# *RAVEN interaction with External Applications*

RAVEN Workshop

www.inl.gov

Idaho National Laboratory

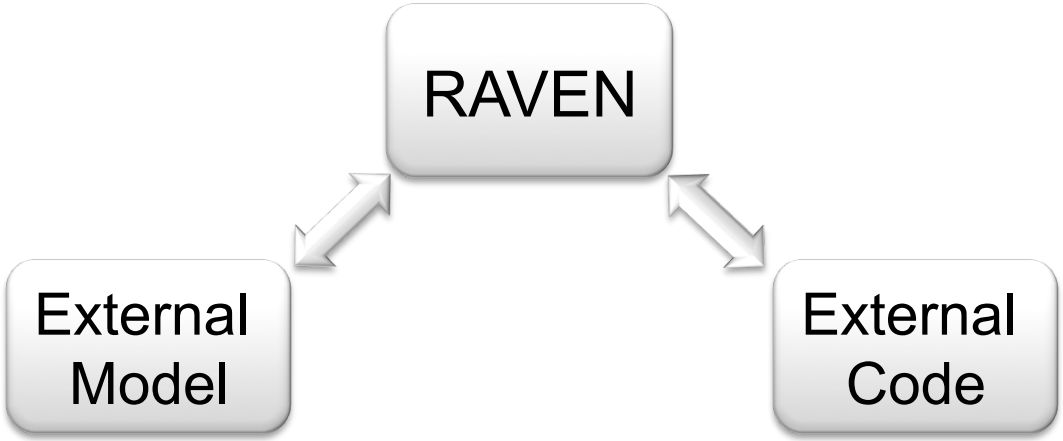
# Outline

- Overview of RAVEN interaction with external Applications
  - Available APIs: External Model and Code APIs
- Coupling a new Application through a Code Interface
  - Introduction
  - Code requirements
  - Interfaces that need to be implemented
  - Interaction with RAVEN
- Practical example of coupling a new code
  - Code overview
  - Creation of the input parser (coding)
  - Creation of the output parser (coding)
  - Execution of the interface (terminal)

# *Overview*

# *RAVEN Interaction with External Applications*

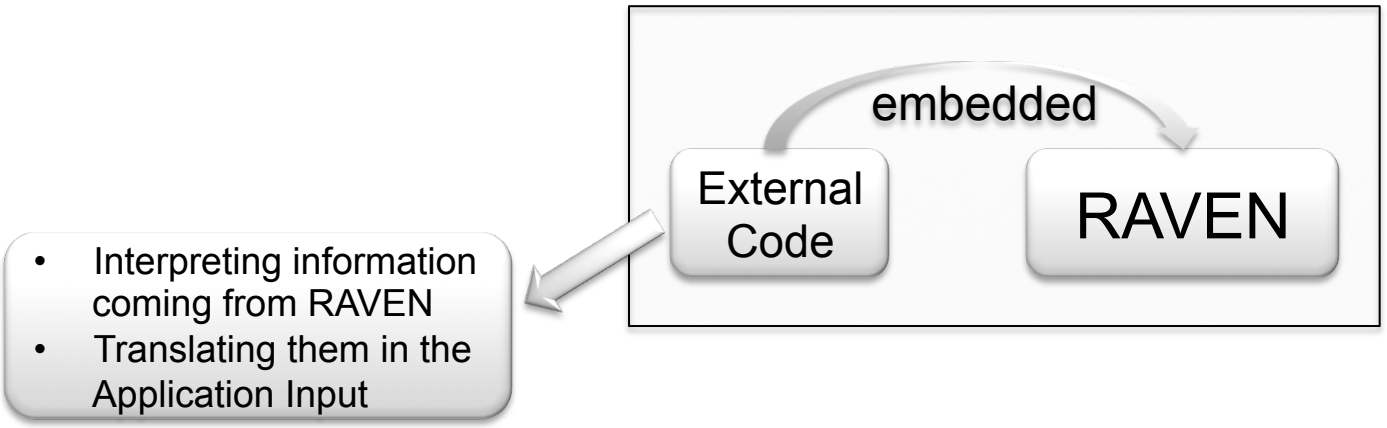
- RAVEN has two preferential APIs to interact with external Applications
  - *External Model*: An external Python "entity" that can act as a system model
  - *External Code*: API to drive external codes
- Both APIs are written in `PYTHON`



