EFDA
EUROPEAN FUSION DEVELOPMENT AGREEMENT
Task Force
INTEGRATED TOKAMAK MODELLING

02/12/2010

Code Specific Parameters

C. Konz, W. Zwingmann, F. Imbeaux

TF Leader : G. Falchetto
Deputies: D. Coster, R. Coelho

EFDA CSU Contact Person: D. Kalupin

https://www.efda-itm.eu/~wwwimp3/TEST/ITM/html/

Definition

Code Specific Parameters:

All parameters which are specific to the code (like switches, scaling parameters, numerical parameters, etc.).
Generally no data (should go into CPOs).

ITM Convention:

As the rest of the data structures, all code specific parameters should be given in XML format, i.e., in form of an XML string.

type_codeparam

- codename
- codeversion - release version/revision number
- parameters – XML string (code specific parameters)
- output_diag – XML string (code specific diagnostic/output)
- output_flag – module return status

```
type type_codeparam  !
  character(len=132), dimension(:), pointer ::codename => null()       ! /codeparam/codename - Name of the code
  character(len=132), dimension(:), pointer ::codeversion => null()       ! /codeparam/codeversion - Version of the code (as in the ITM repository)
  character(len=132), dimension(:), pointer ::parameters => null()       ! /codeparam/parameters - List of the code specific parameters, string expected to be in XML format.
  character(len=132), dimension(:), pointer ::output_diag => null()       ! /codeparam/output_diag - List of the code specific diagnostic/output, string expected to be in XML format.
  integer  :: output_flag=-999999999       ! /codeparam/output_flag - Output flag : 0 means the run is successful, other values meaning some difficulty has been encountered, the exact meaning is then
endtype

type type_param  !
  character(len=132), dimension(:), pointer ::parameters => null()       ! /param/parameters - Actual value of the code parameters (instance of coparam/parameters in XML format).
  character(len=132), dimension(:), pointer ::default_param => null()       ! /param/default_param - Default value of the code parameters (instance of coparam/parameters in XML format).
  character(len=132), dimension(:), pointer ::schema => null()       ! /param/schema - Code parameters schema.
endtype
```

Fortran90

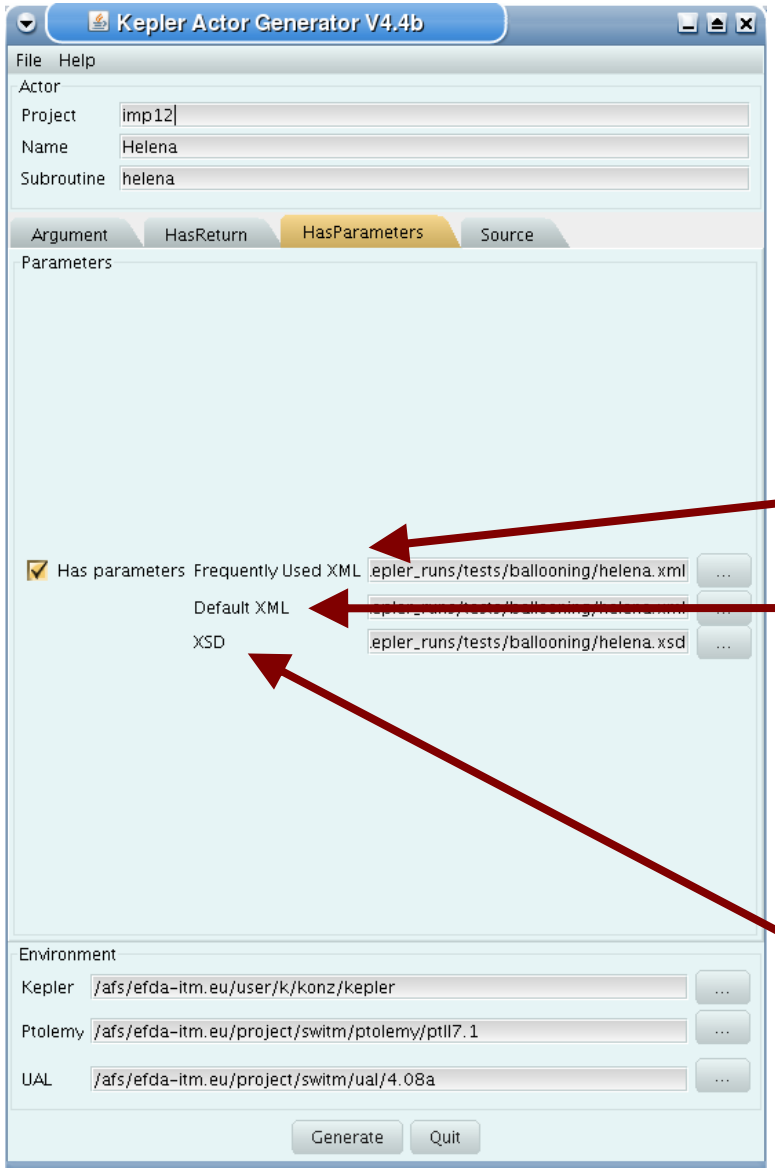
C++

```
struct codeparam {
std::string codename;
std::string codeversion;
std::string parameters;
std::string output_diag;
int output_flag;
} codeparam;
```

