The XICOOL Preprocessor for ICOOL Input Files

Steve Bracker
s_bracker@hotmail.com

Version 1.0 March 3, 2001
Version 1.1 March 12, 2001 (Mucool Note #0196)

Introduction

More than one person has noted [1] that managing the input files for ICOOL and other similar programs can be quite vexing. Simulation programs tend to be used to "play" with apparatus configurations, and it is not unusual for one person to generate dozens or hundreds of runs over a period of months, at least a few of which may be of more than transient value. Simulation results presented to colleagues lose most of their value if there isn't some concise way to report exactly what apparatus configuration was actually being studied. Often enough, a line of development given up as unpromising is later reinvigorated by new thinking, and it is necessary to go back and resurrect work dropped weeks or months ago.

What can be done to help make ICOOL input files easier to generate, to document, and to revive when needed? Three things come immediately to mind:

1. ICOOL is very limited in its ability to support user comments. Many physicists are none too diligent about commenting their work even when it is easy to do, but ICOOL's data format makes it almost impossible. An ICOOL preprocessor might, as a beginning, provide a format for commentary, and strip comments and blank lines before submitting the input to ICOOL.

2. ICOOL control files are often pages and pages of highly repetitive listings of apparatus components. Many times these components are identical, or trivial variants of one another, which repeat dozens or hundreds of times. Apparatus is often naturally structured into hierarchical subsets: each A contains several similar Bs; each B contains several similar Cs . . . . An obvious way to deal with such situations is to provide a single place to define a piece of apparatus, with a few parameters describing the variations, and a mechanism -- basically multiple levels of parameterized text macros -- to expand the (compact, well-commented) definition into the control file wherever it may be required.

3. ICOOL simulations often have many control files -- the main file (for001.dat) , and several auxiliary files which specify magnetic fields, cavity phases, etc. All of these files are crucial to defining the apparatus, and to "save a configuration" it is necessary to save all the files and the fact of their association. Although there are several ways to accomplish this, by far the easiest is to have all the information appear in one file, and to use a preprocessor to extract segments of the information and write all the necessary control files.

The Basics

File preparation for a simulation begins by creating (with your favorite text editor) a single file, compact and well commented. Expansion of the file removes the commentary, expands the concise description into the much more verbose format required by ICOOL, and generates all of the required control files. To preserve an apparatus configuration, you need only save a single self-documenting file -- the XICOOL input file.

On a Unix machine, a single shell command will produce a list of all such files -- name, date, and the first few lines which (hopefully) summarize the apparatus configuration. This makes it easy to prepare a "catalog" of simulation runs.

The user will usually never have to look at the ICOOL-format control files, any more than he now looks at for007.dat, the apparatus description in ICOOL internal format, but the files are available if you wish to inspect them before starting ICOOL. [2]

Once the input file is prepared -- let's imagine that it is called **ring2d** for version 2d of the cooling ring -- XICOOL is used to prepare the input files for ICOOL. In Unix/Linux, a single command does the job: ***xicool ring2d***. There is no need to specify the names of output files; that information is included in the input file **ring2d**.

As XICOOL runs, it prepares a diagnostic file. Usually, there is no need to look at the diagnostic file, but if there are problems in the XICOOL input file, the diagnostic file can be a valuable aid to finding them. The diagnostic file is named **xicool.dia**, and is an ordinary text file that can be examined with ***more***, a text editor, a printout, etc.

In summary:

Input: ring2d (or whatever)
Output: for001.dat, for030.dat, for031.dat etc. -- the ICOOL input files
Output: xicool.dia (the XICOOL diagnostic file)
Run command: ***xicool ring2d***

This Document . . .

This document is divided into two main sections. In the first, I describe the application of XICOOL to a real problem -- a first-order simulation of the Balbekov Cooling Ring. It's often easier to learn this kind of program by example than by formal description. However, the second section of the document does give a somewhat more formal description of the input formats. As you will see, there isn't really very much to XICOOL; someone familiar with ICOOL and some specific apparatus to be simulated can probably learn XICOOL in less than an hour.