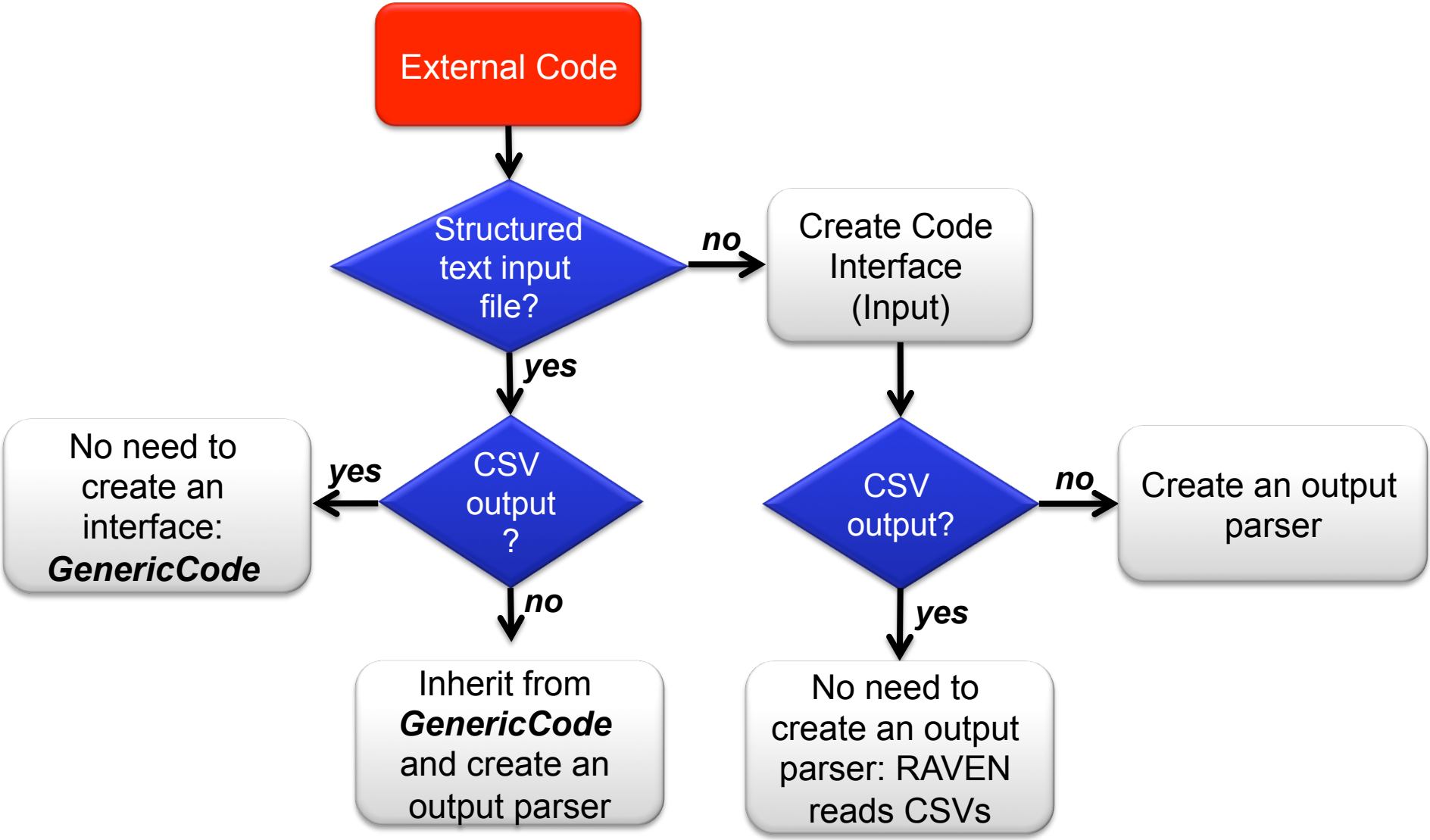
# *Coupling an Application through a code interface*