type_param

- parameters – actual code parameters (case dependent)
- default_param – default code parameters (physics class dependent)
- schema – W3C XML schema (part of the code distribution, revision dependent)

type_param



Actor generator fc2k to create actor with code parameters (can be changed inside Kepler).

XML input file

XML default parameters

W3C XML Schema

Why XML?

- Extremely versatile markup language
- Self-describing data (through use of DTDs or W3C schemas)
- Simple to edit: plain ASCII
- Handles all levels of complexity
- Large and fast growing user community
- Large infrastructure of tools for XML creation, manipulation, and usage
- Already used for definitions of CPOs
- Allows **separation of generic tools and code specific parameters**

Example: Code Parameters (HELENA)

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="./input_helena.xsl"
charset="ISO-8859-1"?>

<parameters>

<!-- profile parameters -->

 <profile_parameters>
   <hbt> .false. </hbt>
   <input_type> p' and FF' </input_type>
   <radial_coordinate> psi </radial_coordinate>
 </profile_parameters>

<!-- shape parameters -->

 <shape_parameters>
   <isol> 0 </isol>
   <ias> 1 </ias>
   <imesh> 2 </imesh>
   <n_acc_points> 2 </n_acc_points>
   <s_acc> 0.7 1.0 </s_acc>
   <sig> 2.5 0.1 </sig>
   <weights> 0.1 1.0 </weights>
   <equidistant> 0. </equidistant>
 </shape_parameters>


<!-- ... and so on ... -->


<!-- diagnostics parameters -->

 <diagnostics_parameters>
   <verbosity> 4 </verbosity>
   <output> full </output>
   <diagnostics_on> .true. </diagnostics_on>
   <standard_output> .true. </standard_output>
 </diagnostics_parameters>

</parameters>
```

boolean

string

integer

array

W3C XML Schemas

- Each W3C XML schema defines an 'XML language' specific for the code module (no standard would be flexible enough to deal with the wide range of codes and the rapid changes of them)
- Other than DTDs, W3C XML schemas can also constrain the type of data in an element (=> validation of code parameters, eliminates unnecessary code crashes)
- XML schemas are themselves XML documents (well-formedness and validity, reduces risk of typos)
- Definition of a schema allows for the design of generic tools through the separation of the specific structure of the code parameters of a specific module from the development of those tools.
- By defining a W3C XML schema, all code specific information is cast into a single file which is itself an XML string.

Converting Code Parameters into XML

- **Step 1: Extraction – XML Schema**
  - Extract structure of code specific parameters (i.e. names, types, structures, dimensions, allowed choices and ranges, etc.) into W3C XML Schema (tool CREATE_SCHEMA may help with this process)
  - No format specific read routines needed anymore
  - All tools can be made generic
  - All code specific information in one single external file or XML string
  - Creation of the schema is a 'once-in-a-code's-lifetime' event
  - Enables input checking before running the code
  - Schema serves as minimum documentation for input

Converting Code Parameters into XML

- **Step 2: Conversion – XML File**
  - Convert former input files containing the code parameters into XML input files (currently no supporting tool available)
  - Text input files easier to understand by user
  - Same advantage as namelists: input does not have to be complete
  - Free order of input parameters as long as structure is maintained
  - Input checks possible
  - XML can be used for namelist input as well as any other format

Converting Code Parameters into XML

- **Step 3: Assignment Function**
  - Create assignment function which assigns the values from XML input file to the corresponding variables in the code module (no automatism possible in compile languages like Fortran because of lack of introspection)
  - Support tool available: CREATE_ASSIGN (for Fortran90)
  - Generic tools as separate library – easier to maintain
  - GUI development or use of existing GUIs possible
  - Users do not need to know about XML at all
  - Developers need to know only very little about XML

XML Parser

- Many parsers available on the market
- XMLLIB: efficient, fast XML parser in Fortran90 developed by ITM (C. Konz), based on W3C XML Schemas
- ITM C/C++ XML Parser (M. Hoffmann), based on W3C XML Schemas
- Fortran90 XML Parser XML2EQ (E. Giovannozzi), standalone XML documents

Example: Assignment Routine

