



IPS Framework & Component Design: A Brief Tutorial

**Wael R. Elwasif
ORNL**



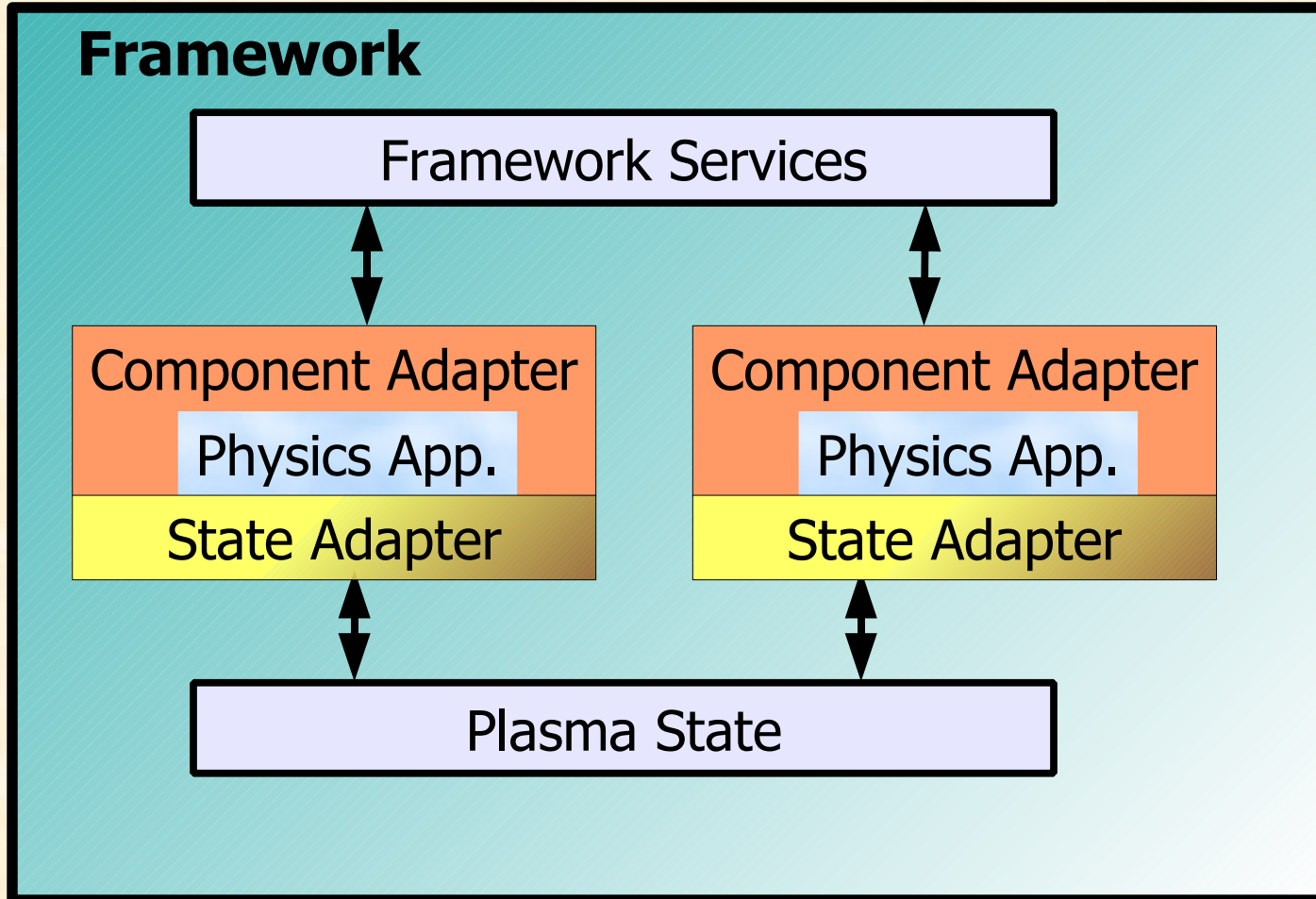
Framework Design Goals

- **Rapid coupling of existing simulation codes.**
- **Minimal (aka NO) change to underlying applications.**
- **Simple coupling and integration protocol.**
- **Integrated data management.**
- **Flexible simulation structure.**

