European Spallation Source

LinacLego

XML based Lattice Description

David McGinnis
7/8/2014

Contents

Introduction

LinacLego was originally developed for the 2013 ESS Linac. LinacLego is an XML based system for describing a linear accelerator lattice in an organized hierarchal manner. The hierarchal structure of LinacLego can be used to develop a product break-down structure of the linac. The XML[1] schema makes a complicated structure readily human readable as well as providing a flexible input platform for computer simulations, graphical design programs, and databases.

Motivation

Linear accelerator lattice designers use tools such as TraceWin, Parmilla, Dynac, etc. to simulate the beam response to the design. Once the lattice designer is happy with the lattice design, the lattice design must be communicated to engineers who must transform the idealized elements such as drifts, quadrupoles, bends, etc, used in the simulation into physical engineered components such as flanges, pipes, manifolds, etc. The input files to simulation programs are flat sequential ASCII text files which for designs with a few number of components are easily human readable. However, as the design becomes larger and more complex, these files get very difficult for a human to read, hence error-prone.

The situation worsens for designs containing large numbers of repeating cells in which only a few parameters such as quadrupole gradients or cavity fields change from cell to cell. The lattice designer must copy large blocks of flat text containing many parameters over and over in the file. If the designer then wants to change a component value such as a drift length in a cell, he must then find each instance of this component in all the cells to make this change. This can often result in errors.

Finally, the component engineers would like to know how many types of elements he needs to construct and where these elements are located. Since it is difficult to annotate the lattice design in the lattice input files, it is difficult to compile the required data with implement some type of ad-hoc program specific, comment format in the lattice input files.

Using and XML structure to describe the lattice instead of flat files makes it easy to group and organize the lattice so that humans can understand the lattice structure. The XML format also readily permits the easy implementation of XML models that can be used to describe repeating structures in which only a few parameters vary from structure to structure. Since the XML format is machine readable, data queries are straightforward to implement.

Workflow

The importance of not burdening the lattice designer with new design syntax cannot be overstated. With this in mind, LinacLego follows a well-defined workflow.

[1] http://docs.oracle.com/javaee/1.4/tutorial/doc/IntroXML2.html

1. LinacLego takes the flat ASCII lattice input file, of which the only format currently implemented is TraceWin, from the lattice designer and converts it to an XML format following an appropriate XML schema.
2. LinacLego then provides tools that aid the accelerator engineers into organizing the XML description of the lattice into a hierarchical form.
3. As a check of the conversion integrity, LinacLego returns a flat ASCII lattice input file, of which the only formats currently implemented are TraceWin and Dynac.
4. LinacLego provides a PBS tree view of the lattice in addition to comma separated value (.csv) reports.

LinacLego XML Structure

The main body of a linacLego is a linac.

- A linac is comprised of sections. For example, in the ESS Linac example of sections are the Spoke section, the Medium Beta section, etc.
- A section is comprised of cells. The boundaries of the cells can be arbitrary but is intended as a grouping of components inside a lattice focusing period such as a FODO cell.
- A cell is comprised of slots. Again, the definition of slot boundaries is arbitrary but slots are intended to represent an engineering assembly such as a cryomodule.
- A slot is comprised of beam-line elements which is the fundamental building block of LinacLego. The beam-line elements have a one-to-one correspondence with the elements in the ASCII lattice input file. Examples of beam-line elements are drifts, quadrupoles, bends, RF cavities, thin steerers, etc.

This hierarchy is shown schematically in XML in Figure 1:

```
<linac>
     <section id="sectionA" >
          <cell id="cellA" >
               <slot id="slotA" >
                    <ble id="driftA" type="drift" />
                    <ble id="quadB" type="quad" />
               </slot>
               <slot id="slotB">…</slot>
          </cell>
          <cell id="cellB">…</cell>
     </section>
     <section id="sectionB" >
          <cell id="cellA">…</cell>
     </section>
</linac>
```

Figure 1. Hierarch of LinacLego XML

Each element has a user-defined identifier or *id* attribute. While a single identifier does not have to be unique, the collection of identifier attributes should give a unique id. The quadrupole in the preceding XML example would have the composite id of:

sectionA-cellA-slotA-quadB

This composite id forms a complete address of where the beam-line element is located in the linac. The XML description above can form a complete description of where a linac component is located and the physical properties of the component. However, it does not describe how the linac is controlled. The interface with control system is done with Control Points that will be discussed later.

LinacLego XML Components

In LinacLego, the linac is described using XML syntax. The structure of the XML file is shown in Figure 2.

```
<linacLego title="linacLegoTitle" revNo="1" >
     <header>
          <slotModels id="slotModelsId">
          <cellModels id="cellModelsId">
          <controlPoints id="controlPointsId">
     </header>
     <linac>
          <linacData>
          <section id="sectionIdA" rfHarmonic="n">
                          .
                          .
          <section id="sectionIdZ" rfHarmonic="n">
     </linac>
</linacLego>
```

Figure 2. LinacLego XML file structure.

The dictionary document (.dtd) describing the LinacLego XML file structure can be found in Appendix C. LinacLego.dtd. The descriptions of LinacLego XML tags are described in the following sections.

Data Element *<d>*

Lattice files contain a large number of parameters. Usually no description of what these parameters represent or their units can be found unless the user looks in the user manual for the description. Parameters in LinacLego are described in a Data Element tag that provides description about the parameter. A Data Element <*d*> tag describing a drift length is shown in Figure 3.

```
<d id="l" type="double" unit="mm">50.0</d>
```

Figure 3. Data Element tag example

where the *id* attribute contains a user defined name of what the data is called. The *type* attribute can take on the values double, int, string, and boolean. The unit attribute is takes on what is required by the element using the <*d*> tag. The value is contained in character data bracketed inside the <*d*> tag.

Beam-line Element *<ble>*

The fundamental block of LinacLego is the beam-line element *<ble>* tag. The beam-line elements have a one-to-one correspondence with the elements in the ASCII lattice input file. Examples of beam-line elements are drifts, quadrupoles, bends, RF cavities, thin steerers, etc. Each beam-line element has a single beam-line element as its upstream neighbor and a single beam-line element as its downstream neighbor. The *<ble>* tag includes attribute an *id* attribute that is up to the user to define and is required. There is also a required type attribute that is used by LinacLego as to which beam-line element Java class is implemented. An optional model attribute can be defined by the user to give more distinction to the beam-line element for sorting. Inside the *<ble>* tag is a list of *<d>* tags that are used as parameters describing the beam line element. A *<ble>* tag describing a drift with a length of 50mm and a radius in of 18.4 mm is described in Figure 4.

```
<ble id="DR10" type="drift" model="mebtShortDrift" >
     <d id="l" type="double" unit="mm">50</d>
     <d id="r" type="double" unit="mm">18.4</d>
     <d id="ry" type="double" unit="mm">0</d>
</ble>
```

Figure 4. Beam-line Element tag example.

The formats of other *<ble>* tags are found in Appendix A.

Slot <slot>

A slot is a collection of beam-line elements that makes up an engineering assembly. Only beam-line elements can be contained in a slot. Slots cannot be nested. For example, a cryomodule that contains a sequence of drifts and cavities is shown in Figure 5.

```
<slot id="slot020">
      <ble id="DR10" type="drift" >
            <d id="l" type="double" unit="mm">368.5</d>
            <d id="r" type="double" unit="mm">28</d>
            <d id="ry" type="double" unit="mm">0</d>
      </ble>
      <ble id="FM0020" model="spokeCavity" type="fieldMap">
            <d id="rfpdeg" type="double" unit="deg">14.5</d>
            <d id="xelmax" type="double" unit="unit">0.93</d>
            <d id="radiusmm" type="double" unit="mm">28</d>
            <d id="lengthmm" type="double" unit="mm">994</d>
            <d id="file" type="string" unit="unit">Spoke_F2F</d>
            <d id="scaleFactor" type="double" unit="unit">1</d>
      </ble>
      <ble id="DR30" type="drift" >
            <d id="l" type="double" unit="mm">135</d>
            <d id="r" type="double" unit="mm">28</d>
            <d id="ry" type="double" unit="mm">0</d>
      </ble>
      <ble id="FM0040" model="spokeCavity" type="fieldMap">
            <d id="rfpdeg" type="double" unit="deg">23.1</d>
            <d id="xelmax" type="double" unit="unit">0.98</d>
            <d id="radiusmm" type="double" unit="mm">28</d>
            <d id="lengthmm" type="double" unit="mm">994</d>
            <d id="file" type="string" unit="unit">Spoke_F2F</d>
            <d id="scaleFactor" type="double" unit="unit">1</d>
      </ble>
      <ble id="DR50" type="drift" >
            <d id="l" type="double" unit="mm">368.5</d>
            <d id="r" type="double" unit="mm">28</d>
            <d id="ry" type="double" unit="mm">0</d>
      </ble>
</slot>
```