References and Pointers

V. Balbekov, ***Using of a Ring Cooler for MUCOOL Experiment***, December 5, 2000

V. Balbekov, ***Ring Cooler Update***, January 29, 2001, Mucool Note 0189

William F. Fawley, ***Quick User Manual for the NIME Pre-Processor***, February 12, 2001, http://www.cap.bnl.gov/numu/software/nime.html

Fortran (g77) source code for XICOOL: linuxfarm1.phy.olemiss.edu/~opticool/utility/xicool.for
Linux command to compile and link XICOOL: g77 -o xicool xicool.for

Example XICOOL input file: linuxfarm1.phy.olemiss.edu/~opticool/utility/ring
Linux command to run xicool example: xicool ring

My e-mail address: s_bracker@hotmail.com

A "Real" Problem: Setting Up the Balbekov Cooling Ring for ICOOL

See Ring Cooler Update (V. Balbekov, January 29, 2001). For now, we ignore the need for injection and extraction. (I think it's useful to describe XICOOL on something like a real problem, but please remember that this is a software writeup, not physics!)

We begin by defining the entire short straight section as a cell containing five regions -- three vacuum drift spaces and 2 wedges. There are two parameters to SSS: the file number within which the cell's magnetic field is defined (2 choices) and the azimuthal angle of the wedges (2 choices)

Both the drift spaces and the wedges are to be defined within lower level macros. Each has a single parameter; SSS drift spaces may be of various lengths, and SSS wedges may have various azimuthal orientations.

Note that WedgeAngle, a parameter of SSS, is passed through to SSS_Wedge, whereas FieldFile is expanded within SSS. Blank lines and comment lines (! in column 1) are stripped during XICOOL processing. The definition of SSS contains many lower-level macro invocations.

```
! SHORT STRAIGHT SECTION
!   FieldFile:  file number of magnetic field specification (31 or 33)
!   WedgeAngle:  azimuthal angle of the wedge in degrees (235 or 315)
!
! Example:    \SSS  33  315.

\Define  SSS  FieldFile  WedgeAngle

! The Short Straight Section is an ICOOL cell
CELL

! Cell repeat count = 1
1

! Cell field flip flag = false  (don't care)
F

! Cell field type
BLOCK
! Field Model,  File# of field definition file
  2  %FieldFile  0.  0.  0.    0.  0.  0.  0.  0.    0.  0.  0.  0.  0.

! The first region of the cell is a vacuum drift space, dipole to first wedge
\SSS_Drift  1.23

! The second region of the cell is a lithium hydride wedge
\SSS_Wedge  %WedgeAngle

! The third region of the cell is a vacuum drift space, between wedges
\SSS_Drift  2.71

! The fourth region of the cell is the second lithium hydride wedge
\SSS_Wedge  %WedgeAngle

! The fifth and last region of the cell is a vacuum drift space, second wedge to dipole
\SSS_Drift  1.23

\End
```

Next, we must define the drift space and the wedge. Although it is convenient in this case to define them right below the place they are used, it doesn't matter. In particular, they don't need to be defined prior to their invocation within SSS. When the input file is scanned for macro definitions, all of the definitions are extracted and saved; all are available at any point during the macro expansion process which follows.

These are both lowest-level macros; there are no macros invoked within these definitions. First, the drift spaces between the wedges . . .

```
! SHORT STRAIGHT SECTION, VACUUM DRIFT SPACE
!  DriftLength -- Length of the drift space (meters)

\Define  SSS_Drift  DriftLength
SREGION

! Length of region, number of radial subregions, step size (meters)
 %DriftLength  1  0.1

! Radial subregion: subregion #, inner radius, outer radius
 1  0.0  0.23

! Region-specific field and parameters
NONE
 0. 0. 0. 0. 0.   0. 0. 0. 0. 0.   0. 0. 0. 0. 0.

! Material tag (vacuum)
VAC

! Material geometry and parameters
CBLOCK
 0. 0. 0. 0. 0.   0. 0. 0. 0. 0.

\End
```

