



# Writing Components

**CCA Forum Tutorial Working Group**

<http://www.cca-forum.org/tutorials/>

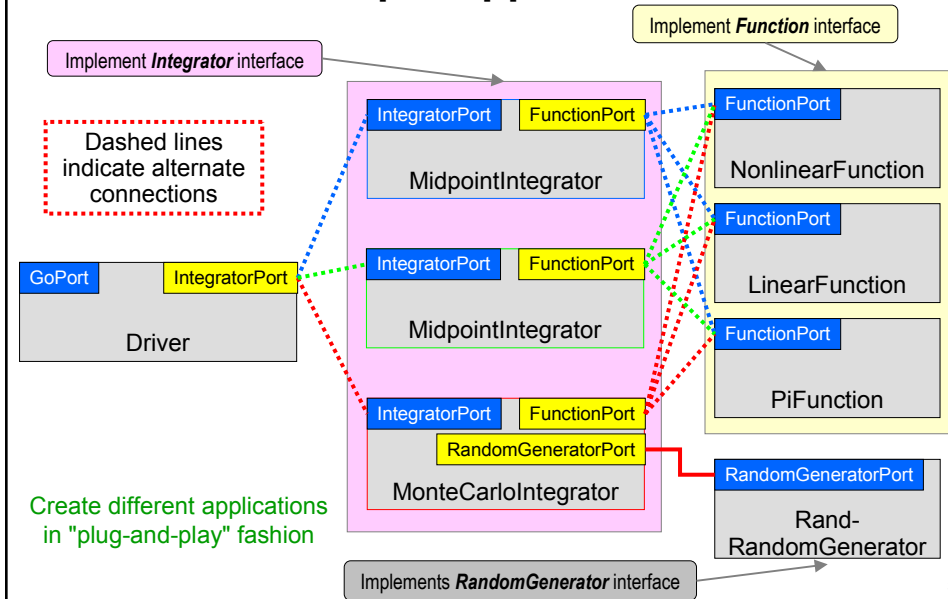
[tutorial-wg@cca-forum.org](mailto:tutorial-wg@cca-forum.org)



## Module Overview

- Goal: present a step-by-step approach to creating CCA components
- Example application
- Steps involved in writing CCA components
  1. Interface definition; ports
  2. Component implementation
    1. Framework interactions
    2. Component interactions: uses and provides ports
  3. Compiling
  4. Running

## Example Applications



3

## Interface Definition

- Component functionality:
  - Random number generator
    - Generates a pseudo-random number
  - Integrator
    - Computes the integral of a scalar function
  - Function
    - Computes a scalar function
  - Driver
    - Entry point into the application

4

## MonteCarloIntegrator Component

1. Use Babel to generate C++ skeletons and implementation files from integrator.sidl
2. Fill in implementation details in integrator-component-c++/:
  - integrator\_MonteCarloIntegrator\_Impl.hh
  - integrator\_MonteCarloIntegrator\_Impl.cc
3. Create C wrapper functions (for component creation):
  - integrator\_Integrator\_wrapper\_Impl.cc
4. Create makefile and build dynamic library
  - Makefile
  - libIntegrator-component-c++.so
5. Create integrator.cca (Ccaffeine-specific)

5

## Integrator Port

```
version integrators 1.0;

package integrators {

  interface Integrator extends gov.cca.Port {
    double integrate(in double lowBound,
                    in double upBound, in int count);
  }

}
```

*integrators.Integrator*

*gov.cca.Component*

MonteCarloIntegrator

Inheritance Tree

IntegratorPort   FunctionPort  
RandomGeneratorPort  
MonteCarloIntegrator

Relevant files:  
integrator.sidl  
function.sidl  
random.sidl

6

## Using Babel to Create The Repository

- A repository containing XML versions of the SIDL definition is created first; it will be used for name resolution later
- Makefile fragment (for all SIDL definitions in this example):

```
SIDLFILES = cca.sidl integrator.sidl function.sidl \  
            random.sidl driver.sidl  
  
.repository: $(SIDLFILES)  
    rm -f repository/*.xml \  
    babel --xml --repository-path=repository \  
          --output-directory=repository $(SIDLFILES)  
    touch .repository
```