Figure 5. Slot tag example

Slot Model <slotModel>

In many cases, the same slot structure is used many times over in the lattice file with only a few parameters that change from instance to instance. For example in the slot example shown in Figure 5, the only parameters that will change are the cavity phase (rfpdeg) and the cavity gradient (xelmax). A *<slotModel>* tag is a template for a slot in which some parameters are marked as variable. Shown in Figure 6 is a *<slotModel>* tag that can be used to describe the *<slot>* tag detailed in Figure 5. In the *<slotModel>* tag shown in Figure 6, there are a number of *<var>* tags that are used to define variables. The *<var>* tag has an *id* attribute that gives a name to the variable and a *type* attribute. Once the variable is defined with a *<var>* tag, it can be used in the character data fields enclosed in the *<data>* tag of a parent *<ble>* tag.

```
<slotModel id="spokeCryomodule">
      <var id="xelmax1" type="double"/>
      <var id="xelmax2" type="double"/>
      <var id="rfpdeg1" type="double"/>
      <var id="rfpdeg2" type="double"/>
      <ble id="DR10" type="drift" >
            <d id="l" type="double" unit="mm">368.5</d>
            <d id="r" type="double" unit="mm">28</d>
            <d id="ry" type="double" unit="mm">0</d>
      </ble>
      <ble id="FM0020" model="spokeCavity" type="fieldMap">
            <d id="rfpdeg" type="double" unit="deg">rfpdeg1</d>
            <d id="xelmax" type="double" unit="unit">xelmax1</d>
            <d id="radiusmm" type="double" unit="mm">28</d>
            <d id="lengthmm" type="double" unit="mm">994</d>
            <d id="file" type="string" unit="unit">Spoke_F2F</d>
            <d id="scaleFactor" type="double" unit="unit">1</d>
      </ble>
      <ble id="DR30" type="drift" >
            <d id="l" type="double" unit="mm">135</d>
            <d id="r" type="double" unit="mm">28</d>
            <d id="ry" type="double" unit="mm">0</d>
      </ble>
      <ble id="FM0040" model="spokeCavity" type="fieldMap">
            <d id="rfpdeg" type="double" unit="deg">rfpdeg2</d>
            <d id="xelmax" type="double" unit="unit"> xelmax2</d>
            <d id="radiusmm" type="double" unit="mm">28</d>
            <d id="lengthmm" type="double" unit="mm">994</d>
            <d id="file" type="string" unit="unit">Spoke_F2F</d>
            <d id="scaleFactor" type="double" unit="unit">1</d>
      </ble>
      <ble id="DR50" type="drift" >
            <d id="l" type="double" unit="mm">368.5</d>
            <d id="r" type="double" unit="mm">28</d>
            <d id="ry" type="double" unit="mm">0</d>
      </ble>
</slotModel>
```

Figure 6. Slot Model tag example.

The *<slotModel>* tag is placed inside the *<slotModels>* tag in the *<header>* tag in the beginning of XML file. Then the *<slotModel>* tag is referenced by a *<slot>* tag as shown in Figure 7.

```
<slot id="slot020" model="spokeCryomodule">
      <d id="xelmax1" type="double">0.93</d>
      <d id="xelmax2" type="double">0.98</d>
      <d id="rfpdeg1" type="double">14.5</d>
      <d id="rfpdeg2" type="double">23.1</d>
</slot>
```

Figure 7. Example of a Slot tag using a Slot Model

Cell <cell>

A cell is comprised of slots. The boundaries of the cells can be arbitrary but is intended as a grouping of components inside a lattice focusing period such as a FODO cell. Figure 8 shows an example of a cell composed of two slots. In this example the slots are described by slot models.

```
<cell id=”cell020” >
      <slot id="slot010" model="spokeLwu">
            <d id="g1" type="double">4.53</d>
            <d id="g2" type="double">4.67</d>
      </slot>
      <slot id="slot020" model="spokeCryomodule">
            <d id="xelmax1" type="double">1.0</d>
            <d id="xelmax2" type="double">1.0</d>
            <d id="rfpdeg1" type="double">14.5</d>
            <d id="rfpdeg2" type="double">23.1</d>
      </slot>
</cell>
```

Figure 8. Cell tag example.

Cell Model <cellModel>

In many cases, the same cell structure is used many times over in the lattice file with only a few parameters that change from instance to instance. For example in the cell example shown in Figure 8, the only parameters that might change are the quad gradients and the cavity phase (rfpdeg). A *<cellModel>* tag is a template for a cell in which some parameters are marked as variable. Shown in Figure 9 is a *<cellModel>* tag that can be used to describe the *<cell>* tag detailed in Figure 8. In the *<cellModel>* tag, there are a number of *<var>* tags that are used to define variables. The *<var>* tag has an *id* attribute that gives a name to the variable and a *type* attribute. Once the variable is defined with a *<var>* tag, it can be used in the character data fields enclosed in the *<data>* tag of a parent *<slot>* tag.

```
<cellModel id=”spokeCell” >
      <var id="gradA" type="double"/>
      <var id="gradB" type="double"/>
      <var id="phase1" type="double"/>
      <var id="phase2" type="double"/>
      <slot id="slot010" model="spokeLwu">
            <d id="g1" type="double">gradA</d>
            <d id="g2" type="double">gradB</d>
      </slot>
      <slot id="slot020" model="spokeCryomodule">
            <d id="xelmax1" type="double">1.0</d>
            <d id="xelmax2" type="double">1.0</d>
            <d id="rfpdeg1" type="double">phase1</d>
            <d id="rfpdeg2" type="double">phase2</d>
      </slot>
</cellModel>
```

Figure 9. Example of a Cell Model Tag

The *<cellModel>* tag is placed inside the *<cellModels>* tag in the *<header>* tag in the beginning of XML file. Then the *<cellModel>* tag is referenced by a *<cell>* tag as shown in Figure 10.

```
<cell id="cell020" model="spokeCell">
      <d id="gradA " type="double">4.53</d>
      <d id="gradB " type="double">4.67</d>
      <d id="gradA" type="double">14.5</d>
      <d id="gradB" type="double">23.1</d>
</cell>
```

Figure 10. Example of a Cell tag using a Cell Model

Section <section>

A linac is comprised of sections. For example, in the ESS Linac example of sections are the Spoke section, the Medium Beta section, etc. The *<section>* tag is used to describe the list of sections comprising the linac. There must be at least one *<section>* tag inside the *<linac>* tag. The *<section>* tags must be listed in the order of beam direction. The *<section>*tag includes required *id* attribute that is up to the user to define. There is also a required *rfHarmonic* attribute which is used to set the RF frequency for the section. Figure 11 shows an example of a *<section>* tag that contains two *<cell>* tags

```
<section id="spoke" rfHarmonic="1">
      <cell id="cell020" model="spokeCell">
            <d id="gradA " type="double">4.53</d>
            <d id="gradB " type="double">4.67</d>
            <d id="gradA" type="double">14.5</d>
            <d id="gradB" type="double">23.1</d>
      </cell>
      <cell id="cell030" model="spokeCell">
            <d id="gradA " type="double">3.12</d>
            <d id="gradB " type="double">3.26</d>
            <d id="gradA" type="double">30.1</d>
            <d id="gradB" type="double">37.6</d>
      </cell>
</section>
```

Figure 11. Example of a Section tag

Linac <linac>

The *<linac>* tag contains two types of tags. The first type is the *<linacData>* tag of which there is only one instance. The second type is *<section>* tags of which there must be at least one *<section>* tag. The *<linacData>* tag contains information about the initial conditions of the linac. These conditions can be omitted and if omitted will be assumed to be zero. Figure 12 shows an example of a <linac> tag with a <linacData> tag and a <section> tag.

```
<linac>
     <linacData>
          <d id="ekin" type="double" unit="MeV">3.6</d>
          <d id="beamFrequency" type="double" unit="MHz">352.21</d>
          <d id="xSurvey" type="double" unit="m">0.0</d>
          <d id="ySurvey" type="double" unit="m">-4.5</d>
          <d id="zSurvey" type="double" unit="m">-598.3</d>
          <d id="pitchSurvey" type="double" unit="deg">0.0</d>
          <d id="rollSurvey" type="double" unit="deg">0.0</d>
          <d id="yawSurvey" type="double" unit="deg">0.0</d>
          <d id="alphaX" type="double" unit="unit">-0.054373</d>
          <d id="betaX" type="double" unit="mm/mrad">0.21108</d>
          <d id="emitX" type="double" unit="mm-mrad">11.469</d>
          <d id="alphaY" type="double" unit="unit">-0.31096</d>
          <d id="betaY" type="double" unit="mm/mrad">0.37123</d>
          <d id="emitY" type="double" unit="mm-mrad">11.423</d>
          <d id="alphaZ" type="double" unit="unit">0.4799</d>
          <d id="betaZ" type="double" unit="deg/keV">0.61397</d>
          <d id="emitZ" type="double" unit="deg-keV">573.227</d>
          <d id="beamCurrent" type="double" unit="mA">62.5</d>
     </linacData>
     <section id="spoke" rfHarmonic="1">
          <cell id="cell020" model="spokeCell">
               <d id="gradA " type="double">4.53</d>
               <d id="gradB " type="double">4.67</d>
               <d id="gradA" type="double">14.5</d>
               <d id="gradB" type="double">23.1</d>
          </cell>
          <cell id="cell030" model="spokeCell">
               <d id="gradA " type="double">3.12</d>
               <d id="gradB " type="double">3.26</d>
               <d id="gradA" type="double">30.1</d>
               <d id="gradB" type="double">37.6</d>
          </cell>
     </section>
</linac>
```

Figure 12. Example of a linac tag.