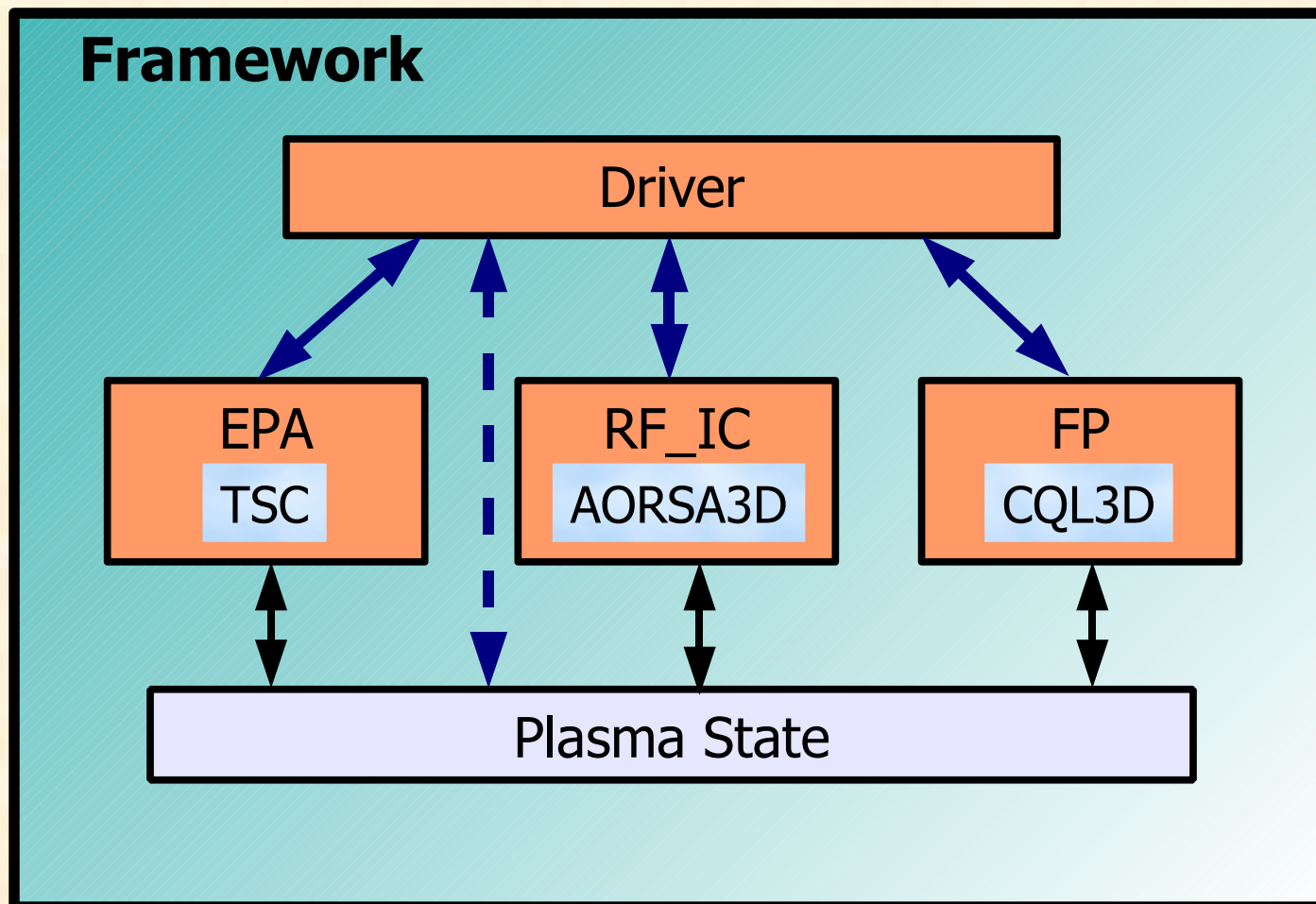
Major Design Features

- **Plasma state**
 - Data repository.
 - Conduit for inter-component data exchange.
- **Scripting Based Framework (Python)**
 - RAD.
 - Adaptability, changeability, and flexibility.
- **Driver/Workers topology:**
 - Simple components connectivity pattern.
- **Unified component interface**
 - `init()`, `step()`, `finalize()`. *Too Simple??*

