DISPLAY
• OUTPUT
• PREPARE
• PROCEDURE
• PRINT
DATA

• Tabular data is no longer specified inline using the DATA command. Instead, data is imported from a CSV file and the DATA command is used to map columns to model variables.

• The DATA command is also used to assign descriptor variable values for use in parameter estimation.

• Example

  data @file='butadiene.csv' ds1e1 t='T' c_exh='S1E1' ... 
    tstop=60 BDW=86.2600 height=1.7400 Sex=1 Age=28
FIT

• The FIT command is used to execute parameter estimation runs from within a CMD file

• Used in conjunction with DATA and SET commands

• Example

```plaintext
data @file='observed.csv' dataset1 t='T' x='X' tstop=60 c=12.0
set k1 = 0.2 @min=0.1 @max=0.35
set k2 = 0.4 @min=0.23 @max=0.45
set k3 = 0.05 @min=0.03 @max=0.09
fit k1 k2 k3
```
LOAD

• Loads a model for use with the enclosing CMD script
• Must use the exact file name of the corresponding CSL file (case-sensitive)
• Example

  \texttt{load \textasciitilde{}butadiene.csl}
PLOT

• Used to create plots, similar to legacy syntax
• Additional syntax for creating “interactive” plots
• Additional syntax for plotting observed data specified in DATA statement
• Examples

```
plot @interactive @xvar=vtheta vthetadot rdot r theta ...
   @param=k @param=l
plot x 'obsdata:xobs'
```
SET

• Used to set model constants, similar to legacy syntax
• Additional syntax for assigning upper and lower limits to constants for use in parameter estimation
• Additional syntax for assigning distributions to constants for use in Monte Carlo analyses
• Examples

```
set k1 = 0.0  k2 = 0.3  k3 = 1.5
set pc_pp=1.4  @dist=norm  @mean=1.4  @std=0.3
```
• Used to start a simulation run, similar to legacy syntax
• Additional syntax for performing Monte Carlo iterations: @hold, @nruns
• Example

```plaintext
prepare t c_ven
set sc_vmax=0.0026 @dist=norm @mean=0.0026 @std=0.0008
start @hold @nruns=20
print @file='temp.csv' t c_ven
```
Example Magnolia-Compatible CMD Script
Advanced Scripting using Python
General Comments

• Python is a full-featured general programming language with a huge number of libraries. The Python website (python.org) is a good starting point to find language and library documentation, tutorials, examples, videos, etc.

• A few specific scripting tasks are presented here for guidance:
  • Loading and running a model
  • Setting the values of model constants
  • Getting the value of model variables or time histories following a run
  • Loading tabulated data from a file
  • Plotting from Python script code within Magnolia
Loading and running a model

• Models are represented as Python objects. Magnolia allows multiple models to be open and loaded into memory simultaneously. Use the models.get() method to assign a reference to an open model to a Python variable.

• Use the run() function of the model object to run a simulation

• Example

```python
mdl = models.get("butadiene.csl")
mdl.run()
```
Setting model constants

• Model constants (and variables) are referenced as Python object fields in using the “dot” notation

• Examples

\[
\begin{align*}
\text{mdl.xic} &= 0.0 \\
\text{mdl.k[0][1]} &= 3.0
\end{align*}
\]
Getting model variables and time histories

• Again, use the “dot” notation to reference model variables
• Use the `prepae()` function of the model object to add a variable to the prepare list
• Time histories are accessed using the `history()` function of the model object
• Examples

```python
mdl.prepare("x")
mdl.run()
yval = mdl.y
xvals = mdl.history("x")
```
Loading tabulated data

• Tabulated data can be loaded from a CSV file using the Python “csv” library

• Example

```python
csvreader = csv.reader(open('obsdata.csv'),
                       delimiter=',')
next(csvreader)  # Skip header row
for row in csvreader:
    mdl.x = row[0]  # Col 1 value
    mdl.y = row[1]  # Col 2 value
    # Etc...
```
Plotting from Python code

- Magnolia provides multiple methods for generating plots from Python using the “plot” Python object interface
- The handle-based plotting methods are similar to those used in acslX M language scripts
- Example

```python
import plot
import numpy as np

a = np.array(range(0, 100))/10.0
b = np.sin(a)
c = np.cos(a)

h = plot.line(a, b, ".R")
plot.append(h, a, c, ".B")

names = ["sine", "cosine"]
plot.legend(h, names)
plot.title(h, "Sin and Cos")
plot.xlabel(h, "Theta (radians)")
plot.ylabel(h, "Sin/Cos")

plot.xmin(h, -1.0)
plot.xmax(h, 11.0)
plot.ymin(h, -2.0)
plot.ymax(h, 2.0)
```
Example Magnolia-Compatible
Python Script