# Coupling an Application with RAVEN: Introduction

- The procedure of coupling a new Application with RAVEN is a straightforward process
- The coupling is performed through a `Python` Interface
- The Interface has two functions:
  1. Interpret the information coming from RAVEN
  2. Translate such information in the input of the driven code
- The coupling procedure does not require any modification of RAVEN

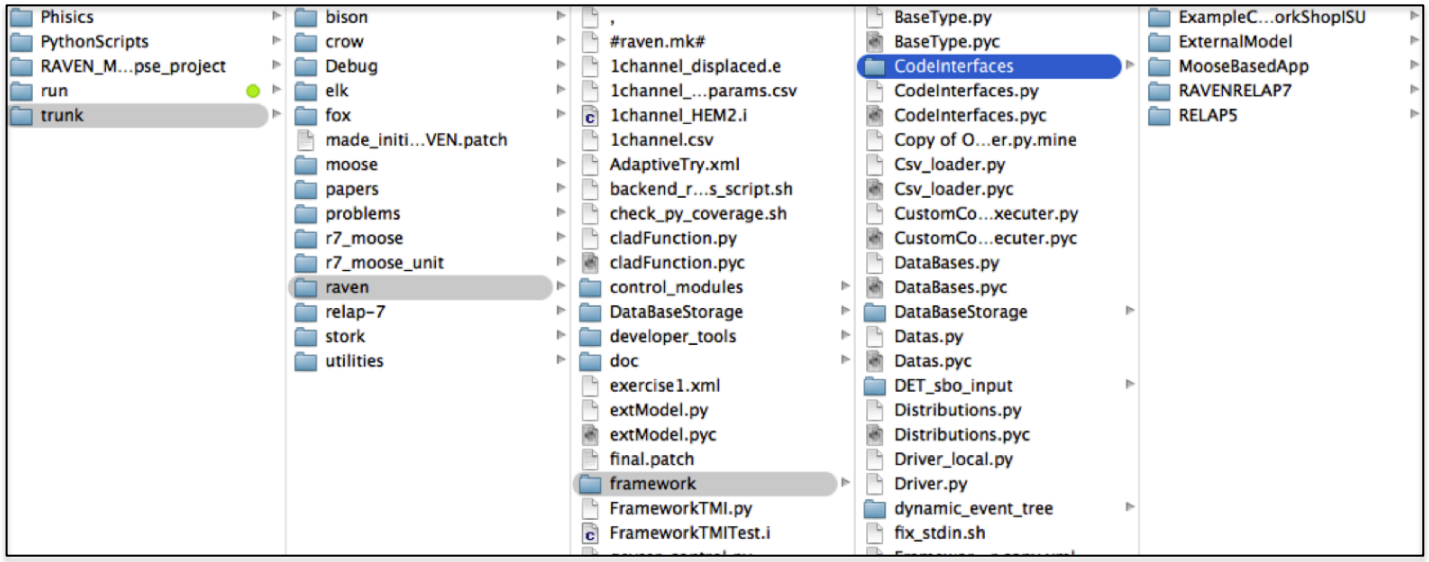


# Choose how to couple your Application to RAVEN



# Coupling an Application with RAVEN: Interfaces

- RAVEN becomes aware of the codes it can use as Models only at run-time
  - RAVEN looks for code interfaces and loads them automatically
- The code interface needs to be placed in a new folder under the directory "*./raven/framework/CodeInterfaces*"