7

## Using Babel to Generate Code

- Makefile fragment (top-level directory):

```
.integrator-component-c++: integrator.sidl cca.sidl  
    babel --server=C++ --repository-path=repository \  
    --output-directory=integrator-component-c++ \  
    --suppress-timestamp integrators \  
    randomgen.RandomGenerator functions.Function  
    touch .integrator-component-c++
```

- Important: the *randomgen.RandomGenerator* and *functions.Function* interfaces are referenced by the Integrator implementation(s) and are thus included in the command line for generating the sources for the integrators package.

8

### Contents of integrator-component-c++/

SIDL.hh	gov_cca_ComponentID_IOR.c	integrators_Integrator.hh
SIDL_BaseClass.cc	gov_cca_ComponentID_IOR.h	integrators_Integrator_IOR.c
SIDL_BaseClass.hh	gov_cca_Component_IOR.c	integrators_Integrator_IOR.h
SIDL_BaseException.cc	gov_cca_Component_IOR.h	<b>integrators_Integrator_wrapper_Impl.cc</b>
SIDL_BaseException.hh	gov_cca_Port.cc	integrators_MidpointIntegrator.cc
SIDL_BaseInterface.cc	gov_cca_Port.hh	integrators_MidpointIntegrator.hh
SIDL_BaseInterface.hh	gov_cca_Port_IOR.c	integrators_MidpointIntegrator_IOR.c
SIDL_DLL.cc	gov_cca_Port_IOR.h	integrators_MidpointIntegrator_IOR.h
SIDL_DLL.hh	gov_cca_Services.cc	<b>integrators_MidpointIntegrator_Impl.cc</b>
SIDL_Loader.cc	gov_cca_Services.hh	<b>integrators_MidpointIntegrator_Impl.hh</b>
SIDL_Loader.hh	gov_cca_Services_IOR.c	integrators_MidpointIntegrator_Skel.cc
babel.make	gov_cca_Services_IOR.h	integrators_MonteCarloIntegrator.cc
functions_Function.cc	gov_cca_Type.hh	integrators_MonteCarloIntegrator.hh
functions_Function.hh	gov_cca_TypeMap.cc	integrators_MonteCarloIntegrator_IOR.c
functions_Function_IOR.c	gov_cca_TypeMap.hh	integrators_MonteCarloIntegrator_IOR.h
functions_Function_IOR.h	gov_cca_TypeMap_IOR.c	<b>integrators_MonteCarloIntegrator_Impl.cc</b>
gov_cca_CCAException.cc	gov_cca_TypeMap_IOR.h	<b>integrators_MonteCarloIntegrator_Impl.hh</b>
gov_cca_CCAException.hh	gov_cca_TypeMismatchException.cc	integrators_MonteCarloIntegrator_Skel.cc
gov_cca_CCAExceptionType.hh	gov_cca_TypeMismatchException.hh	integrators_ParallelIntegrator.cc
gov_cca_CCAExceptionType_IOR.c	gov_cca_TypeMismatchException_IOR.c	integrators_ParallelIntegrator.hh
gov_cca_CCAExceptionType_IOR.h	gov_cca_TypeMismatchException_IOR.h	integrators_ParallelIntegrator_IOR.c
gov_cca_CCAException_IOR.c	gov_cca_TypeMismatchException_Impl.cc	integrators_ParallelIntegrator_IOR.h
gov_cca_CCAException_IOR.h	gov_cca_TypeMismatchException_Impl.hh	<b>integrators_ParallelIntegrator_Impl.cc</b>
gov_cca_CCAException_Impl.cc	gov_cca_TypeMismatchException_Skel.cc	<b>integrators_ParallelIntegrator_Impl.hh</b>
gov_cca_CCAException_Impl.hh	gov_cca_Type_IOR.c	integrators_ParallelIntegrator_Skel.cc
gov_cca_CCAException_Skel.cc	gov_cca_Type_IOR.h	randomgen_RandomGenerator.cc
gov_cca_Component.cc	<b>integrators.cca</b>	randomgen_RandomGenerator.hh
gov_cca_Component.hh	integrators.hh	randomgen_RandomGenerator_IOR.c
gov_cca_ComponentID.cc	integrators_IOR.h	randomgen_RandomGenerator_IOR.h
gov_cca_ComponentID.hh	integrators_Integrator.cc	