IPS Framework Layout



Sample IPS Application Structure



IPS Evolution

- **Version 0.1:**
 - **No framework !!!**
 - **Standalone scripts for each component method.**
 - **Data management explicitly managed by each component.**
- **Version 0.2 (current):**
 - **Components implemented as Python objects.**
 - **Framework handles data management, configuration management, and application job launch.**



AORSA3D step() Script – Version 0.1

```

#!/usr/bin/env python

import sys
import os
import subprocess
import getopt
import shutil
IPS_ROOT=''
SIM_ROOT=''
CURRENT_TIME = ''
COMPONENT_CLASS = 'rf'
COMPONENT_SUBCLASS = 'ic'
COMPONENT_NAME = 'aorsa'
NUM_PROC=1
INPUT_FILES = ['aorsa2d.in', 'grfont.dat', 'ZTABLE.TXT']
OUTPUT_FILES = ['out_swim', 'out15', 'aorsa2d.ps', 'aorsa2d.in']

def printUsageMessage():
    print 'Usage: %s --ipsroot=FULL_IPS_ROOT_PATH \
--simroot=FULL_PATH_TO_CURRENT_SIMULATION \
--curtime=CURRENT_TIME_IN_MSEC \
--nproc=NUM_PROCESSORS' % (sys.argv[0])

def main(argv=None):
    global IPS_ROOT, SIM_ROOT, COMPONENT_CLASS,
    COMPONENT_SUBCLASS, COMPONENT_NAME,
    INPUT_FILES, NUM_PROC
    # Parse command line arguments
    if argv is None:
        argv = sys.argv
    try:
        opts, args = getopt.gnu_getopt(argv[1:], '\
["ipsroot=", "simroot=", "curtime=", "nproc="]')
    except getopt.error, msg:
        print 'Exception'
        printUsageMessage()
        return 1
    for arg,value in opts:
        print arg, value
        if (arg == '--ipsroot'):
            IPS_ROOT = value
        elif (arg == '--simroot'):
            SIM_ROOT = value
        elif (arg == '--curtime'):
            CURRENT_TIME = value
        elif (arg == '--nproc'):
            NUM_PROC = value
    if (IPS_ROOT == '' or SIM_ROOT == '' or
CURRENT_TIME == '' ):
        printUsageMessage()
        return 1

    prepare_input = os.path.join(IPS_ROOT, 'bin', \
'prepare_aorsa_input')
    process_output = os.path.join(IPS_ROOT,'bin', \
'process_aorsa_output')
    prepare_eqdsk = os.path.join(IPS_ROOT, 'bin', \
'xeqdsk_setup')
    aorsa_bin = os.path.join(IPS_ROOT, 'bin', 'xaorsa2d')

    #
    # Check existence and/or create working directory for the current run
    workdir = os.path.join(SIM_ROOT, 'work', \
COMPONENT_CLASS, COMPONENT_SUBCLASS, COMPONENT_NAME)
    try:
        os.chdir(workdir)
    except OSError, (errno, strerror):
        print 'Directory %s does not exist - will \
attempt creation' % (workdir)
    try:
        os.makedirs(workdir)
    except OSError, (errno, strerror):
        print 'Error creating directory %s : %s' \
% (workdir, strerror)
        return 1
    os.chdir(workdir)

    # Copy current and prior state over to working directory
    statedir = os.path.join(SIM_ROOT, 'work', 'plasma_state')
    cur_state = 'plasma_state.cdf'
    prior_state = 'prior_state.cdf'
    for f in [cur_state, prior_state]:
        try:
            shutil.copyfile(os.path.join(statedir,f), \
os.path.join(workdir,f))
        except IOError, (errno, strerror):
            print 'Error copying file %s to work \
directory %s : %s' % (f, workdir, strerror)
            return 1

    # Call prepare_aorsa-input
    retcode = subprocess.call([prepare_input])
    if (retcode != 0):
        print 'Error in call to prepare_aorsa_input'
        return 1
    shutil.copyfile(os.path.join(workdir, 'aorsa2d.in_new'), \
os.path.join(workdir, 'aorsa2d.in'))

    # Call xeqdsk_setup to generate eqdsk.out file
    retcode = subprocess.call([prepare_eqdsk])
    if (retcode != 0):
        print 'Error in call to prepare_eqdsk'
        return 1

    # Call aorsa2d (in parallel only if NUM_PROC > 1)
    retcode = subprocess.call(['mpirun', '-np', \
NUM_PROC, aorsa_bin])
    if (retcode != 0):
        print 'Error in call to xaorsa2d'
        return 1

    # How do we know that the call succeeded ??

    # Call process_output and copy files over
    retcode = subprocess.call([process_output])
    if (retcode != 0):
        print 'Error in call to %s' % (process_output)
        return 1

    # Update (original) plasma state
    for f in ['plasma_state.cdf', 'prior_state.cdf']:
        try:
            shutil.copyfile(os.path.join(workdir,f), \
os.path.join(statedir,f))
        except IOError, (errno, strerror):
            print 'Error copying file %s to state directory \
%s : %' % (f, statedir, strerror)
            return 1

    # Copy simulation results over to
    # SIM_ROOT/simulation_results/history/current_time

    targetdir = os.path.join(SIM_ROOT, 'simulation_results',
'history', CURRENT_TIME, 'components', COMPONENT_CLASS,
COMPONENT_SUBCLASS, COMPONENT_NAME)
    try:
        os.makedirs(targetdir)
    except OSError, (errno, strerror):
        if (errno != 17): #Directory exists
            print 'Error accessing directory %s : %s' % \
(targetdir, strerror)
            return 1
        pass
    for f in OUTPUT_FILES:
        try:
            shutil.copyfile(f,
os.path.join(targetdir,f))
        except IOError, (errno, strerror):
            print 'Error copying file %s to %s : %s' % \
(f, targetdir, strerror)
            return 1

    if __name__ == "__main__":
        sys.exit(main())