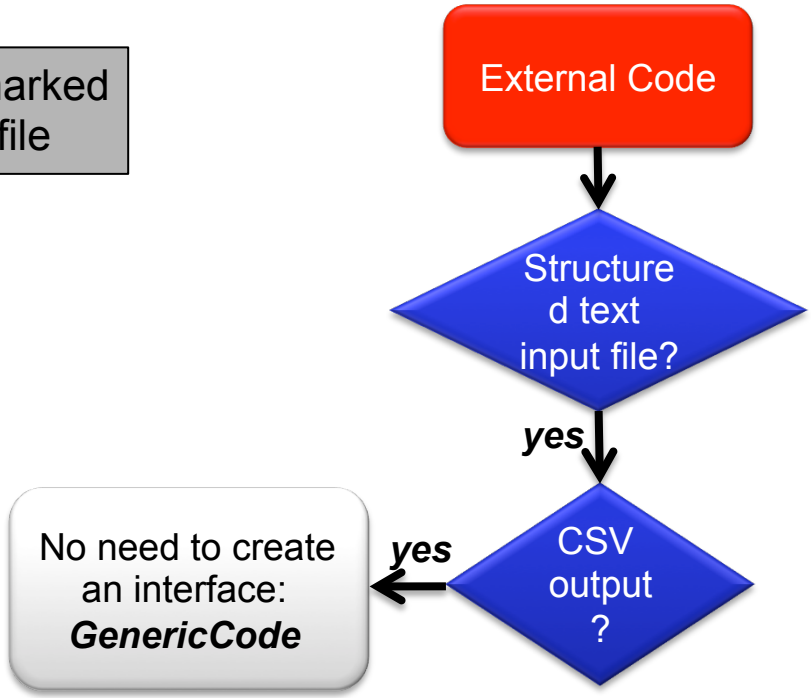
# Coupling strategy: GenericCode interface

- The *GenericCode* interface is meant to handle a wide variety of generic codes. The *GenericCode* interface can be used if the code:
  - Accepts a keyword-based input file with no cross dependent inputs
  - The outputs are stored in a CSV file

```
<variableToChange>
 $RAVEN-variableName$
</variableToChange>
```

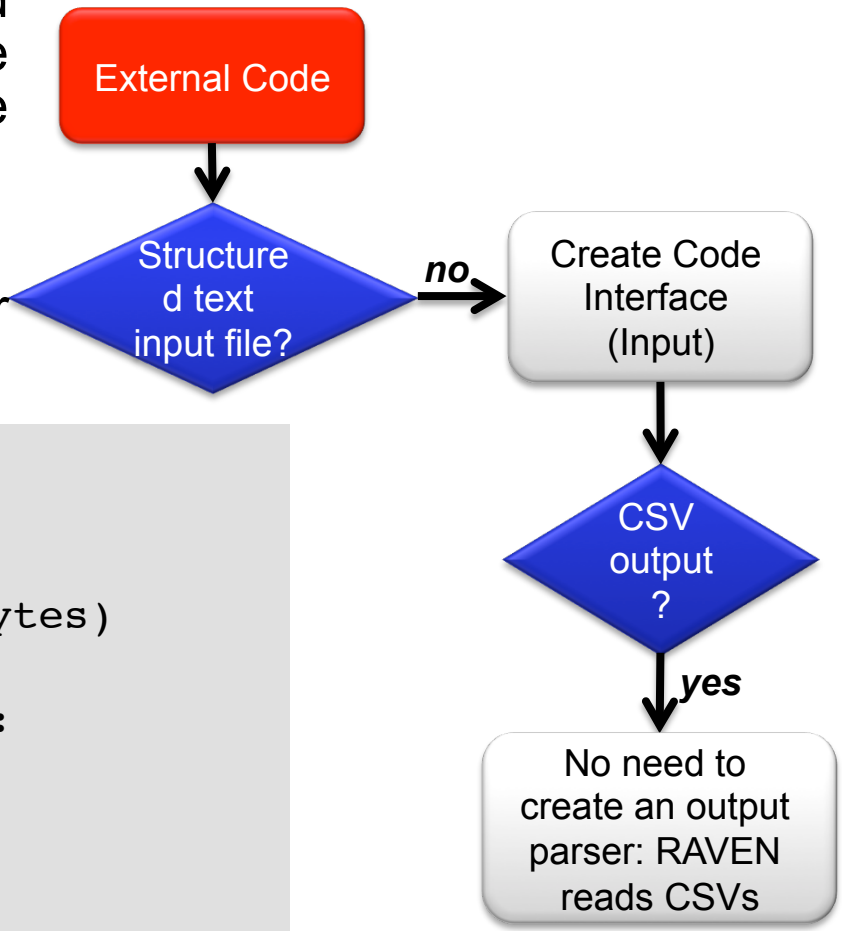
Variable needs to be marked in the original input file

```
<Code>
 <executable>path/to/exe</executable>
 <inputExtcensions>
  .xml,.aux
 </inputExtcensions>
 <clargs type='prepend' arg='python'/>
 <clargs type='input' arg='-i'
         extension='.xml'/>
 <clargs type='input' arg='-aux'
         extension='.aux'/>
 <clargs type='output' arg='-o'/>
</Code>
```