And here are the wedges:

```
! SHORT STRAIGHT SECTION, WEDGE ABSORBERS
!  WedgeAngle -- Azimuthal angle of the wedge (degrees, 0-360)

\Define  SSS_Wedge  WedgeAngle
SREGION

! Length of region, number of radial subregions, step size (meters)
 n.nn  1  0.01

! Radial subregion: subregion #, inner radius, outer radius
 1  0.0  0.23

! Region-specific field and parameters
NONE
 0. 0. 0. 0. 0.   0. 0. 0. 0. 0.   0. 0. 0. 0. 0.

! Material tag = Lithium Hydride
LIH

! Material geometry and parameters  (ICOOL manual, p. 68)
WEDGE
 n.nn  n.nn  n.nn  n.nn  n.nn     n.nn   0.  0.  0.  0.

\end
```

The Long Straight Section is defined in a similar manner. In some ways, it is even simpler. The LSS contains three regions -- two identical accelerating cavities and one liquid hydrogen absorber. Moreover, at the moment, all accelerating cavitiues in the entire ring are identical. (This may well change in the future, for example to tune the phases, but we'll make sure it's easy to modify the file when the need arises.) All absorbers in the entire ring are also identical. The only difference between one LSS and another is the direction of the solenoidal magnetic field, so for now there is only a single parameter -- the field definition file number.

```
! LONG STRAIGHT SECTION
!   FieldFile:  file number of magnetic field specification (30 or 32)
!
! Example:    \LSS  30

\Define  LSS  FieldFile

! The Long Straight Section is a cell
CELL

! Cell repeat count = 1
1

! Cell field flip flag = false  (don't care)
F

! Cell field type (solenoid surrounding the entire cell)
BLOCK
! Field Model,  File# of field definition file
  2  %FieldFile  0.  0.  0.    0.  0.  0.  0.  0.    0.  0.  0.  0.  0.

! The first region of the cell is an accelerating cavity. We will define it within a
! lower level macro because there will be a second one, identical or nearly so,
! later in the LSS.
\LSS_Cavity

! The second region of the cell is a liquid hydrogen absorber. We will specify it
! right here since we anticipate that every such absorber in the whole ring will
! be identical.

SREGION

! Length of region, number of radial subregions, step size (meters)
 n.nn  1  0.01

! Radial subregion: subregion #, inner radius, outer radius
 1  0.0  0.74

! Region-specific field and parameters
NONE
 0. 0. 0. 0. 0.   0. 0. 0. 0. 0.   0. 0. 0. 0. 0.

! Material tag = Liquid Hydrogen
LH

! Material geometry and parameters
CBLOCK
 0.  0.  0.  0.  0.    0.  0.  0.  0.  0.

! The third region of the cell is an accelerating cavity.
\LSS_Cavity

\End
```

We still have to define the accelerating cavity. For now it has no parameters, but it may easily be given some as needed later.

```
! ACCELERATING CAVITY
!  No parameters
! Example:   \AccelCav

\Define AccelCav

SREGION

   <etc. etc. etc.>

\End
```

We're almost done with components for the entire ring. The only other component we need to worry about right now is the 45-degree bending dipole. I suspect that eventually there is going to be a background field associated with dipoles here that will motivate having two or even four subspecies of dipoles, but we'll let that be for now.

```
! 45 DEGREE BENDING DIPOLE
! For now, no parameters
! Example:  \Dipole

\Define Dipole

SREGION

  <etc. etc. etc.>

\End
```

We have defined all the basic components; it remains only to put them together. The minimum almost-repeatable unit of the ring is the quarter turn. There must be provision for flipping the solenoid field of both the LSS and the SSS, and changing the azimuthal angle of the wedges in the SSS. Starting with a long straight section:

```
\Define  QuarterTurn  LSSField  SSSField  WedgeAngle

! Long Straight Section
\LSS %LSSField

! A 45 degree bending dipole
\DIPOLE

! Short Straight Section
\SSS  %SSSField  % WedgeAngle

! A 45 degree bending dipole
\DIPOLE

\End
```