Control Point <cnpt>

Control points are LinacLego tags that describe how the lattice is to be interfaced with the control system. A control point is attached to a beam-line element. There can be more than one control point attached to a beam-line element. The attributes of the *<cnpt>* tag contain the section-cell-slot-ble address of the beam-line element the control point is attached to. To describe the control point, there are the type, model, and id attributes that are up to the user to define. Inside the *<cnpt>* tag are three data tags to describe the x,y, and z position of the control point relative to the beam-line element. This can be useful for describing control points that occupy the same space as the parent beam-line element such as bpm's that are placed inside a quadrupole or cavity couplers. The format of a control point tag is shown in Figure 13.

Control Points <controlPoints>

The list of *<cnpt>* tags is placed inside a *<controlPoints>* tag. There can be more than one *<controlPoints>* tag per file. Multiple *<controlPoints>* tag can be useful to separate the *<cnpt>* tags of different sections of the linac for example. The *<controlPoints>* tags are placed inside the *<header>* tag as shown in Figure 2.

```
<controlPoints id="spokeControlPoints">
      <cnpt section="SPOK" cell="cell010" slot="slot010" ble="QD0020"
            devName="SWU01-BMD:QH-010"  type="bmd" model="spokeQuadPs" id="PS" >
            <d id="dxmm" unit="mm">0</d>
            <d id="dymm" unit="mm">0</d>
            <d id="dzmm" unit="mm">0</d>
      </cnpt>
      <cnpt section="SPOK" cell="cell010" slot="slot010" ble="QD0020"
            devName="SWU01-PBI:BPM-010"  type="pbi" model="spokeBPM" id="BPM" >
            <d id="dxmm" unit="mm">0</d>
            <d id="dymm" unit="mm">0</d>
            <d id="dzmm" unit="mm">0</d>
      </cnpt>
      <cnpt section="SPOK" cell="cell010" slot="slot010" ble="QD0040"
            devName="SWU02-BMD:QV-030"  type="bmd" model="spokeQuadPs" id="PS" >
            <d id="dxmm" unit="mm">0</d>
            <d id="dymm" unit="mm">0</d>
            <d id="dzmm" unit="mm">0</d>
      </cnpt>
</controlPoints>
<controlPoints id="otherControlPoints">
            .
            .
</controlPoints>
```

Figure 13. Example of a Control Points and Control Point tag.

XML Include Files

Because of the hierarchical structure of the LinacLego XML file, the XML file can be separated into sub-files that describe various sections of the linac using *<xi:include>* tags[2]. Breaking the file up into smaller sub-sections makes it easier for a team authoring. Figure 14 shows a collapsed view of the XML file describing the spoke linac. Figure 15 shows how files describing sections of the linac can be included in a larger file.

Figure 14. Example XML file of the Spoke section of the ESS Linac in collapsed view