# Coupling strategy: No text-based input format

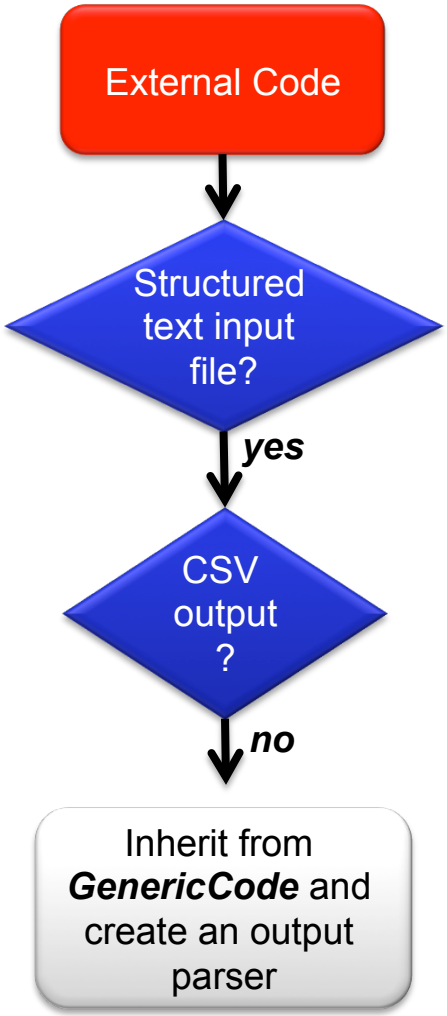
- If the code input file is not text-based (e.g. binary) or too complicated to handle with "wild-cards", the GenericCode interface can not be used:
  - An input-parser needs to be created
- If the output file is a CSV, no other parsers are needed




```
class parserBinaryInput():
  def __init__(self,filen):
    bytes=open(filen,"rb").read()
    self.unPack = struct.unpack("FMT", bytes)

  def perturbTheInput(self,inDictionary):
    # perturb the binary file

  def writeNewInput(self,filen):
    fileObj = open(filen, "wb")
    fileObj.write(struct.pack("FMT", self.unPack))
```