File: integrator-component-c++/integrators\_MonteCarloIntegrator\_Impl.hh

## MonteCarloIntegrator Component (C++):Implementation Header

```

namespace integrators {

/**
 * Symbol "integrators.MonteCarloIntegrator" (version 1.0)
 */
class MonteCarloIntegrator_impl
{
private:
    // Pointer back to IOR.
    // Use this to dispatch back through IOR vtable.
    MonteCarloIntegrator self;

    // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator._implementation)
    // Put additional implementation details here...
    gov::cca::Services frameworkServices; ← Reference to framework Services object
    // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator._implementation)

    ...

}; // end class MonteCarloIntegrator_impl

} // end namespace integrators

```

## MonteCarloIntegrator Component (C++): Framework Interaction

```

integrators::MonteCarloIntegrator_impl::setServices (
/*in*/ gov::cca::Services services )
throw ()
{
    // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.setServices)
    frameworkServices = services;
    if (frameworkServices._not_nil ()) {
        gov::cca::TypeMap tm = frameworkServices.createTypeMap ();
        gov::cca::Port p = self; // Babel required cast

        // Port provided by all Integrator implementations
        frameworkServices.addProvidesPort (p, "IntegratorPort",
        "integrators.Integrator", tm);

        // Ports used by MonteCarloIntegrator
        frameworkServices.registerUsesPort ("FunctionPort", "functions.Function",
        tm);
        frameworkServices.registerUsesPort ("RandomGeneratorPort",
        "randomgen.RandomGenerator", tm);
    }
    // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices)
}

```

Annotations for the above code:

- Save a pointer to the Services object (points to `frameworkServices = services;`)
- Port name (points to `"IntegratorPort"`)
- Port type (points to `"integrators.Integrator"`)
- TypeMap reference (points to `tm`)

## MonteCarloIntegrator Component (C++): integrate() Method

```

double
integrators::MonteCarloIntegrator_impl::integrate (
/*in*/ double lowBound, /*in*/ double upBound, /*in*/ int32_t count ) throw ()
{
    // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate)
    gov::cca::Port port;
    double sum = 0.0;
    functions::Function function_m;
    randomgen::RandomGenerator random_m;

    random_m = frameworkServices.getPort ("RandomGeneratorPort");
    function_m = frameworkServices.getPort ("FunctionPort");

    for (int i = 0; i < count; i++) {
        double x = random_m.getRandomNumber ();
        sum = sum + function_m.evaluate (x);
    }

    frameworkServices.releasePort ("RandomGeneratorPort");
    frameworkServices.releasePort ("FunctionPort");

    return (upBound - lowBound) * sum / count;
    // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.integrate)
}

```

Annotations for the above code:

- Get a RandomGenerator reference (points to `random_m = frameworkServices.getPort ("RandomGeneratorPort");`)
- Get a Function reference (points to `function_m = frameworkServices.getPort ("FunctionPort");`)
- Get a random number (points to `double x = random_m.getRandomNumber ();`)
- Evaluate function at random value (points to `sum = sum + function_m.evaluate (x);`)
- Release ports (points to `frameworkServices.releasePort ("RandomGeneratorPort");` and `frameworkServices.releasePort ("FunctionPort");`)
- Return integral value (points to `return (upBound - lowBound) * sum / count;`)