[2] http://msdn.microsoft.com/en-us/library/aa302291.aspx

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE linacLego SYSTEM "LinacLego.dtd" >
<linacLego title="MEBT Linac" xmlns:xi="http://www.w3.org/2001/XInclude" revNo="1">
    <header>
        <!--SlotModels-->
        <xi:include href="spoke.xml"              parse="xml" xpointer="element(/1/1/1)" />
        <xi:include href="medBeta.xml"            parse="xml" xpointer="element(/1/1/1)" />
        <xi:include href="highBeta.xml"           parse="xml" xpointer="element(/1/1/1)" />
        <!--CellModels-->
        <xi:include href="spoke.xml"              parse="xml" xpointer="element(/1/1/2)" />
        <xi:include href="medBeta.xml"            parse="xml" xpointer="element(/1/1/2)" />
        <xi:include href="highBeta.xml"           parse="xml" xpointer="element(/1/1/2)" />
        <!--ControlPoints-->
        <xi:include href="spoke.xml"              parse="xml" xpointer="element(/1/1/3)" />
        <xi:include href="medBeta.xml"            parse="xml" xpointer="element(/1/1/3)" />
        <xi:include href="highBeta.xml"           parse="xml" xpointer="element(/1/1/3)" />
    </header>
    <linac >
        <linacData>
            <d id="ekin" type="double" unit="MeV">3.6</d>
            <d id="beamFrequency" type="double" unit="MHz">352.21</d>
            <d id="xSurvey" type="double" unit="m">0.0</d>
            <d id="ySurvey" type="double" unit="m">-4.5</d>
            <d id="zSurvey" type="double" unit="m">-598.3</d>
            <d id="pitchSurvey" type="double" unit="deg">0.0</d>
            <d id="rollSurvey" type="double" unit="deg">0.0</d>
            <d id="yawSurvey" type="double" unit="deg">0.0</d>
            <d id="alphaX" type="double" unit="unit">-0.054373</d>
            <d id="betaX" type="double" unit="mm/mrad">0.21108</d>
            <d id="emitX" type="double" unit="mm-mrad">11.469</d>
            <d id="alphaY" type="double" unit="unit">-0.31096</d>
            <d id="betaY" type="double" unit="mm/mrad">0.37123</d>
            <d id="emitY" type="double" unit="mm-mrad">11.423</d>
            <d id="alphaZ" type="double" unit="unit">0.4799</d>
            <d id="betaZ" type="double" unit="deg/keV">0.61397</d>
            <d id="emitZ" type="double" unit="deg-keV">573.227</d>
            <d id="beamCurrent" type="double" unit="mA">62.5</d>
        </linacData>
        <xi:include href="spoke.xml"        parse="xml" xpointer="element(/1/2/2)" />       <!--SPOKE-->
        <xi:include href="medBeta.xml"      parse="xml" xpointer="element(/1/2/2)" />       <!--MBL-->
        <xi:include href="highBeta.xml"     parse="xml" xpointer="element(/1/2/2)" />       <!--HBL-->
    </linac>
</linacLego>
```

Figure 15. Example XML file of the ESS Linac using include elements

