NATIVE FORTRAN COMPONENTS

in OMS3.1

What does it do?

- Use FORTRAN code in OMS directly
  - No C/C++ bridge required!

- Define OMS components in FORTRAN
- Integrated with build system
- Allow automatic documentation generation from source

Requirements

- FORTRAN 90+ syntax
- MODULES notation
- `ISO_C_BINDING`
- GCC 4.4+ / gfortran
- OMS3.1+

Example

```
! @Execute("we")
SUBROUTINE we(eroout, eroout_len,runoff,peakro,effdrn)

    ! @In
    CHARACTER(kind = C_CHAR, len = eroout_len) :: eroout
    INTEGER(C_INT), intent(in), VALUE :: eroout_len

    ! @In
    REAL(C_FLOAT) :: runoff,peakro,effdrn
    ..
```

- **Annotations**
- **ISO_C_BINDING (F2003+)**
- **F90+ constructs**
- **Declaration part, no API**

Annotations

```
! @Execute("we")
SUBROUTINE we(eroout, eroout_len, runoff,peakro,effdrn)

    ! @In
    CHARACTER(kind = C_CHAR, len = eroout_len) :: eroout
    INTEGER(C_INT), intent(in), VALUE :: eroout_len

    ! @In
    REAL(C_FLOAT) :: runoff,peakro,effdrn
    !
```

- **'Hidden' in language comments**
- **optional arguments**
- **Preceding the language construct**
  - **subroutine, function, argument, module**

Annotations

- Annotations = meta data for source elements

- No arguments

```
@In
```

- Single Value Argument

```
@Execute("Erosion")
```

- Named argument(s)

```
@Description(en="Erosion module",
  de="Erosionsmodul")
```

Module

```
@DLL("Erosion")
MODULE erosion

    USE, INTRINSIC :: ISO_C_BINDING
    IMPLICIT NONE

CONTAINS
   …
END MODULE
```

- **Module definition**
- **DLL annotated**
  - **Naming the shared library**
- **ISO_C_BINDING**

ISO_C_BINDING

- C Interoperability for FORTRAN
- Data type mapping
  - C_DOUBLE
  - C_FLOAT
  - C_CHAR
  - C_BOOL

@Execute

```
! @Execute("we")
SUBROUTINE we(eroout, eroout_len, runoff,peakro,effdrn)
```

- **@Execute** defines entry point

Input

```
! @Description("Erosion output name")
! @In
CHARACTER(kind = C_CHAR, len = eroout_len) :: eroout
INTEGER(C_INT), VALUE :: eroout_len
```

- **@In** input to the subroutine
- Multiple annotations possible, one per line

CHARACTER

```
! @Execute("we")
SUBROUTINE we(eroout, eroout_len, ...

! @In
CHARACTER(kind = C_CHAR, len = eroout_len) :: eroout
INTEGER(C_INT), VALUE :: eroout_len

...
```

- **Character length passed in**
- **Length follows the character array as Int**

INTEGER

```
! @Execute("we")
SUBROUTINE we(..., npart,  ...

! @In
INTEGER(C_INT), VALUE :: npart

…
```

- **Call By value**
- **ISO_C_BINDING: C_INT**

REAL

```
! @Execute("we")
SUBROUTINE we(..., runoff, peakro, effdrn,  ...
...

! @In
REAL(C_FLOAT) :: runoff, peakro, effdrn
...
```

- **Call By value**
- **ISO_C_BINDING: C_FLOAT (4byte) or C_DOUBLE**

Arrays (e.g. REAL)

```
! @Execute("we")
SUBROUTINE we(..., npart, ..., spg,  ...
...

! @In
REAL(C_FLOAT), DIMENSION(npart) :: spg
...
```

- **ISO_C_BINDING: C_FLOAT, C_DOUBLE, C_INT, …**
- **DIMENSION by some other @In (C_INT)**