# Coupling strategy: Text input but no CSV output

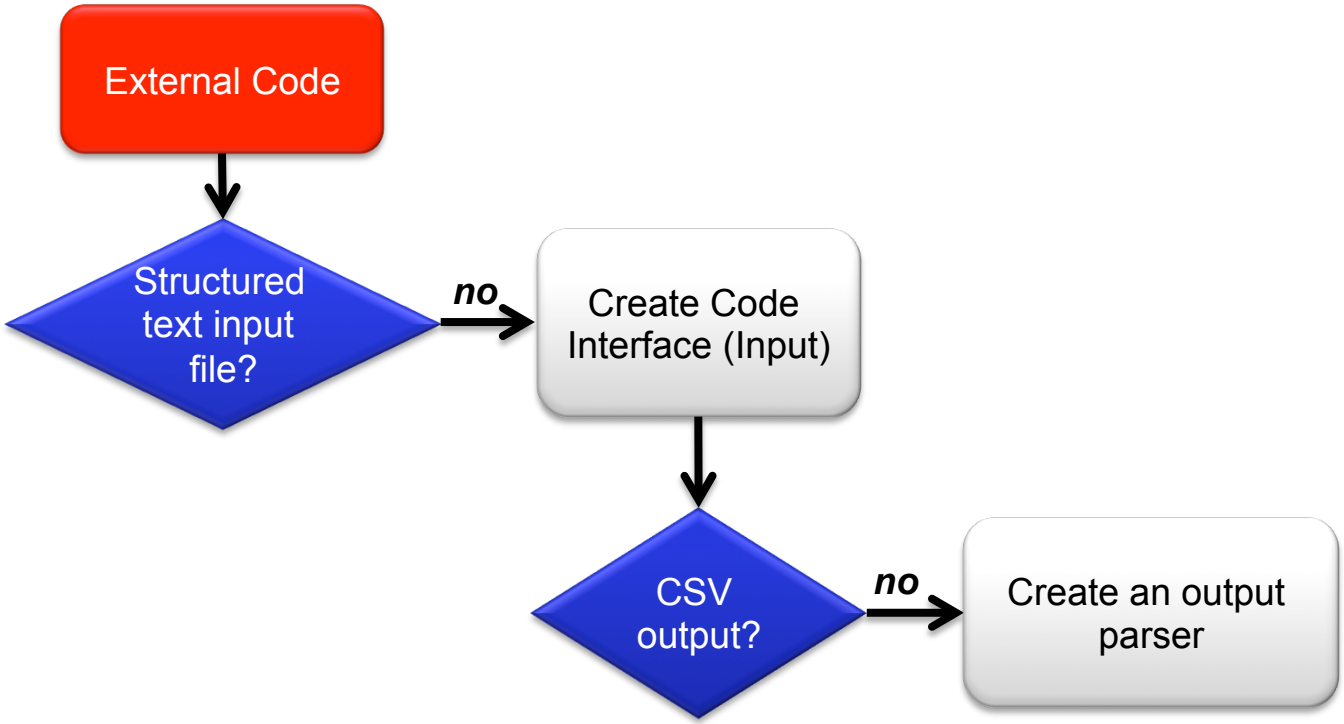
- If the code input file is text-based, the *GenericCode* interface can be used for the input perturbation
- If the output file is not a CSV, an output parser needs to be created




```
def convertOutputFileToCSV(outfile):
  keyDict = {}
  CSVfile = open(outputfile + '.csv')
  lines = open(outputfile).readlines()
  for line in lines:
    values = line.split("=")
    key,val = values[0], values[1]
    keyDict[key] = val
  CSVfile.write(','.join(keyDict.keys()))
  CSVfile.write(','.join(keyDict.values()))
```

# Coupling strategy: Brand-new interface

- If the input structure is too complicated or the interface developer prefers a more specific perturbation syntax and the output file is not a CSV, a new code interface needs to be created:
  - Input Parser, for the input reading and perturbation
  - Output Parser, for converting the output file(s) into a single CSV



# Coupling an App with RAVEN: Methods

- RAVEN imports all the “Code Interfaces” at run-time, without actually knowing the syntax of the driven codes
- In order to make RAVEN able to drive a new Application, a `Python` module containing few methods (strict syntax) needs to be implemented:

```
class newApplication(CodeInterfaceBase):
```

```
 def generateCommand(self,input,exe,clargs,fargs)
```
Required

```
 def createNewInput(self,inputs,oinputs,samplerType,**Kwargs)
```
Required

```
 def finalizeCodeOutput(self,command,output,workDir)
```
Optional

```
 def checkForOutputFailure(self,output,workDir)
```
Optional

```
 def getInputExtension(self)
```
Optional

```
 def setInputExtension(self, exts)
```
Optional

# Coupling an Application with RAVEN: generateCommand

- Used to:
  - retrieve the command needed to launch the driven Application
  - retrieve the root of the output file
- The return data type must be a TUPLE

```
def generateCommand(self,inputs,exe,clargs,fargs):
  (…)
  return (executeCommand,outfile)
```

Required

- Arguments:
  - *inputs:* *list of current input files*
  - *exe:* *executable absolute path*
  - *clargs:* *dictionary of command line flags the user can specify under the <Code> block*
  - *fargs:* *dictionary of command line flags for identifying auxiliary input files that the user can specify under the <Code> block*