The LinacLego Application

The LinacLego Application is a graphical user interface for reading LinacLego XML files. The program is written in Java 1.7. The major features of the program are:

- Checking the XML grammar against the LinacLego.dtd file (*File->Open XML File*)
- Displaying the XML document in a tree display
- Parsing the XML document into a TraceWin or Dynec file (*Actions->Parse XML File*)
- Viewing the lattice in a PBS format (*Actions->Parse XML File*)
- Creating spreadsheet reports from the XML document (*Actions->Parse XML File*)
- Creating an XML document from a TraceWin file (*File->Open TraceWin File*)
- Saving a newly created XML document (*File->Save XML File*)
- Finding and marking sections of the lattice that match slot models (*Actions->Match Slot Models*)
- Finding and marking sections of the lattice that match cell models (*Actions->Match Cell Models*)
- Creating XML field profiles from TraceWin field profiles (*Field Bulder->Build XML Field Profil*e)

A screen shot of the application is shown in Figure 16.

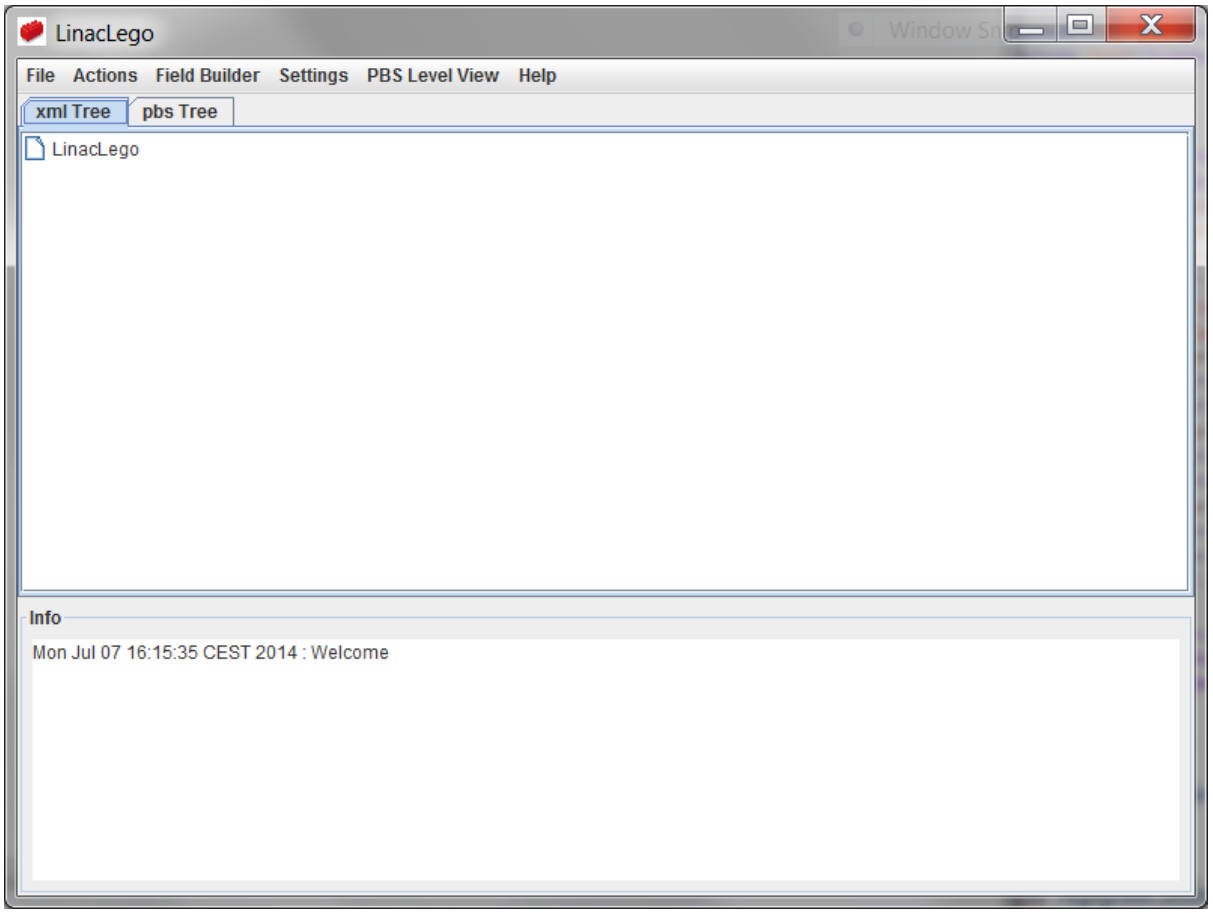

Figure 16. LinacLego Application

Checking the XML grammar

When an XML file is read into the application by executing *File->Open XML File* , it is checked to make sure that all the tags are well formed and the grammar is checked against the LinacLego.dtd file (See Appendix C). If an error is encountered a message dialog describing the error and the line location in the file is displayed.

Displaying the XML document in a tree display

When an XML file is read into the application by executing *File->Open XML File*, the XML file is displayed in a tree format as shown in Figure 17.

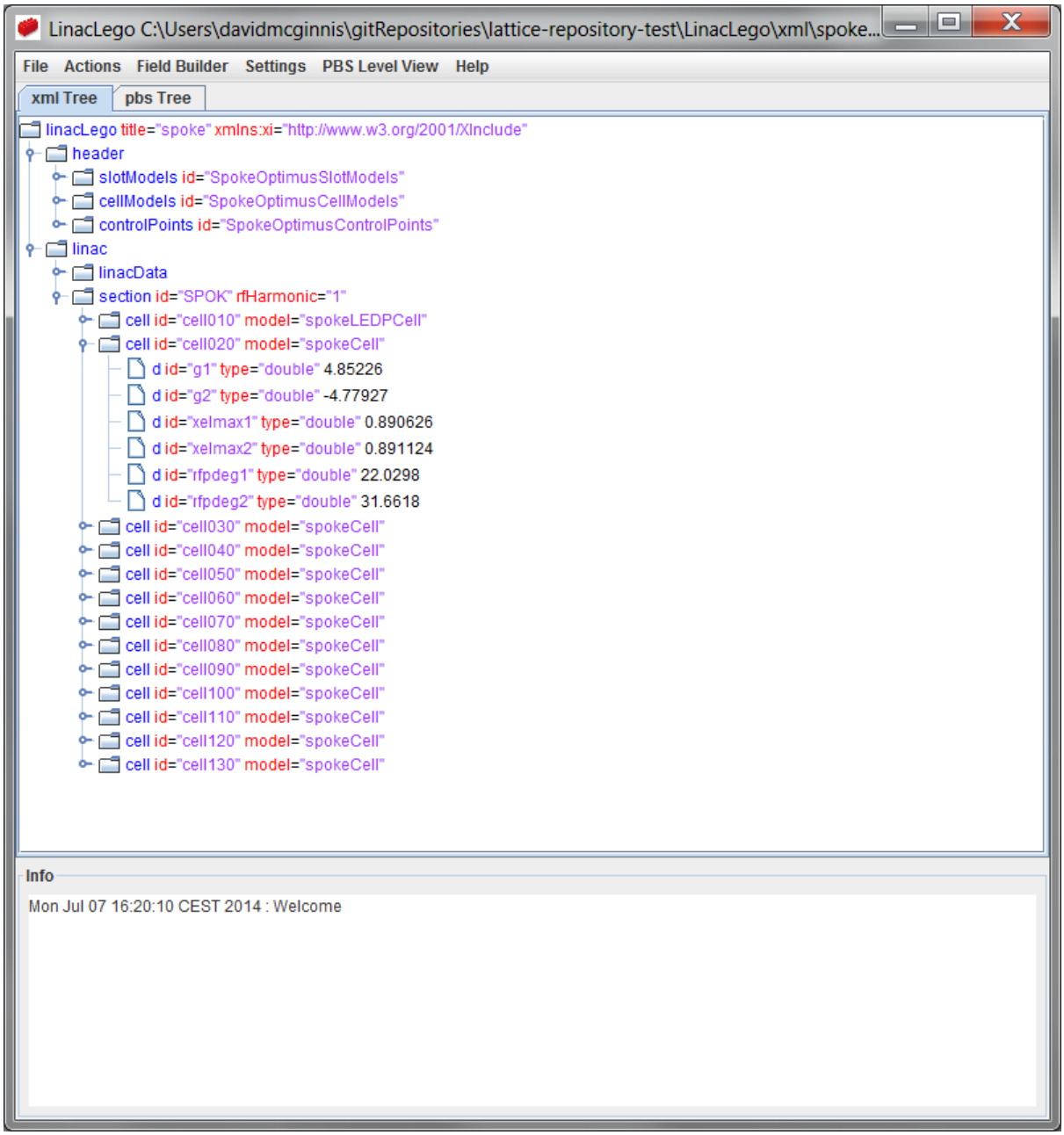

Figure 17. Tree view display of XML file.

Parsing the XML document

After the XML document has been read into the application, it can be parsed to provide a TraceWin file or a Dynec file by executing *Actions->Parse XML File*. A directory named linacLegoOutput is created in the same directory as the XML file. The TraceWin (*.dat) and Dynec files (*.in) are placed in this created directory.

Viewing the lattice in a PBS format

When the XML file is parsed, the document is also displayed in a product breakdown structure format (PBS) as shown in Figure 18. The PBS tree view differs from the XML tree view because each instance of the cell and slot models expanded in the PBS view.

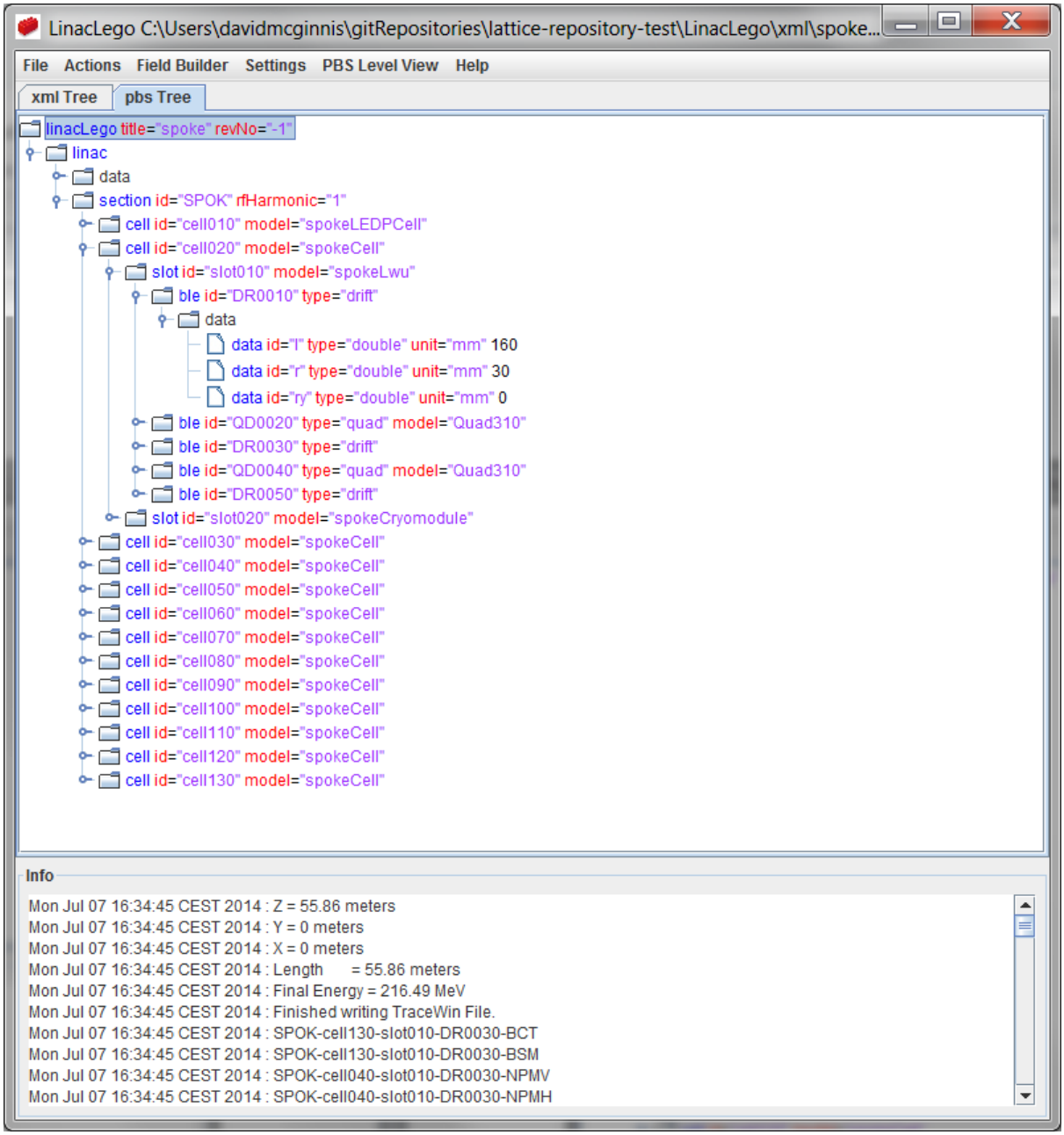

Figure 18. PBS view of LinacLego file

Creating spreadsheet reports from the XML document

When the XML file is parsed, a number of comma separated value (*.csv) reports are generated and placed in linacLegoOutput directory. The *.csv files are readily imported into Excel. The .csv file that begins with the same name as the input XML file contains an ordered list of all the beam-line elements and control points with the following data:

- Id Number
- Section Id
- Cell Id
- Slot Id
- BLE Id
- CNPT Id
- Type
- Model
- Output beam energy
- Output beam velocity
- Element length
- Center location
- Survey coordinates
- Voltage
- Synchronous phase
- Gradient
- Bend angle

The other *.csv file is appended with Paramters.csv. This file contains a section by section part count of the various models found in the lattice along with the range of characteristic values the model parameters encompass. Characteristic values vary by beam-line element type. The characteristic values are:

- Bend – Bending magnetic field
- Drift – drift length
- DtlCell – gap voltage
- FieldMap – voltage
- Ncells – voltage
- Quad – gradient
- RfGap – voltage
- ThinSteering – integrated magnetic field

Creating an XML document from a TraceWin file

Executing *File->Open TraceWin File* will ask the user to specify the location of the TraceWin *.dat file. Once the file is selected, the application will ask the user for the input beam energy and bunch frequency. The application will scan for element types detailed in Appendix A. It will be build an XML file with one cell per section and one slot per cell unless the application encounters a Lattice command in the TraceWin file. If a Lattice command is encountered in the TraceWin File, a new section will be started and elements are divided into slots according to the number of elements specified in the TraceWin Lattice command. For example, if a lattice command “Lattice 6 1” is encountered each slot will contain 6 elements and there will be one slot per cell.

Saving a newly created XML document

Once a TraceWin file has been read into the application, it has not been saved to disk. Executing *File->Save XML File*, the application will ask the user where he would like to write the file to disk.

Finding Instances of Slot Models in the Lattice

When an XML file is created from a TraceWin file, the resulting XML file has very little structure. An example TraceWin file describing the medium beta section is shown in Figure 19. When the TraceWin file is read by the application, the resulting XML file is shown in Figure 20. Because no Lattice command was found in the TraceWin file, the resulting XML structure is very flat with one section containing one cell containing one slot containing 128 beam-line elements. Once the XML file has been created it is saved under *File->Save XML File* and can be easily edited by the user in any convenient text editor. The application Notepad++[3] is a very good text editor for editing XML files because it highlights the tags and attributes and allows the user to expand or collapse tags.