```



AORSA3D COMPONENT – Version 0.2

```
#!/usr/bin/env python

import sys
import os
import subprocess
import getopt
import shutil
import string
from component import component

class aorsa(component):
    def __init__(self, services, config):
        for i in config.keys():
            try:
                setattr(self, i, config[i])
            except Exception, e:
                print 'Error setting AORSA parameter : ', i, ' - ', e
                sys.exit(1)
        self.services = services
        print 'Created %s' % (self.__class__)

    def init(self):
        print 'aorsa.init() called'
        self.services.stageInputFiles(self, self.INPUT_FILES)
        return

    def step(self, timeStamp):
        print 'aorsa.step() calld'

        if (self.services == None) :
            print 'Error in aorsa:;step () : init() function not
called before step().'
            sys.exit(1)
        services = self.services

        prepare_input = os.path.join(self.BIN_PATH,
'prepare_aorsa_input')
        process_output = os.path.join(self.BIN_PATH,
'process_aorsa_output')
        prepare_eqdsk = os.path.join(self.BIN_PATH, 'xeqdsk_setup')
        aorsa_bin = os.path.join(self.BIN_PATH, 'xaorsa2d')

        # Copy current and prior state over to working directory
        services.stageCurrentPlasmaState(self)

        # Call prepare_aorsa-input
        cur_state_file =
services.getGlobalConfigParameter('CURRENT_STATE')
        cur_eqdsk_file =
services.getGlobalConfigParameter('CURRENT_EQDSK')
        retcode = subprocess.call([prepare_input, cur_state_file,
cur_eqdsk_file])
        if (retcode != 0):
            print 'Error executing ', prepare_input
            return 1
            shutil.copyfile('aorsa2d.in_new', 'aorsa2d.in')

        # Call xeqdsk_setup to generate eqdsk.out file
        # retcode = subprocess.call([prepare_eqdsk])
        # if (retcode != 0):
        #     print 'Error in call to prepare_eqdsk'
        #     return 1

        # Call aorsa2d (in parallel)
        retcode = services.launchJob(aorsa_bin, self.NPROC)
        if (retcode != 0):
            print 'Error executing command: ', aorsa_bin
            sys.exit(1)

        # Call process_output and copy files over
        retcode = subprocess.call([process_output, cur_state_file])
        if (retcode != 0):
            print 'Error executing', process_output
            sys.exit(1)

        # Update (original) plasma state
        services.updatePlasmaState(self)

        # "Archive" output files in history directory
        services.stageOutputFiles(self, timeStamp, self.OUTPUT_FILES)

    def finalize(self):
        print 'aorsa.finalize() called'
```