## MonteCarloIntegrator Component (F77): Framework Interaction

### subroutine

```

& integrators_MonteCarloIntegrator_setServices_impl(self,
& services)
implicit none
integer*8 self
integer*8 services
C DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.setServices)
C Insert the implementation here...
integer*8 tm, excpt, retval, myport, myservices
common/MonteCarloState/myservices

myservices = services
call gov_cca_Services_addReference_f(services)
call gov_cca_Services_createTypeMap_f(myservices, tm, excpt)
call integrators_MonteCarloIntegrator_cast_f(self,
& Port type → 'gov.cca.Port', myport)
call gov_cca_Services_addProvidesPort_f(myservices, myPort,
& 'IntegratorPort', 'integrators.Integrator',
& tm, excpt)
C The ports I use
call gov_cca_Services_registerUsesPort_f(myservices,
& 'FunctionPort', 'functions.Function', tm, excpt)
call gov_cca_Services_registerUsesPort_f(myservices,
& 'RandomGeneratorPort',
& 'randomgen.RandomGenerator', tm, excpt)
C DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices)
end

```

Save a handle to the Services object

TypeMap reference

Explicit cast to Port

Port type

Port name

TypeMap reference

Port name

## MonteCarloIntegrator Component (F77): integrate() Method

```

subroutine integrators_MonteCarloIntegrator_integrate_impl(self,
& lowBound, upBound, count, retval)
implicit none
integer*8 self
double precision lowBound, upBound
integer*4 count
double precision retval
C DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate)
C Insert the implementation here...
integer*8 excpt, port, funport, randport, myservices
common/MonteCarloState/myservices
double precision sum, h, x, temp
integer i

call gov_cca_Services_getPort_f(myservices, 'FunctionPort', port, excpt)
call gov_cca_Services_getPort_f(myservices, 'RandomGeneratorPort', port, excpt)
call gov_cca_Services_getPort_f(myservices, 'RandomGeneratorPort', port, excpt)
call gov_cca_Services_cast_f(port, 'randomgen.RandomGenerator', randport)
sum = 0.0
do 50 i = 0, count, 1
    call randomgen_RandomGenerator_getRandomNumber_f(randport, x)
    x = lowBound + (upBound - lowBound) * x
    call functions_Function_evaluate_f(funport, x, temp)
    sum = sum + temp
50 end do
C Release ports
call gov_cca_Services_releasePort_f(myservices, 'FunctionPort', excpt)
call gov_cca_Services_releasePort_f(myservices, 'RandomGeneratorPort', excpt)
retval = (upBound - lowBound) * sum / count
C DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.integrate)
end

```

Get a Function reference

Get a RandomGenerator reference

Get a random number

Evaluate function at random value

Release ports

Return integral value

## Writing the C Wrapper

- At present, Ccaffeine requires some C functions for dynamic loading of components; example for two components:

```
#include "integrators.hh"
#include "gov_cca_Component.hh"
#include <stdio.h>

extern "C" {
  gov::cca::Component create_MonteCarloIntegrator() {
    ::gov::cca::Component ex = integrators::MonteCarloIntegrator::_create();
    return ex;
  }
  gov::cca::Component create_ParallelIntegrator() {
    ::gov::cca::Component ex = integrators::ParallelIntegrator::_create();
    return ex;
  }
  char **getComponentList() {
    static char *list[3];
    list[0] = "create_MonteCarloIntegrator integrators.MonteCarloIntegrator";
    list[1] = "create_ParallelIntegrator integrators.ParallelIntegrator";
    list[2] = 0;
    return list;
  }
}
```

Annotations:

- Create a MonteCarloIntegrator instance (points to `integrators::MonteCarloIntegrator::_create()`)
- Create a ParallelIntegrator instance (points to `integrators::ParallelIntegrator::_create()`)
- Return a list of components contained in this dynamic library (points to `list`)
- C wrapper function name (points to `create_MonteCarloIntegrator`)
- Component name (points to `integrators.MonteCarloIntegrator`)

5

## MonteCarloIntegrator: integrators.cca

- Ccaffeine-specific file specifying the name of the dynamic library and creation method for each component

```
!date=Thu Aug 15 14:53:23 CDT 2002
!location=
!componentType=babel
libIntegrator-component-c++.so
create_MonteCarloIntegrator integrators.MonteCarloIntegrator
create_ParallelIntegrator integrators.ParallelIntegrator
```