```
subroutine assign_code_parameters(code_parameters, return_status)

!------------------------------------------------------------------------
! calls the XML parser for the code parameters and assign the
! resulting values to the corresponding variables
!------------------------------------------------------------------------

  use itm_types

!Add the modules hosting the relevant variables here!

  use euitm_schemas
  use euitm_xml_parser

  implicit none

  integer(itm_i4), parameter :: iu6 = 6

  type (type_param) :: code_parameters
  integer(itm_i4), intent(out) :: return_status

  type(tree) :: parameter_list
  type(element), pointer :: temp_pointer
  integer(itm_i4) :: nparm, n_values
  character(len = 132) :: cname

  nparm = 0
  n_values = 0
  return_status = 0      ! no error

!-- parse xml-string code_parameters%parameters using W3C XML schema in
!   code_parameters%schema
  call euitm_xml_parse(code_parameters, nparm, parameter_list)

!-- assign variables

  temp_pointer => parameter_list%first

  outer: do
    cname = char2str(temp_pointer%cname)    ! necessary for AIX
    select case (cname)
      case ("parameters")
        temp_pointer => temp_pointer%child
        cycle
      case ("shot")
        temp_pointer => temp_pointer%child
        cycle
```

call XML parser

Example: Call Assign

Last parameter in list because of FC2K

Copy code parameters into output CPO

Call assignment routine

```
subroutine helena(equilibrium_in, equilibrium_out, in_path, code_parameters)

  use itm_types

! .....

!-- call parameters
  type (type_equilibrium), pointer :: equilibrium_in(:)
  type (type_equilibrium), pointer :: equilibrium_out(:)
  type (type_param) :: code_parameters


! ......

  allocate(equilibrium_out(1))
  allocate(equilibrium_out(1)%codeparam%codename(1))
  allocate(equilibrium_out(1)%codeparam%codeversion(1))
  if (.not. associated(code_parameters%parameters)) then
    write(iu6, *) 'ERROR: code parameters not associated!'
    stop
  else
    allocate(equilibrium_out(1)%codeparam%parameters(size( &
     code_parameters%parameters)))
  end if

!-- add to equilibrium_out
  equilibrium_out(1)%codeparam%codename = codename
  equilibrium_out(1)%codeparam%codeversion = codeversion
  equilibrium_out(1)%codeparam%parameters = code_parameters%parameters

!-- assign code parameters to internal variables
  call helena_assign_code_parameters(code_parameters, return_status)

  if (return_status /= 0) then
    write(iu6, *) 'ERROR: Could not assign code parameters.'
    return
  end if
```

ITM XML Parser

- Compact (~500 lines Fortran90), efficient, fast parser
- Parses XML documents with arbitrary depth and complexity (except for attributes)
- Based on W3C XML Schemas
- Uses tree-like lists with parent, child, and sibling pointers
- Tag names and value lists of arbitrary length (dynamical memory allocation)
- Available as module euitm_xml_parser

ITM XML Parser

- Parses the schema and builds an emptry tree with the structure describe by the schema: associates the corresponding pointers, allocates the tag names cname and fills in the tag names
- Parses the actual XML document and fills the parsed values cvalue into the tree.
- Returns the complete tree in parameter_list and the number of successfully parsed parameters nparm