IPS Framework Features

- ***Configuration*** via simulation configuration file (outlined next).
- ***Instantiates*** components specified in the configuration files.
- ***Mediates*** driver/components connection, and method invocations.
- ***Manages*** data movement and archival.
- ***Handles*** apps invocation (parallel, or serial on a compute node).
- **More services to come (based on usage feedback)**

Simulation Configuration File

- **Four major sections:**
 - **Global configuration options.**
 - ***Ports* configuration.**
 - **Components configuration.**
 - **Time loop specification.**
- **Note:**
 - **Configuration structure can be changed and/or extended easily upon request.**
 - **One change: Split platform-specific data into a separate file (Jaguar, MHD, ..etc).**



Configuration: Global Data

(File: components/drivers/berry/rfsim.conf)

```
IPS_ROOT = /home/elwasif/ips/trunk      # Root of IPS component and binary tree
SIM_NAME = AORSA_SIM                   # Name of current simulation
SIM_ROOT = $IPS_ROOT/$SIM_NAME         # Where to put results from this simulation
PLASMA_STATE_WORK_DIR = $SIM_ROOT/work/plasma_state
                                         # Where to put plasma state files as the simulation evolves

RUN_ID = $SIM_NAME
OUTPUT_PREFIX =
CURRENT_STATE = ${RUN_ID}_ps.cdf
PRIOR_STATE = ${RUN_ID}_psp.cdf
CURRENT_EQDSK = ${RUN_ID}_ps.geq
                                         # What files constitute the plasma state
PLASMA_STATE_FILES = $CURRENT_STATE $PRIOR_STATE $CURRENT_EQDSK

PLATFORM = jaguar                      # Simulation Platform
BATCH_SYSTEM = pbs                     # Which Batch system to use
MPIRUN = yod                           # How are MPI jobs launched (interactively)

SIMULATION_MODE = SINGLE_STEP | RESTART # Simulation mode
INITIALIZATION_MODE =                  # Initialization Mode
MACHINE_CONFIG_FILE =                  # Machine configuration file
```

Configuration: Ports

```
[PORTS]
  NAMES = DRIVER INIT RF_IC EPA LINEAR_STABILITY FOKKER_PLANCK
  [[DRIVER]]                                     # REQUIRED Port section
    IMPLEMENTATION = AORSA_CQL3D_DRIVER
                                     # How is the simulation initialized
                                     # (generate the very first state - if needed)
  [[INIT]]                                       # REQUIRED Port section (check currently disabled)
    IMPLEMENTATION = AORSA_CQL3D_INIT

  [[RF_IC]]
    IMPLEMENTATION = AORSA

  [[EPA]]
    IMPLEMENTATION = TSC

  [[FOKKER_PLANCK]]
    IMPLEMENTATION = CQL3D
```