Next, user edits the XML file and defines a focusing section slotModel called "medBetaLwu" as shown in Figure 21. The LinacLego application poles the operating system to see if the XML file has been modified outside the LinacLego application and if it detects an outside modification, it will ask the user if he would like to reload the XML file in LinacLego. Once the slotModel has been defined, executing *Actions->Match Slot Models* will cause the application to search for groups of beam-line elements in the lattice that match slotModel. It will replace these groups with the matching slotModel as shown in Figure 22. The process can continue as the user defines more and more slotModels as shown in Figure 23. It is also possible to define and collect all the slotModels at one time.

Finding Instances of Cell Models in the Lattice

After the process of collecting slotModels is completed the user can proceed to collecting cellModels if desired. Figure 24 shows the XML file with a cellModel consisting of two slots defined. Executing *Actions->Match Cell Models* will cause the application to search for groups of slots in the lattice that match cellModel. It will replace these groups with the matching cellModel as shown in Figure 25.

The end result is a fairly structured XML lattice file in which minimal hand manipulation was required to structure the lattice.

Creating XML field profiles from TraceWin field profiles

The philosophy of LinacLego is to avoid unstructured flat files whenever possible. Lattice simulation programs such as TraceWin and Dynec use flat fiels to describe field profiles inside of cavities. LinacLego replaces these flat field profile files with XML files as well. The structure of XML field profile file is detailed in Appendix B. The .dtd file is detailed in Appendix D. The LinacLego application can convert TraceWin one-dimensional field profiles by executing *Field Bulder->Build XML Field Profil*e. During the parsing process the TraceWin one-dimensional field profiles are generated and stored in the linacLegoOutput directory.

[3] http://notepad-plus-plus.org/

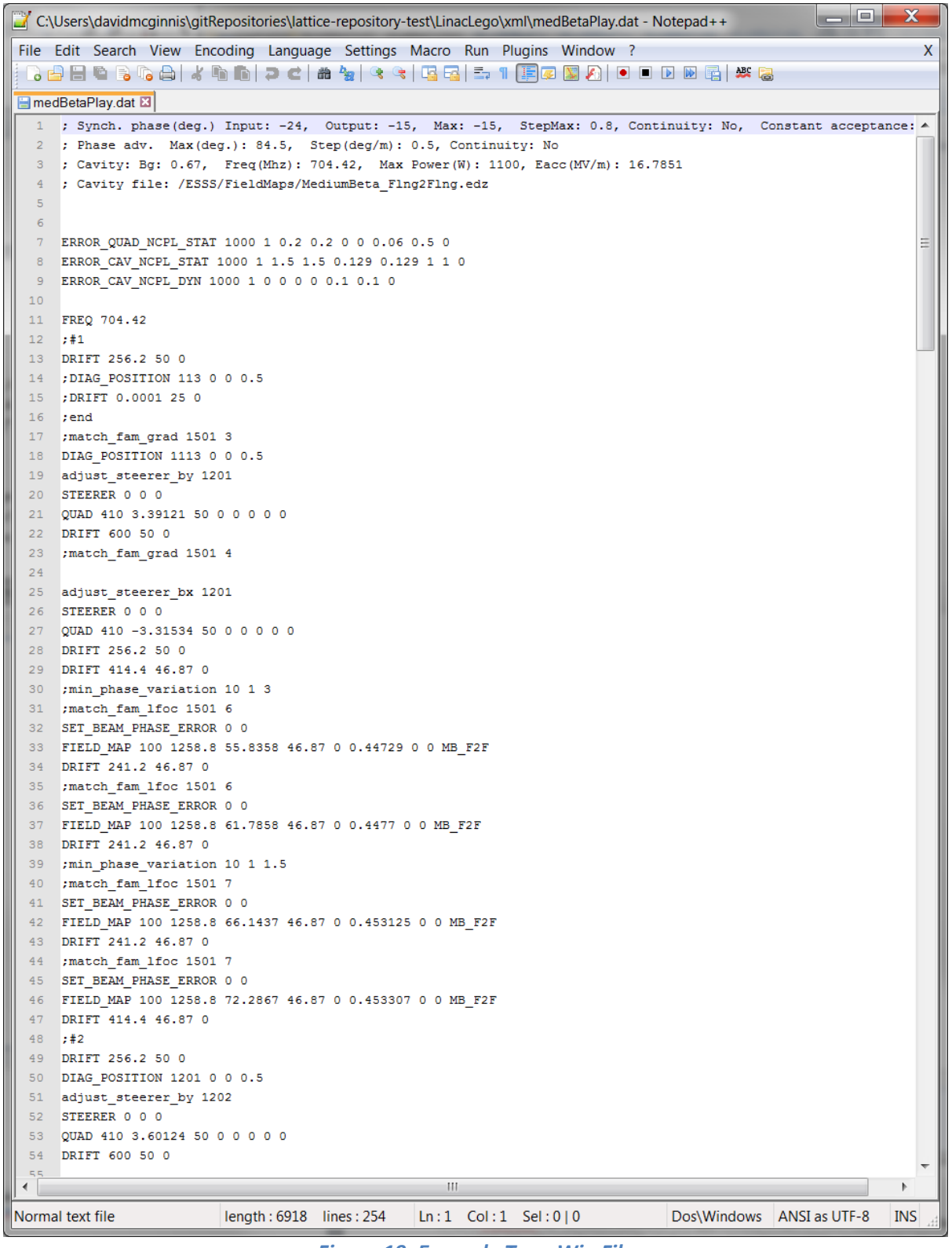

Figure 19. Example TraceWin File

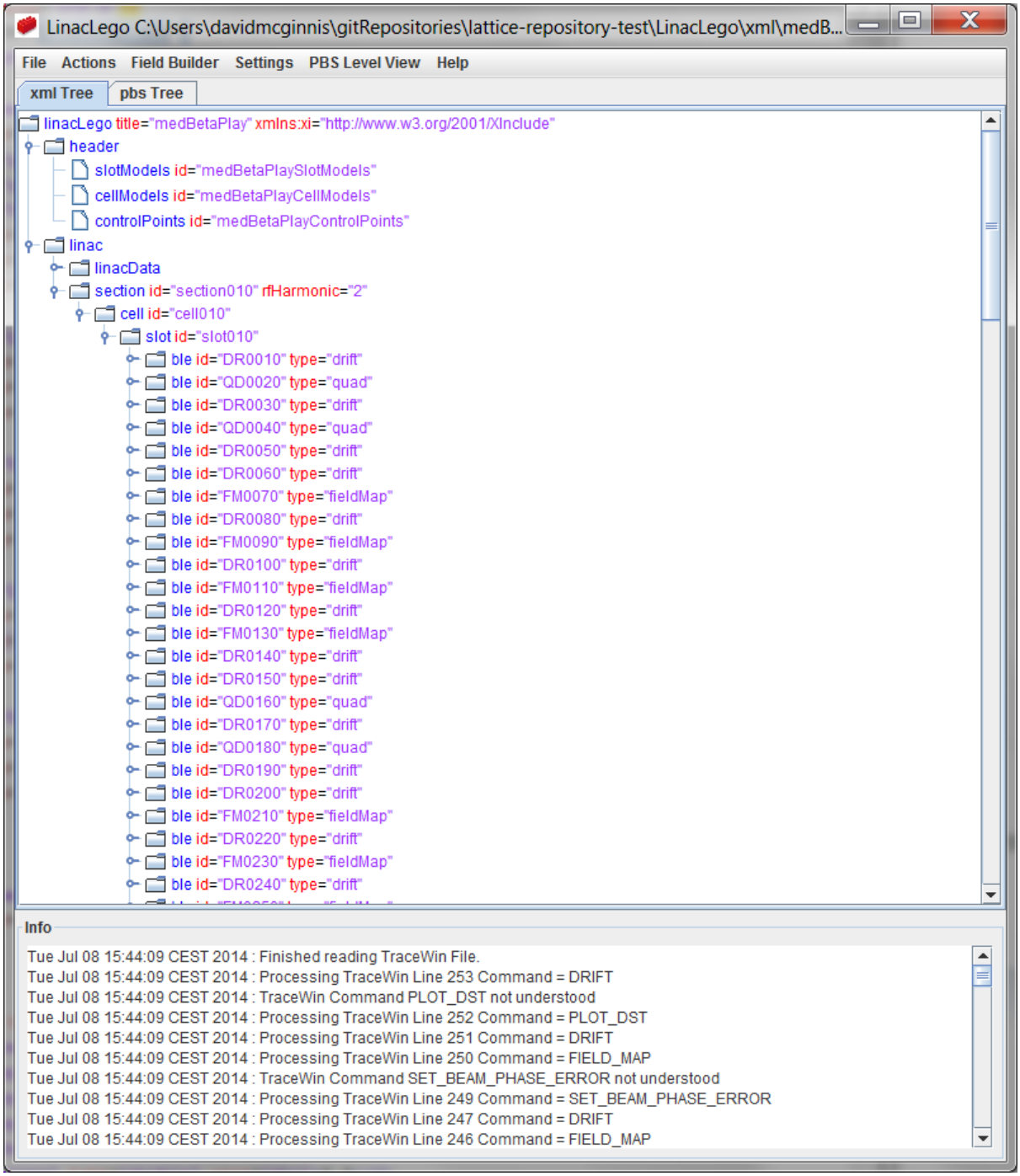

Figure 20. XML File derived from TraceWin file

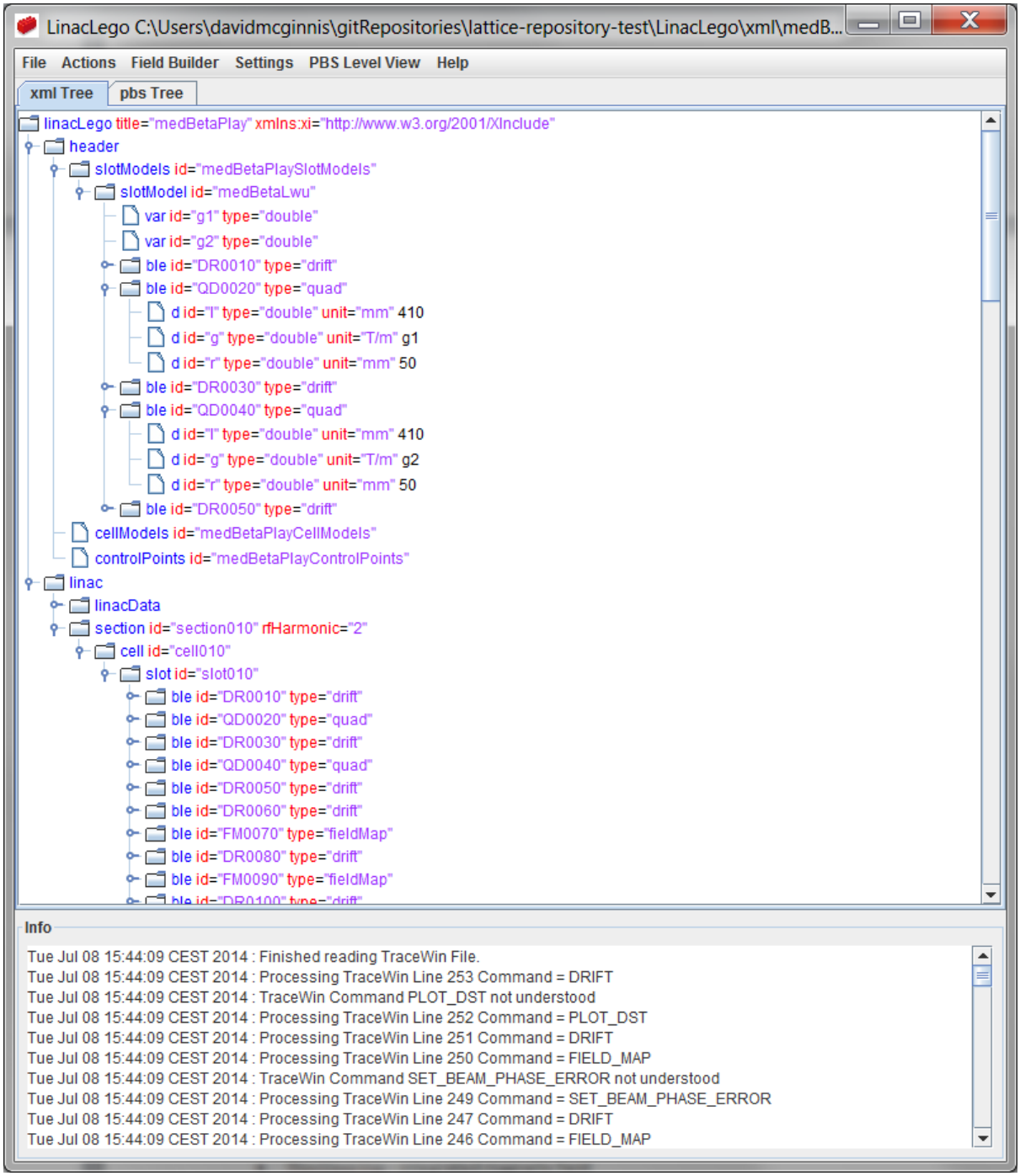

Figure 21. XML File with "medBetaLwu" slotModel defined.

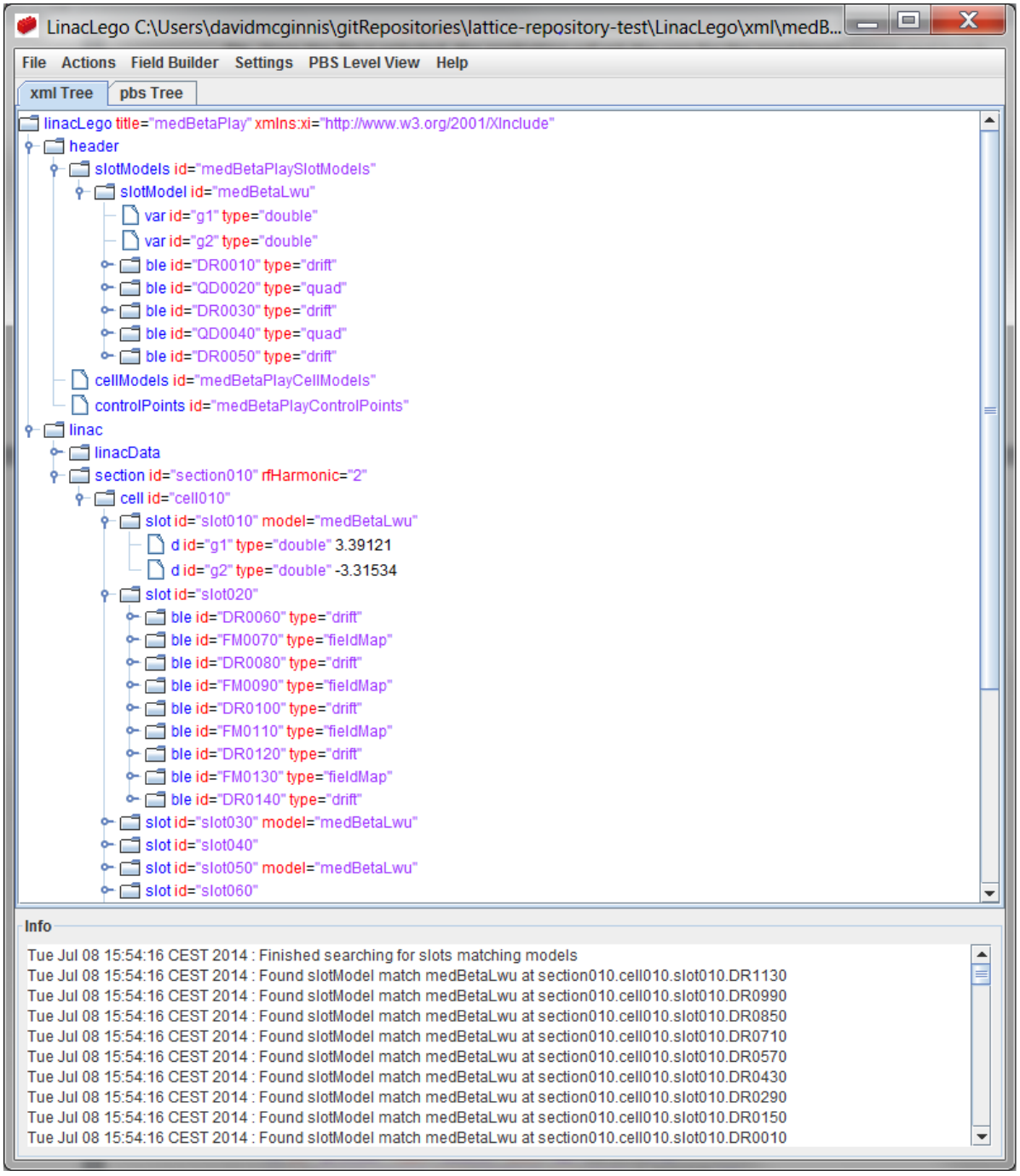

Figure 22. XML File with "medBetaLwu" slotModel collected

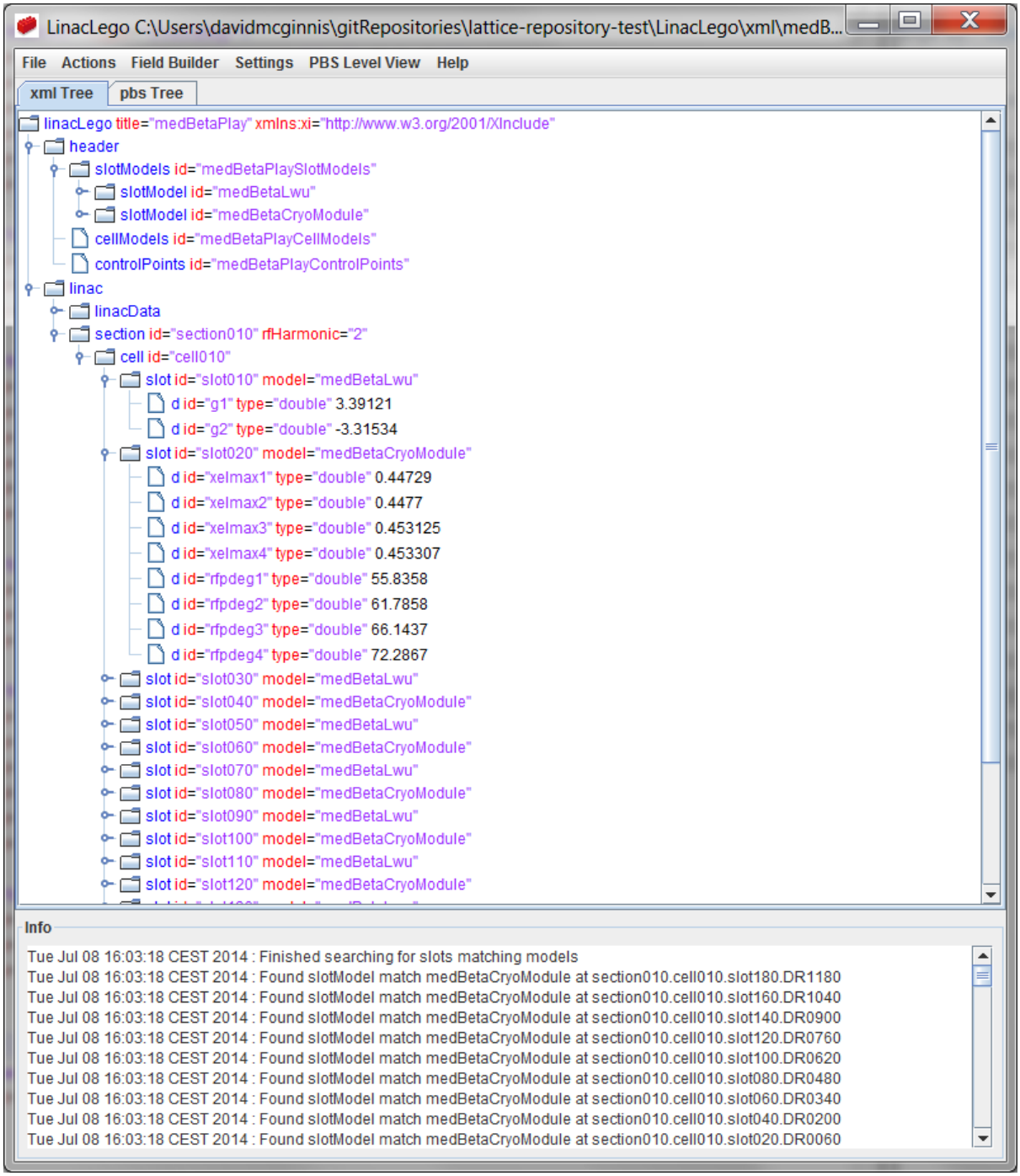

Figure 23. XML File with "medBetaLwu" and "medBetaCryoModule" defined and collected.

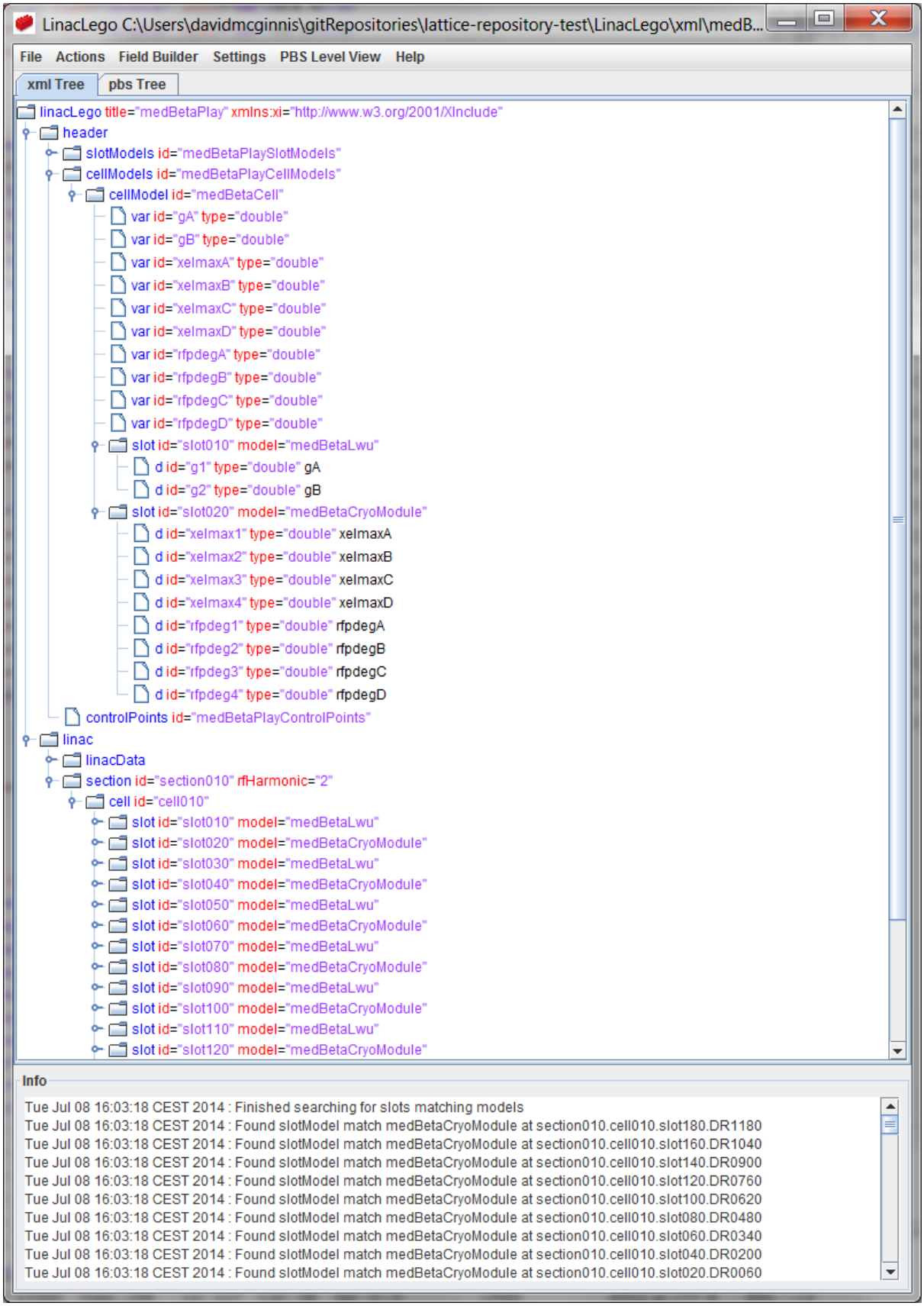

Figure 24. XML File with "medBetaCell" cellModel defined.

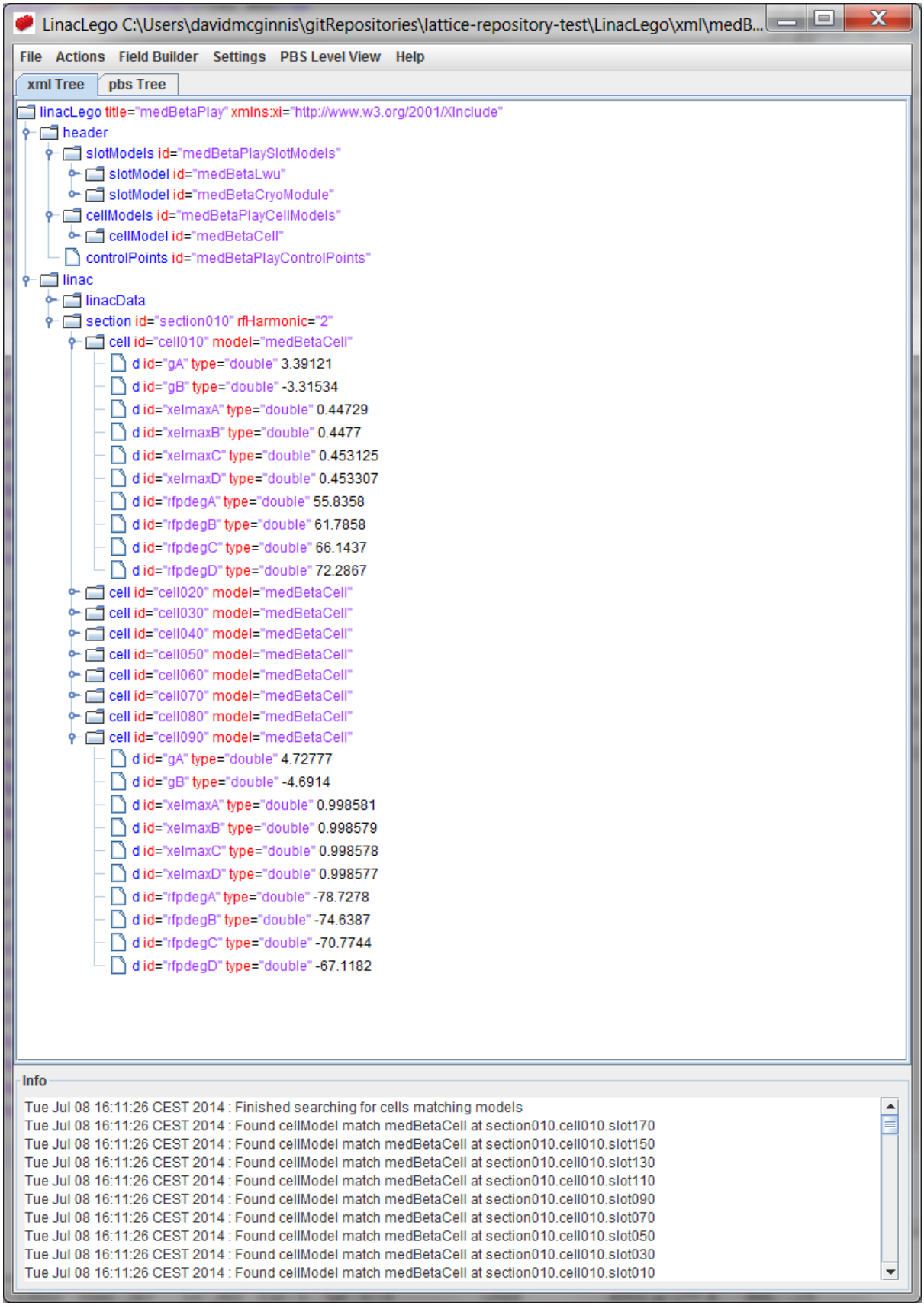

Figure 25. XML file with "medBetaCell" cellModel collected.

Appendix A. Beam-Line Element Types

Bend

