JSFBases/Spring programers guide

- version 1.0 -

Akiya Miyamoto
KEK, 305-0801 Japan
akiya.miyamoto@kek.jp

December 13, 1998

Abstract

How to convert Bases/Spring code written by Fortran so as to run in JSF, version 1-2.

1 Instruction

1.1 Make a template

Create the template by buildjsf command. Let's say the Bases/Spring class name is `XX`.

1.2 What to edit ?

Edit XXSpring.cxx which is created by buildjsf command. A programer has to write following functions.

- `XXBases::XXBases(...)` Set default parameters.
- `XXBases::Func(x)` Calculate function value.
- `XXBases::Userin()` Set aditional parameters and define histogram.
- `XXBases::Usrout()` Output bases result.

In `XXBases::Func` and `XXBases::Usrout`, the programer can simply call corresponding Fortran subroutines which were used togather with lclib: Bases/Spring program to run with lclib has basically four subroutines;

```
REAL FUNCTION*8 FUNC(X) ; Bases user function
SUBROUTINE USERIN       ; Bases user initialization.
SUBROUTINE USROUT       ; Bases user termination
SUBROUTINE BS51CM       ; Get parameter
```

You can call `FUNC` and `USROUT` from `XXBases::Func` and `XXBases::Usrout`. To call fortran functions, `FUNC` and `USROUT` from `XXBases` class function, define fortran function name at the begining of `XXSpring.cxx` as follows,

```
extern "C" {
extern void usrout_();
extern Double_t func_(double x[]);
};
```

1.3 Set parameters ; Constructor and Userin

When JSFBases is used, parameters can be set at three places; (1) At the constructor of `XXBases`, (2) In a ROOT macro, after bases object is created, and (3) In the `Userin` member function of `XXBases` class.

The constructor of the `XXBases` is a place to set the default parameters. With a help of the Environment class of ROOT, user can write the program so that it can obtain parameters from a file. The default parameter file name is jsf.root[?]. For an example, by a statement

```
  sscanf(gJSF->Env()->GetValue("XXBases.Ecm","300.0"),"%g",&fEcm);
```

the default value for the variable `fEcm` is `300.0` and its value can be changed by a line

```
XXBases.Ecm:              300.0
```

in the parameter file, `jsf.conf`.

After a base object is created, parameter values set to the bases class data member can be changed in a macro. This improves the flexibility of application and very nice feature of ROOT system.

The actual bases calculation is performed in `DoBases()` member function of the bases object. In `DoBases()`, `XXBases::Userin()` is called after Bases initialization, then the routine for Bases integration, and `XXBases::Usrout` at the end. The task of `XXBases::Userin()` is to calculate parameters derived from user input and to define histograms. The parameters defined in the XXBases constractor and others defined by the macro are saved as the data member of the XXBases class. In `XXBases::Userin()`, values in the class data member must be copied to the Bases Fortran common. The standard parameters such as `NDIM`, `XL`, `XU`, etc are defined in the JSFBases class and they are loaded into Fortran common block when `JSFBases::Userin()` is called in `XXBases::Userin()`.

To use Bases from a fortran program, user needs to set parameters in commons `BPARM1` and `BPARM2` in the subroutine `USERIN`. In the JSF, clone of variables in these commons are prepared as the data member of the class JSFBases. They can be set in the constructor of XXBases class, and they are copied to `BPARM1` and `BPARM2` in `JSFBases::Userin()`.

1.4 Spring

To use the fortran subroutine, `SPEVNT`, in JSF, basically two member functions, `XXSpringBuf::SetPartons()` and `XXSpring::Initialize()`, must be added to the template code. An example is shown below.

```
extern "C" {
extern void spevnt_(Int_t *nret);
};

//_____________________________________________________________________________
Bool_t XXSpringBuf::SetPartons()
{
  Int_t iret;
  spevnt_(&iret);
  return kTRUE ;
}
//_____________________________________________________________________________
Bool_t XXSpring::Initialize()
{
  // Make sure to set JSFHadronize::fCopySpringClassDataToBank = kFALSE
  JSFSpring::Initialize();
  JSFHadronizer *had=(JSFHadronizer*)gJSF->FindModule("JSFHadronizer");
  if(had) had->SetCopySpringClassDataToBank(kFALSE);
  return kTRUE;
}
```

`XXSpringBuf::SetPartons()` is called at every event to create JSFSpringParton class. In the example above, we call `spevnt_` instead, where TBS Bank `Spring:Parton_List` and `Spring:Header` is created.

The `JSFHadronizer` class is to hadronize partons of class `JSFSpring`. But, if it is called with `fCopySpringClassDataToBank=FALSE`, it uses the data in the TBS bank as an input data.