A half-turn is made up of two similar but not identical quarter turns:

```
\Define  HalfTurn

!  Quarter turn with LSS field 30, SSS field 31, Wedge angle 225 degrees
\QuarterTurn  30  31  225.

!  Quarter turn with LSS field 32, SSS field 33, Wedge angle 315 degrees
\QuarterTurn  32  33  315.

\End
```

Next, a few convenience macros for running particles once around the whole ring, five times around, etc.

```
! Propagate particles around one full turn of the ring.
\Define FullTurn
\HalfTurn
\HalfTurn
\End
```

```
! Propagate particles five times around the ring.
\Define FiveTurns
\FullTurn
\FullTurn
\FullTurn
\FullTurn
\FullTurn
\End
```

For sending particles around the ring ten times, the following will generate the for001.dat file. Note the \file command. It says "after you have stripped comments and blank lines, removed macro definitions, and expanded all the macros at all levels, then take what's between here and the next \file command (or the end of the file) and write it into a file named for001.dat.

```
\File  for001.dat

! Document the control variables . . .
$cont     <a few control variables>  $

! Document the beam generation specification
$bmt      <a few beam definition variables> $

! Document the interaction model specification
$ints     <define interaction model> $

$nhs      <define histograms> $
  <define scatter plots, z-histories, r-histories, emittance calculations, etc.>

! Apparatus Specification
SECTION
\FiveTurns
\FiveTurns
ENDSECTION
```

However, we have four additional files which must be prepared. All four are short magnetic-field definition files, which generate straight-section solenoid fields using current blocks. There are two LSS fields (files 30 and 32) and two SSS fields (files 31 and 33) which are (at the moment) trivial modifications of each other; the difference is characterized by a single number, either +1 or -1.

We need two macros to generate the fields:

```
\Define LSS_Field  FieldSign

  < text to define a current-block field with %FieldSign determining the field direction>

\End

\Define SSS_Field  FieldSign

  < text to define a current-block field with %FieldSign determining the field direction>

\End
```

Then to produce the four magnetic field definition files, we specify:

```
\File  for030.dat
\LSS_Field  1.

\File  for032.dat
\LSS_Field  -1.

\File  for031.dat
\SSS_Field  1.

\File  for033.dat
\SSS_Field  -1.
```

General Description of XICOOL

XICOOL accepts a single user input file. It strips comments and blank lines from the file, expands multiple levels of text macros, and generates one or more output files ready for use as input to ICOOL.

In Unix/Linux, XICOOL is invoked by the following command line: ***xicool <inputfilename>***. No user interaction is expected. A diagnostic file is prepared, but normally it is not necessary to examine it.

The Major Processing Steps

1. Read the user-specified input file and strip out comments (all lines whose first non-blank character is !) and all blank lines. The stripped output file becomes the input to the next step.

2. Read the comment-and-blank-line-stripped input file and extract the macro definitions. Build data structures in memory that preserve the macro definitions, but strip the definitions from the output file. The output file, now stripped of comment lines, blank lines and macro definitions, becomes the input to the next step.

3. Read the input file produced in step 2, and copy it line-by-line into the output file. Whenever a macro invocation is found in the input file, use the macro definitions stored in memory to replace the invocation line with the specified text, substituting values for macro parameters as required. If a macro definition includes a macro invocation [4], just move that line into the output file just as you would for ordinary text, but keep a count the number of macro invocation lines copied. The output file becomes the input file for the next step. If the number of macro invocations copied into the output file is zero, go to step 4. Otherwise repeat step 3.

4. As specified by the \File commands, write the segments of the stripped, macro-expanded input file into the specified output files. Close all the files and exit.

The \File directive

\File filename

Once stripping and expanding have been completed, put everything from the \file statement to the next \file statement (or end-of-file) into a file with the specified name, overwriting anything that is already in a file by that name. For ICOOL, typical filenames are for001.dat, for030.dat, etc.