Shared Libraries

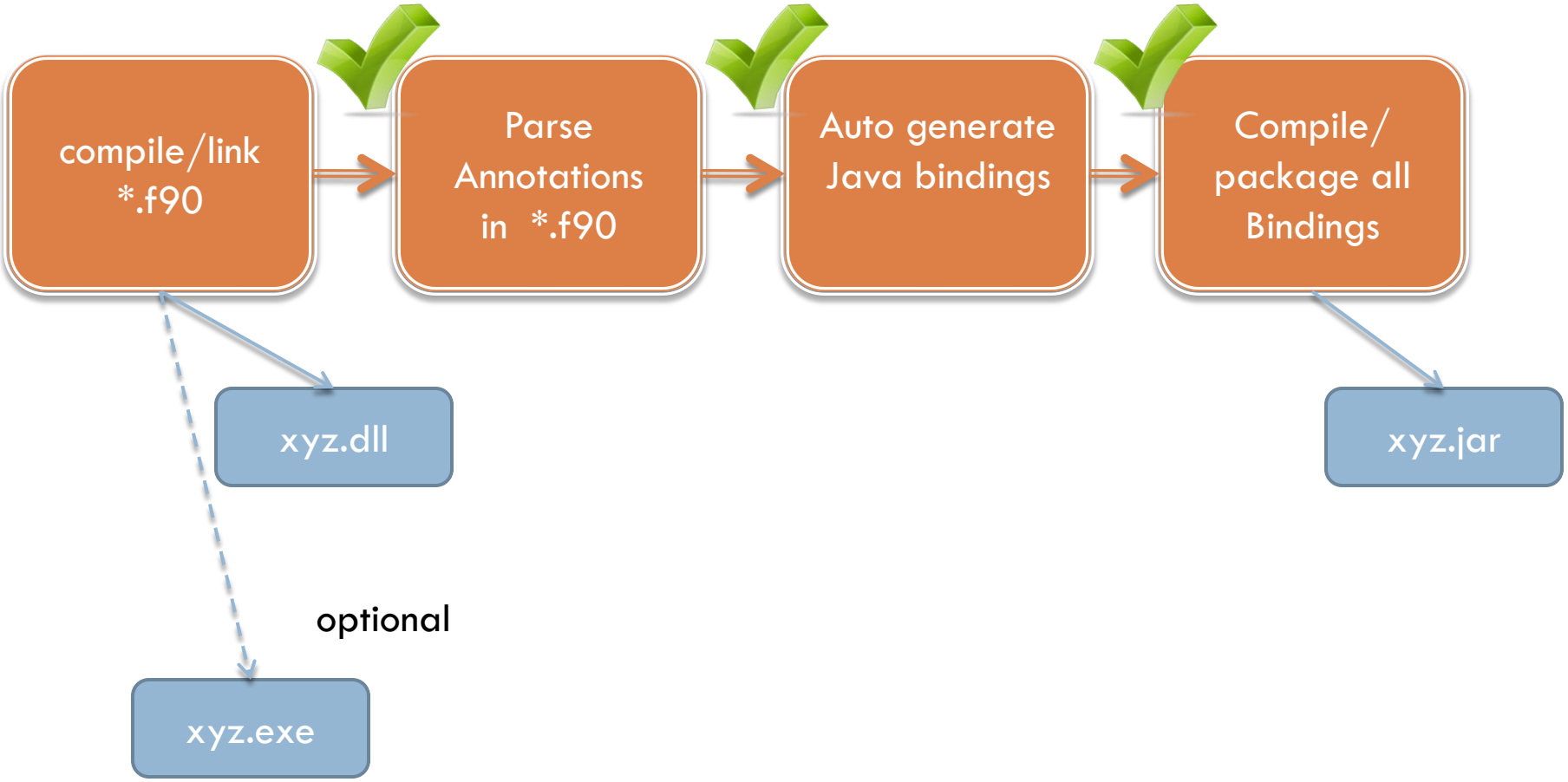
- Native code goes into DLL
  - Erosion.dll (windows)
  - libErosion.so (linux)
- DLL loads at model execution time.
- `@Execute` annotated method gets called

Resources in Simulations

```
model(classname:"hillslope.MainOMS") {
        resource "$oms_prj/dist/*.jar"
         resource "$oms_prj/dist"
        parameter() {
            // output files
            eroout      "/tmp/erosion.out"
            // values
            runoff      "0.127631"
            peakro      "1.88969E-05"
            effdrn      "6754.0983"
        }
    }
```

- **resource** (search path for windows DLLs and shared libraries)
- **resource** for jar file(s)

Fully automated Build system



Conclusions

- Annotations can be used to extend FORTRAN sources.
- Enable existing native sources to seamlessly integrate into OMS3, following the OMS3 annotation conventions.

Resources

- http://gcc.gnu.org/onlinedocs/gfortran/ISO_005fC_005fBINDING.html
- http://gcc.gnu.org