Annotations:

- Component type: "babel" or "classic" (C++) (points to `!componentType=babel`)
- C wrapper function name (points to `create_MonteCarloIntegrator`)
- Component name (points to `integrators.MonteCarloIntegrator`)

*Note: This mechanism is expected to change soon*

16



## Parallel Example: **MidpointIntegrator** integrate() Method

```
double
integrators::ParallelIntegrator_impl::integrate ( /*in*/ double lowBound, /*in*/ double upBound,
/*in*/ int32_t count ) throw ()
{
  // DO-NOT-DELETE splicer.begin(integrators.ParallelIntegrator.integrate)
  gov::cca::Port port;
  functions::Function function_m;

  // Get Function port
  function_m = frameworkServices.getPort("FunctionPort"); ← Get a Function reference

  int n, myid, numprocs, i;
  double result, myresult, h, sum, x;
  int namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Get_processor_name(processor_name, &namelen); } ← Parallel environment details

  fprintf(stderr, "Process %d on %s: number of intervals = %d\n", myid,
    processor_name, count);
  fflush(stderr);
  // ... Continued on next page...
```

## Parallel Example: **MidpointIntegrator** integrate() Method (continued)

```
// ...
MPI_Bcast(&count, 1, MPI_INT, 0, MPI_COMM_WORLD);
if (count == 0) {
  return -1;
} else {
  h = (upBound - lowBound) / (double) count;
  sum = 0.0;
  for (i = myid + 1; i <= count; i += numprocs) {
    x = h * ((double) i - 0.5);
    sum += function_m.evaluate(x); ← Evaluate function
  }
  myresult = h * sum;

  MPI_Reduce(&myresult, &result, 1, MPI_DOUBLE, MPI_SUM, 0,
    MPI_COMM_WORLD); } ← Compute integral in parallel

  frameworkServices.releasePort("FunctionPort"); ← Release port
  printf("result is %f\n", result);
  return result; ← Return integral value

  // DO-NOT-DELETE splicer.end(integrators.ParallelIntegrator.integrate)
}
```

## RandRandomGenerator Component

1. Use Babel to generate C++ skeletons and implementation files for random.sidl
2. Fill in implementation details in random-component-c++/:
  - randomgen\_RandRandomGenerator\_Impl.hh
  - randomgen\_RandRandomGenerator\_Impl.cc
3. Create C wrapper functions (for component creation).
  - randomgen\_RandomGenerator\_wrapper\_Impl.cc
4. Create makefile and build dynamic library
  - Makefile
  - libRandom-component-c++.so
5. Create random.cca (Ccaffeine-specific)

19

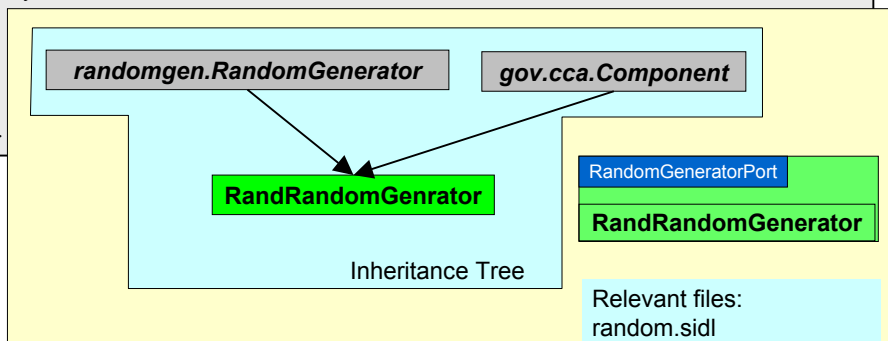
## RandomGenerator Port

```
version randomgen 1.0;

package randomgen {

  interface RandomGenerator extends gov.cca.Port {
    double getRandomNumber();
  }

}
```