Configuration: Components

```
[AORSA]
CLASS = rf                               # Component categorization
SUB_CLASS = ic                            # Component's Python class name.
NAME = aorsa                              # Number of processors.
NPROC = 4                                 # Where to look for application
BIN_PATH = $IPS_ROOT/bin                 # Where to look for input files.

INPUT_DIR = $IPS_ROOT/components/$CLASS/$NAME # List of input files
INPUT_FILES = aorsa2d.in grfont.dat ZTABLE.TXT g096028.02650
                                                # List of "important" output files
OUTPUT_FILES = out_swim out15 aorsa2d.ps aorsa2d.in
                                                # Where to find the component
SCRIPT = $BIN_PATH/rf_ic_aorsa.py

# Can add extra component-specific configuration entries here.
```

Configuration: Time Loop

```
# For MODE = REGULAR, the framework uses the variables
# START, FINISH, and NSTEP
# For MODE = EXPLICIT, the framework uses the variable VALUES
# (space separated list of time values)
[TIME_LOOP]
  MODE = EXPLICIT
  START = 3.5
  FINISH = 3.7
  NSTEP = 2
  VALUES = 3.4 3.5 3.6 3.7
```

Component Code



- No need to edit the component constructor (the `__init__()` method).
- Framework always executes code in the same directory
(`$SIM_ROOT/work/$CLASS/$SUB_CLASS/$NAME`).
- Direct access to component configuration variables (through the `self` variable).
- Access to framework services through the `self.services` variable.
- Example uses file: `components/aorsa/src/rf_ic_aorsa.py`

Component code: `init()`

- Called once before time loop.
- One-time input files staging and initialization.
- Example:

```
def init(self):  
    print 'aorsa.init() called'  
    self.services.stageInputFiles(self, self.INPUT_FILES)  
    return
```

Component Code : `step()`

- Signature: `step(self, timeStamp)`.
- Called once for each time step.
- Typically where the calls to the underlying application occur.
- Generally encapsulates five operations:
 - `get_current_plasma_state`
 - `prepare_physics_input` (State  Native Physics)
 - `call_physics_code(s)`
 - `process_physics_output` (Native Physics  State)
 - `update_plasma_state`

Sample step() code: AORSA

```
def step(self, timeStamp):
    print 'aorsa.step() called'

    if (self.services == None) :
        print 'Error in aorsa: step(): init() function not called
before step().'
        sys.exit(1)
    services = self.services

    prepare_input = os.path.join(self.BIN_PATH,
'prepare_aorsa_input')
    process_output = os.path.join(self.BIN_PATH,
'process_aorsa_output')
    prepare_eqdsk = os.path.join(self.BIN_PATH, 'xeqdsk_setup')
    aorsa_bin = os.path.join(self.BIN_PATH, 'xaorsa2d')
```

AORSA step() code (cont.)

```
# Copy current and prior state over to working directory
services.stageCurrentPlasmaState(self)

# Call prepare_aorsa-input
cur_state_file =

services.getGlobalConfigParameter('CURRENT_STATE')
cur_eqdsk_file =

services.getGlobalConfigParameter('CURRENT_EQDSK')
retcode = subprocess.call([prepare_input,
cur_state_file,
cur_eqdsk_file])
if (retcode != 0):
    print 'Error executing ', prepare_input
    return 1
shutil.copyfile('aorsa2d.in_new', 'aorsa2d.in')
```

AORSA step() code (cont.)

```
# Call aorsa2d (in parallel)
retcode = services.launchJob(aorsa_bin, self.NPROC)
if (retcode != 0):
    print 'Error executing command: ', aorsa_bin
    sys.exit(1)

# Call process_output and copy files over
retcode = subprocess.call([process_output, cur_state_file])
if (retcode != 0):
    print 'Error executing', process_output
    sys.exit(1)

# Update (original) plasma state
services.updatePlasmaState(self)

# "Archive" output files in history directory
services.stageOutputFiles(self, timeStamp, self.OUTPUT_FILES)
```

Component code: finalize()