I hope it's clear that one should not name a macro "File", nor should one name an output file so as to overwrite the input file or anything else you aren't prepared to lose. My file convention is to regard the ICOOL for0xx.dat files as temporary files, and to store the apparatus configuration data in XICOOL input files with more mnemonic names, e.g. BalbekovWedgeRotationTest03.

The \Define directive and the \End directive -- Part 1

\Define MacroName ParamName1 ParamName2 . . .
. . . . . .
. . . . . .
\End

Defines a macro -- text which replaces an input line ***invoking*** the macro. A macro may have 0-16 parameters. If a component appears at several places within the apparatus, without any modification, a no-

parameter macro will suffice. If minor variants of a component appear at several places within the apparatus, then a parameterized macro will be useful; the parameters are used to characterize the variants.

Every \Define directive must be followed by an \End directive. The intervening lines specify the text to be used to replace the macro invocation line whenever it occurs.

When a macro is invoked, values must be specified for the parameters. The macro definition may include zero or more ***substitution markers*** for each parameter. During macro expansion, the value of each substitution marker is replaced by the parameter's value. A substitution marker for a parameter consists of % followed immediately (no space) by the parameter's name.

Example (noteworthy items highlighted):

```
\Define Absorber  Length  Material
SREGION

!  Length as specified;  1 radial subregion;  1 cm stepsize
%Length  1  0.01

!  Radial region #1;  inner radius 0 meters;  outer radius 0.70 meters
1  0.00  0.70

!  No region-specific field
NONE
0. 0. 0. 0. 0.    0. 0. 0. 0. 0.    0. 0. 0. 0. 0.

! Material as specified
%Material

!Material geometry is a simple cylindrical block
CBLOCK
0. 0. 0. 0. 0.    0. 0. 0. 0. 0.

\End
```

Invocation of a Macro

\MacroName ParamValue1 ParamValue2 . . .

For the absorber macro defined above - -

To invoke an absorber with 1.2 meters of liquid hydrogen:

```
\Absorber   1.2  LH
```

To invoke an absorber with 14 cm of lithium hydride:

```
\Absorber  0.14  LIH
```

A macro invocation consists of a line of text. It begins with \ in column 1 followed immediately by the name of the macro. Following that are the values of the parameters, in the same order they appear in the macro definition. Default values for parameters (and omission of explicit values assigned to each parameter) are not permitted. [3]

The \Define directive and the \End directive -- Part 2

Macro definitions may (and often do) include only ordinary text with or without parameter substitution. Such macros are known as lowest-level macros.

However, macro definitions may also include macro invocation lines. Expansion of the macro now being defined can trigger expansion of other macros, which can trigger expansion of yet other macros . . . until the chain of expansions finally ends (hopefully!) in lowest-level macros. The real power of XICOOL lies in its ability to map multi-level apparatus hierarchies into multi-level macro expansions. The Balbekov Ring example above contains many instances.

However, when multi-level macros are expanded, one must decide how to handle parameter passing. Within a given macro, parameters for nested macro invocations may be either constants (4.567 or LIH) or may be substitution markers for the upper-level macro's own parameters. The latter permits a macro to accept a parameter value and pass it on to any macros that it invokes.

Suppose that the Long Straight section (LSS) cell contains an absorber, and that the LSS cell itself is specified by a macro named LSS. Then LSS can invoke the absorber with constant parameters:

```
\Define  LSS
 . . . . . .
 . . . . . .
\Absorber   1.2  LH
 . . . . . .
 . . . . . .
\End
```

But suppose there were two different kinds of Long Straight Section with different absorber lengths (but always liquid hydrogen). Then one could define LSS with an absorber length parameter, and pass its value to the Absorber macro it invokes:

```
\Define LSS AbsorberLength
 . . . . . .
 . . . . . .
\Absorber   %AbsorberLength   LH
 . . . . . .
 . . . . . .
\End
```

Then \LSS 1.2 would specify a Long Straight Section that includes a 1.2 meter long absorber, while \LSS 0.8 would specify a Long Straight Section with a shorter absorber.