20

## RandRandomGenerator Component: Implementation Header

```
namespace randomgen {

/**
 * Symbol "randomgen.RandRandomGenerator" (version 1.0)
 */
class RandRandomGenerator_impl
{
private:
    // Pointer back to IOR.
    // Use this to dispatch back through IOR vtable.
    RandRandomGenerator self;

    // DO-NOT-DELETE splicer.begin(randomgen.RandRandomGenerator._implementation)
    // Put additional implementation details here...
    gov::cca::Services frameworkServices; ← Reference to framework Services object
    // DO-NOT-DELETE splicer.end(randomgen.RandRandomGenerator._implementation)

    ...

}; // end class RandRandomGenerator_impl
} // end namespace randomgen
```

21

## RandRandomGenerator Component: Framework Interaction

```
randomgen::RandRandomGenerator_impl::setServices (
    /*in*/ gov::cca::Services services )
throw ()
{
    // DO-NOT-DELETE splicer.begin(randomgen.RandRandomGenerator.setServices)
    frameworkServices = services; ← Save a pointer to the Services object
    if (frameworkServices._not_nil ()) {
        gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

        gov::cca::Port p = self; // Babel required cast

        // Port provided by RandomGenerator implementations
        frameworkServices.addProvidesPort (p, "RandomGeneratorPort",
            "randomgen.RandomGenerator", tm); ← Port name, TypeMap reference
        // No ports are used by this RandomGenerator implementation
    }
    // DO-NOT-DELETE splicer.end(randomgen.RandRandomGenerator.setServices)
}
```

22

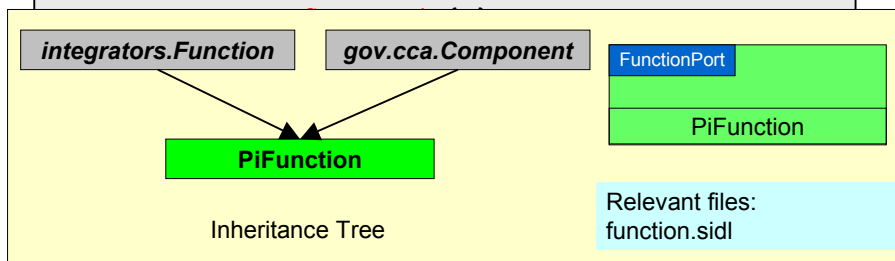
## PiFunction Component

1. Use Babel to generate C++ skeletons and implementation files for function.sidl
2. Fill in implementation details in function-component-c++/:
  - functions\_PiFunction\_Impl.hh
  - functions\_PiFunction\_Impl.cc
3. Create C wrapper functions (for component creation).
  - functions\_Function\_wrapper\_Impl.cc
4. Create makefile and build dynamic library
  - Makefile
  - libFunction-component-c++.so
5. Create functions.cca (Ccaffeine-specific)

23

## Function Port

```
version functions 1.0;  
  
package functions {  
  
  interface Function extends gov.cca.Port {  
    double evaluate(in double x);  
  }  
  
  class PiFunction implements-all Function,
```



24

## PiFunction Component: Implementation Header

```
namespace functions {

/**
 * Symbol "function.PiFunction" (version 1.0)
 */
class PiFunction_impl
{
private:
    // Pointer back to IOR.
    // Use this to dispatch back through IOR vtable.
    PiFunction self;

    // DO-NOT-DELETE splicer.begin(functions.PiFunction._implementation)
    // Put additional implementation details here...
    gov::cca::Services frameworkServices; ← Reference to framework Services object
    // DO-NOT-DELETE splicer.end(functions.PiFunction._implementation)

    ...

}; // end class PiFunction_impl
} // end namespace functions
```

25

## PiFunction Component: Framework Interaction

```
functions::PiFunction_impl::setServices (
/*in*/ gov::cca::Services services )
throw ()
{
    // DO-NOT-DELETE splicer.begin(functions.PiFunction.setServices)
    frameworkServices = services; ← Save a pointer to the Services object
    if (frameworkServices._not_nil ()) {
        gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

        gov::cca::Port p = self; // Babel required cast

        // Port provided by Function implementations
        frameworkServices.addProvidesPort (p, "FunctionPort",
        Port type → "functions.Function", tm); ← TypeMap reference
        Port name →
        // No Ports are used by this Function implementation
    }
    // DO-NOT-DELETE splicer.end(functions.PiFunction.setServices)
}
```