- Called once after completion of time loop.
- Any necessary cleanup and or logging.
- Example:

```
def finalize(self):  
    print 'aorsa.finalize() called'  
    returns
```



Framework Services (1)

Launch a (possibly parallel) interactive job on one or more compute nodes.
`launchJob(executable, nproc, *args)`

Retrieve the working directory for the component argument.
`getWorkingDirectory(component)`

Copy input files for the component argument into the component's work directory.
`stageInputFiles(component, inputFileList)`

Copy output files into the component's archival directory for "timestamp".
`stageOutputFiles(component, timeStamp, outputFileList)`

Get the most recent plasma state file(s) into the component's working directory.
`stageCurrentPlasmaState(component)`

*# Copy the component's newly updated plasma state file(s) to the
common state work directory.*
`updatePlasmaState(component)`

Return the value of a global configuration parameter
`getGlobalConfigParameter(configPar)`

Log message from a component using the framework's logging mechanism
`logMessage(*args)`

Framework Services (2): Driver methods

*# Call method "methodName" on component instance "component", passing
arguments "*args" to the called method.*
`call(component, methodName, *args)`

Return a reference to a component that implements port portID.
`getPort(portID)`

*# Return a list of time instances (computed by the framework based on
the TIME_LOOP section of the configuration file).*
`getTimeLoop()`

Framework operation

- Parse config. File.
- Find and instantiate components specified in the config file.
- Call the `init()`, `step()`, and `finalize()` methods on the component implementing the INIT port.
- Call the `init()`, `step()`, and `finalize()` methods on the component implementing the DRIVER port.

Miscellaneous

- **Framework invocation:**
 - `ips --config=CONFIG_FILE_NAME \`
`--log=SIM_LOGFILE_NAME`
- **Limitations**
 - Cannot use more than one component that implements the same port in the same simulation.
- **Debugging:**
 - All input files are located correctly.
 - Python class name in the config file matches the name in the SCRIPT config. Variable.
 -

IPS Drivers

- **Regular components.**
- **Specified separately in the PORTS section of the configuration file (Required entry, DRIVER = . . .).**
- **Encapsulate the simulation logic.**
- **The only component that connects to other components (currently).**



Sample driver code: RF_IC/EPA Coupling (in components/drivers/berry/new_berry_driver.py)

```
class testDriver(component):  
    def __init__(self, services, config):  
        for i in config.keys():  
            try:  
                setattr(self, i, config[i])  
            except Exception, e:  
                print 'Error setting TESTDRIVER parameter : ',  
                    i, ' - ', e  
                sys.exit(1)  
        self.services = services  
        print 'Created %s' % (self.__class__)
```

```
def init(self):  
    return
```

Sample driver code: (cont.)

```
def step(self, timestamp=0):
    services = self.services

#     services.setWorkingDirectory(self)

    rfComp = services.getPort('RF_IC')
    profAdvanceComp = services.getPort('EPA')

    if(rfComp == None or profAdvanceComp == None):
        print 'Error accessing physics components'
        sys.exit(1)

    timeloop = services.getTimeLoop()
    tlist_str = ['%.2f'%t for t in timeloop]

    services.call(rfComp, 'init')
    services.call(profAdvanceComp, 'init')
```

Sample driver code (cont.)

```
for t in tlist_str:
    print 'Current time = ', t
    services.call(rfComp, 'step', t)
    services.call(profAdvanceComp, 'step', t)

    services.stageCurrentPlasmaState(self)
    services.stageOutputFiles(self, t, self.OUTPUT_FILES)

services.call(rfComp, 'finalize')
services.call(profAdvanceComp, 'finalize')
```

```
def finalize(self):
    pass
```

Going forward

- **Populating the repository, *PLEASE*.**
- **Framework extensions – support for multiple simultaneous component(s) invocations.**
- **Output directories layout and naming conventions, what's the most usable layout?.**
- **Rich(er) interaction between the driver and components, unexpected events, control flow, ...**
- **Richer interfaces, direct intra-component connections, in-memory coupling.**
- **.....**