# Coupling an Application with RAVEN: createNewInput

- Used to generate an input based on the information that RAVEN s
- This method needs to return a list containing the path and filenames of the modified input files

```
def createNewInput(self,inputs,oinputs,samplerType,**Kwargs):
  (…)
  return newInputFiles
```

Required

- Arguments:
  - *inputs : list of current input files (the one that should be modified)*
  - *oinputs : list of original input files (unperturbed ones)*
  - *samplerTyp: sampler type (e.g. None, MonteCarlo, Grid, etc.)*
  - *Kwargs : dictionary of all the information needed to create an input. Eg:*
    - *executable: executable absolute path*
    - *SampledVars: dictionary of the sampled variables ({'var1":newValue}*
    - *ProbabilityWeight*: float representing the Pb weight of this realization
    - *crowDist*: dictionary containing the info regarding the distributions associated to the variables
    - etc.

# Coupling an Application with RAVEN: finalizeCodeOutput

- This method is called, if present, by RAVEN at the end of each run. It can be used, for example, to convert the whatever Application output format into a CSV
- RAVEN checks if a string is returned
  - RAVEN interprets that string as the new output file root

```
def finalizeCodeOutput(self,command,output,workDir):
  (…)
  return newOutputRoot
```

Optional

- Arguments:
  - *command* : *the command that has been generated by **generateCommand** method*
  - *output* : *the current output root (from **generateCommand** method)*
  - *workDir* : *the current working directory (where the code is currently outputting)*

# Coupling an App with RAVEN: checkForOutputFailure

- Used to check if a run failed even if the *returncode = 0*
- This method is called, if present, by RAVEN at the end of each run.
- It must return a Boolean value. True if failure, False otherwise

```
def checkForOutputFailure(self,output,workDir):
  (…)
  return failure
```

Optional

- Arguments:
  - *output* : *the current output root (from **generateCommand** method)*
  - *workDir* : *the current working directory (where the code is currently outputting)*

# *Practical example*

# Practical example: Simple code I/O overview

- Simple Python code that simulates 4 isotope chain evolutions

**Input File**

```
<AnalyticalBateman>
  <totalTime>10</totalTime>
  <powerHistory>1 1 1</powerHistory>
  <flux>10000 10000 10000</flux>
  <stepDays>0 100 200 300</stepDays>
  <timeSteps>10 10 10</timeSteps>
  <nuclides>
    <A>
        <equationType>N1</equationType>
        <initialMass>1.0</initialMass>
        <decayConstant>1e-08</decayConstant>
        <sigma>6.16193701021</sigma>
        <ANumber>230</ANumber>
    </A>
    <B>
        <equationType>N2</equationType>
        <initialMass>1.0</initialMass>
        <decayConstant>1e-08</decayConstant>
        <sigma>6.16193701021</sigma>
        <ANumber>200</ANumber>
    </B>
    <C>
        <equationType>N3</equationType>
        <initialMass>1.0</initialMass>
        <decayConstant>0.000000005</decayConstant>
        <sigma>45</sigma>
        <ANumber>150</ANumber>
    </C>
    <D>
        <equationType>N4</equationType>
        <initialMass>1.0</initialMass>
        <decayConstant>0.00000008</decayConstant>
        <sigma>3</sigma>
        <ANumber>100</ANumber>
    </D>
  </nuclides>
</AnalyticalBateman>
```