26

## Driver Component

1. Use Babel to generate C++ skeletons and implementation files for driver.sidl
2. Fill in implementation details in driver-component-c++/:
  - tutorial\_Driver\_Impl.hh
  - tutorial\_Driver\_Impl.cc
3. Create C wrapper functions (for component creation).
  - tutorial\_Driver\_wrapper\_Impl.cc
4. Create makefile and build dynamic library
  - Makefile
  - libDriver-component-c++.so
5. Create driver.cca (Ccaffeine-specific)

27

## Driver SIDL Definition

- Driver implements standard gov.cca.ports.GoPort
- No additional interfaces defined

```
version tutorial 1.0;  
  
package tutorial {  
  
    class Driver implements-all gov.cca.ports.GoPort,  
                                gov.cca.Component  
  
    {  
    }  
  
}
```

28

## Driver Component: Framework Interaction

```

tutorial::Driver_impl::setServices (
/*in*/ gov::cca::Services services )
throw ()
{
    // DO-NOT-DELETE splicer.begin(tutorial.Driver.setServices)
    frameworkServices = services; ← Save a pointer to the Services object
    if (frameworkServices._not_nil ()) {
        gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

        gov::cca::Port p = self;    // Babel required cast

        // Port provided by Function implementations
        frameworkServices.addProvidesPort (p, "GoPort",
        "gov.cca.ports.GoPort", tm);
        // Port type
        // TypeMap pointer

        // Port used by the Driver component
        frameworkServices.registerUsesPort ("IntegratorPort",
        "integrators.Integrator", tm);
    }
    // DO-NOT-DELETE splicer.end(tutorial.Driver.setServices)
}

```

## Driver Component: GoPort implementation

```

int32_t
tutorial::Driver_impl::go () throw ()
{
    // DO-NOT-DELETE splicer.begin(tutorial.Driver.go)
    double value;
    int count = 100000; // number of intervals/random samples
    double lowerBound = 0.0, upperBound = 1.0;

    // Ports
    ::gov::cca::Port port;
    ::integrators::Integrator integrator;

    port = frameworkServices.getPort("IntegratorPort");
    integrator = port;
    value = integrator.integrate (lowerBound, upperBound, count);
    fprintf(stdout, "Value = %f\n", value);
    frameworkServices.releasePort ("IntegratorPort");
    return 0;

    // DO-NOT-DELETE splicer.end(tutorial.Driver.go)
}

```

## Build Issues

- Dynamic (shared) libraries
  - For each component or a set of components, build a dynamic library
  - No linking of libraries for components on which current component depends
  - Non-component libraries on which a component depends directly (e.g., BLAS), must be linked explicitly when the shared library is created

31

## Complete Makefile for MonteCarloIntegrator

```
include babel.make

WRAPPERS = integrators_Integrator_wrapper_Impl.cc

INCLUDES = -I$(BABEL_ROOT)/include -I-!$(MPI_HOME)/include

all: libIntegrator-component-c++.so

.c.o:
    gcc -g -fPIC $(INCLUDES) -c $< -o $(<:.c=.o)
.cc.o:
    g++ -g -fPIC $(INCLUDES) -c $< -o $(<:.cc=.o)

OBJS = $(IMPLSRCS:.cc=.o) $(IORSRCS:.c=.o) $(SKELSRCS:.cc=.o) \
        $(STUBSRCS:.cc=.o) $(WRAPPERS:.cc=.o)

LIBS = -Wl,-rpath,$(BABEL_ROOT)/lib -L$(BABEL_ROOT)/lib -lsidl

libIntegrator-component-c++.so: $(OBJS)
    g++ -shared $(INCLUDES) $(OBJS) -o $@ $(LIBS)

clean:
    $(RM) *.o libIntegrator-component-c++.so
```

32



## Running the Example

**Next: Using the Ccaffeine framework**