SomeBasic Principles for a Good XICOOL Data File

If a significant block of text (more than a line or two) will appear at two or more places in the output file, especially if there is any prospect that the block of text will be changed in the future, make a macro of it.

If minor variants of a significant block of text will appear in the output file, make parameters of the variant portions and define the whole block as a parameterized macro.

Try to set up the file in such a way that a design change that involves altering one operating parameter requires that parameter to be edited at only one place in the text.

Make macro names and parameter names as mnemonic as reasonably possible.

Don't worry about execution time; input file preparation will always be negligible compared to any significant simulation task.

Comment exhaustively, and at every level. Explain what apparatus each macro represents and what design variable each parameter represents, including units.

Strategies

Given a problem, how do you get started? There are several strategies that will probably work well. I would start with the apparatus definition section. One choice is to begin at the top with one or a very few high-level macros that specify the entire apparatus, and then work your way down, filling in the details at each step as you go. Another choice is to start at the bottom, writing detailed implementations of each type of elementary component, and then building them into more complex subsystems.

The apparatus definition goes into for001.dat, so start a \file section for it and insert the apparatus specification into it. For now, don't worry about all the stuff that goes at the beginning of for001.dat. When you're far enough into the apparatus specification to do so, make a list of all the auxiliary files that you'll need, and create \file specifications for them. It is natural to work on the details of the apparatus definition and the auxiliary files together. When that's all done, then look after the control variables, the beam generation, the histogram definitions etc. Remember that although XICOOL macros were really made for apparatus definitions, they will do just as well for groups of similar histogram definitions, etc.

However you proceed, test often. Put in dummy component output lines and check the XICOOL output files. For small modifications to an established and tested configuration, you won't need to do much checking. At the moment XICOOL is not very forthcoming with error information, and when troubles appear it's just as well not to have 200 lines of changes to examine.

Notes:

[1] In a classic case of convergent evolution, I was putting the finishing touches on this XICOOL writeup when I received e-mail announcing William Fawley's NIME. NIME and XICOOL don't do exactly the same things, and their implementations are quite different, but they address many of the same basic needs. XICOOL was originally written to stand between OPTICOOL and ICOOL. OPTICOOL generates apparatus configurations without human interactivity, so XICOOL must be non-interactive as well. XICOOL would normally be called as a subroutine within OPTICOOL (though the version described here is a stand-alone program); hence it was most natural to write it in g77 Fortran, the same as OPTICOOL and our version of ICOOL (a trivial modification of ICOOL for Windows).

[2] I strongly discourage editing the XICOOL output files (the ICOOL input files) directly, just as I would discourage anyone from modifying the executable of a compiled program rather than changing the source code and recompiling. By doing so, one sacrifices the documentary value of the XICOOL input file. Expanding even a very large XICOOL input file takes a small fraction of a second, so there is little reason to change the output files by any means other than changing the input file. However, for learning XICOOL and for checking your work, *examining* the XICOOL output files is never a bad idea.

[3] I have a strong personal antipathy to default values for parameters; some of my hardest-to-find blunders have involved defaulted parameters where I either misremembered the default value or forgot to override a default. Of course opinions may vary, but in XICOOL defaults are not allowed; every parameter must be given an explicit value.

[4] A macro definition can include macro definitions, to any level. (It may be well to impose a cutoff just to avoid catastrophe if the user accidentally calls for a macro to be expanded within its own definition, or specifies a circular expansion.)

! Don't let this self-calling macro definition expand forever.

```
\define  SelfCalled
- - - - - - - -
- - - - - - - -
\SelfCalled
- - - - - - - -
- - - - - - - -
\end

\SelfCalled
```

! Don't let this circular macro expansion expand forever either.

```
\define  A
- - - - - - - -
- - - - - - - -
\B
- - - - - - - -
- - - - - - - -
\end

\define B
- - - - - - - -
- - - - - - - -
\A
- - - - - - - -
- - - - - - - -
\end

\A
```

In the end, of course, it should be the case that every multi-level macro expansion ends up expressed in terms of lowest-level macros -- those that produce pure text and call for no additional macro expansions.