**Text output**

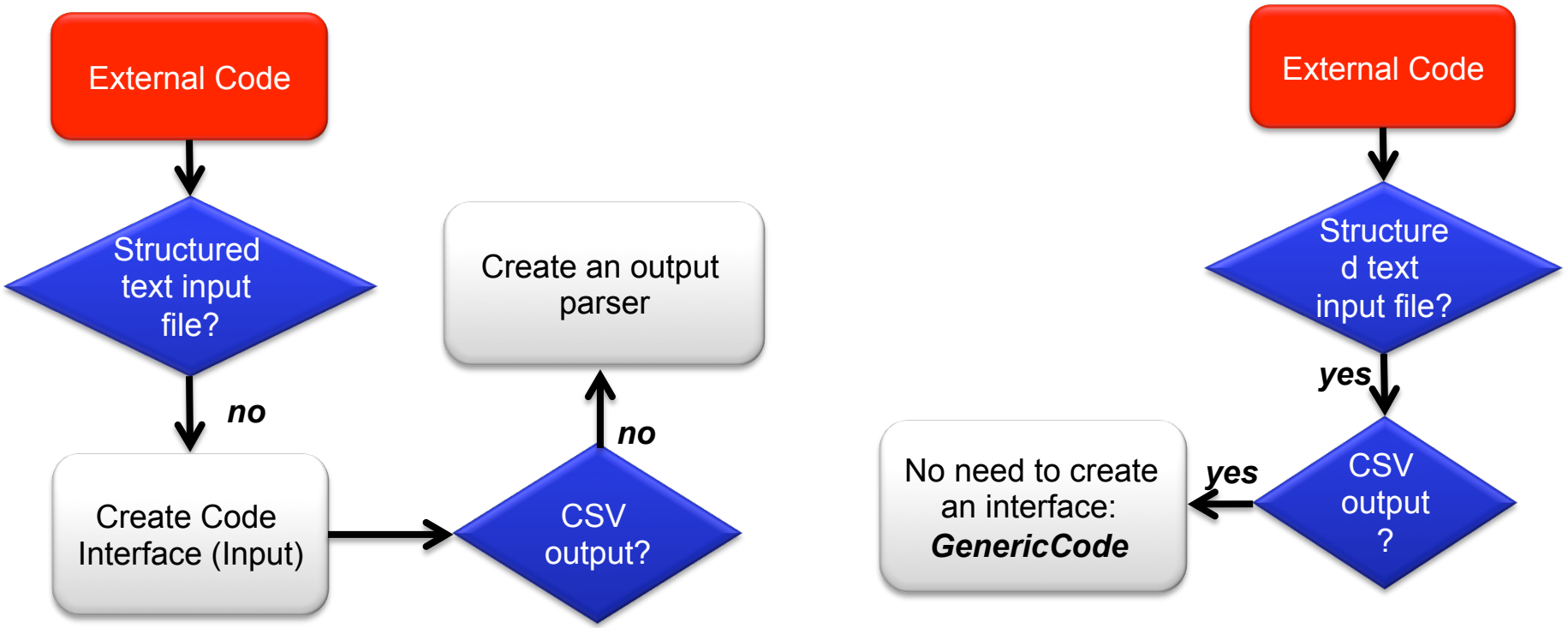
```
Copyright 2017 Battelle Energy Alliance, LLC

Licensed under the Apache License, Version 2.0 (the 'License');
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an 'AS IS' BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

------------------------------------------------------------------------

Input is   : batemanWorkshopInput.xml
Outputs are: results.csv | results.out

------------------------------------------------------------------------
RESULTS:
time         A            C            B            D
0.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00
8.640000E+06 9.172273E-01 9.577198E-01 9.711734E-01 1.042627E+00
1.296000E+07 8.784467E-01 9.372549E-01 9.568156E-01 1.063448E+00
1.728000E+07 8.413059E-01 9.172273E-01 9.424941E-01 1.083870E+00
2.592000E+07 7.716687E-01 8.784467E-01 9.140025E-01 1.123975E+00
CPU TIME: 0.00283598899841 s----------------------------------------
SUCCESS
```

**CSV output**

| time | A | C | B | D |
|---|---|---|---|---|
| 0.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 |
| 8.64E+06 | 9.17E-01 | 9.58E-01 | 9.71E-01 | 1.04E+00 |
| 1.30E+07 | 8.78E-01 | 9.37E-01 | 9.57E-01 | 1.06E+00 |
| 1.73E+07 | 8.41E-01 | 9.17E-01 | 9.42E-01 | 1.08E+00 |
| 2.59E+07 | 7.72E-01 | 8.78E-01 | 9.14E-01 | 1.12E+00 |

# Practical example: 2 Examples

- This simple code allows to test two distinct examples:
  1. Creation of a brand new interface (parsing the XML input and the Text Output)
  2. Usage of the *GenericCode* interface since the XML input can be perturbed with “wild-cards” and the code generates already a CSV



*Thank you*

*Questions?*