```
<ble id="BD0030" model="dogLegBendMagnet" type="bend">
    <d id="bendAngleDeg" type="double" unit="deg">10.0</d>
    <d id="radOfCurvmm" type="double" unit="mm">25783.1</d>
    <d id="fieldIndex" type="int" unit="unit">0</d>
    <d id="aperRadmm" type="double" unit="mm">50</d>
    <d id="HVflag" type="int" unit="unit">1</d>
    <d id="K1in" type="double" unit="unit">0.0</d>
    <d id="K2in" type="double" unit="unit">0.0</d>
    <d id="K1out" type="double" unit="unit">0.0</d>
    <d id="K2out" type="double" unit="unit">0.0</d>
</ble>
```

Drift

```
<ble id="DR0010" type="drift" model="mebtBeamPipe" >
    <d id="l" type="double" unit="mm">50</d>
    <d id="r" type="double" unit="mm">18.4</d>
    <d id="ry" type="double" unit="mm">0</d>
</ble>
```

DtlCell

```
<ble id="DT0020" type="dtlCell" model="tank2Cell2Dtl" >
    <d id="cellLenmm" type="double" unit="mm">82.7618</d>
    <d id="q1Lenmm" type="double" unit="mm">25.3919</d>
    <d id="q2Lenmm" type="double" unit="mm">24.6773</d>
    <d id="cellCentermm" type="double" unit="mm">-0.0051598</d>
    <d id="grad1Tpm" type="double" unit="T/m">0</d>
    <d id="grad2Tpm" type="double" unit="T/m">59.5537</d>
    <d id="voltsT" type="double" unit="Volt">201352</d>
    <d id="voltMult" type="double" unit="unit">1.0</d>
    <d id="rfPhaseDeg" type="double" unit="deg">-34.18</d>
    <d id="phaseAdd" type="double" unit="deg">0.0</d>
    <d id="radApermm" type="double" unit="mm">10</d>
    <d id="phaseFlag" type="int" unit="unit">0</d>
    <d id="betaS" type="double" unit="m">0.09728</d>
    <d id="tts" type="double" unit="unit">0.810969</d>
    <d id="ktts" type="double" unit="unit">-0.33379</d>
    <d id="k2tts" type="double" unit="unit">-0.173483</d>
</ble>
```

Edge

```
<ble id="EG0020" type="edge">
    <d id="poleFaceAngleDeg" type="double" unit="deg">5.0</d>
    <d id="radOfCurvmm" type="double" unit="mm">25783.1</d>
    <d id="gapmm" type="double" unit="mm">100</d>
    <d id="K1" type="double" unit="unit">0.45</d>
    <d id="K2" type="double" unit="unit">2.8</d>
    <d id="aperRadmm" type="double" unit="mm">50</d>
    <d id="HVflag" type="int" unit="unit">1</d>
</ble>
```

FieldMap

```
<ble id="FM0020" model="spokeCavity" type="fieldMap">
    <d id="rfpdeg" type="double" unit="deg">13.3</d>
    <d id="xelmax" type="double" unit="unit">0.98</d>
    <d id="radiusmm" type="double" unit="mm">28</d>
    <d id="lengthmm" type="double" unit="mm">994</d>
    <d id="file" type="string" unit="unit">Spoke_F2F</d>
    <d id="scaleFactor" type="double" unit="unit">1.0</d>
</ble>
```

The format of a field map file is outlined in Appendix B.

Ncells

```
<ble id="nc010" type="ncell" model="highBetaCavity">
    <d id="mode" type="int" unit="unit">1</d>
    <d id="ncells" type="int" unit="unit">5</d>
    <d id="betag" type="double" unit="unit">0.86</d>
    <d id="e0t" type="double" unit="Volt/m">0.86</d>
    <d id="theta" type="double" unit="deg">110.3</d>
    <d id="radius" type="double" unit="mm">50.0</d>
    <d id="p" type="double" unit="unit">1</d>
    <d id="ke0ti" type="double" unit="unit">0.34</d>
    <d id="ke0to" type="double" unit="unit">0.34</d>
    <d id="dzi" type="double" unit="mm">12.3</d>
    <d id="dzo" type="double" unit="mm">-12.3</d>
    <d id="betas" type="double" unit="unit">0.92</d>
    <d id="ts" type="double" unit="unit">0.78</d>
    <d id="kts" type="double" unit="unit">0.112</d>
    <d id="k2ts" type="double" unit="unit">-0.3</d>
    <d id="ti" type="double" unit="unit">0.78</d>
    <d id="kti" type="double" unit="unit">0.112</d>
    <d id="k2ti" type="double" unit="unit">-0.3</d>
    <d id="to" type="double" unit="unit">0.78</d>
    <d id="kto" type="double" unit="unit">0.112</d>
    <d id="k2to" type="double" unit="unit">-0.3</d>
</ble>
```

Quad

```
<ble id="QD0020" model="Quad310" type="quad">
    <d id="l" type="double" unit="mm">310</d>
    <d id="g" type="double" unit="T/m">5.3</d>
    <d id="r" type="double" unit="mm">30</d>
</ble>
```

RfGap

```
<ble id="RF0030" model="mebtBuncherCavity" type="rfGap">
    <d id="voltsT" type="double" unit="Volt">14396.3</d>
    <d id="rfPhaseDeg" type="double" unit="deg">-90</d>
    <d id="radApermm" type="double" unit="mm">14.5</d>
    <d id="phaseFlag" type="int" unit="unit">0</d>
    <d id="betaS" type="double" unit="m">0</d>
    <d id="tts" type="double" unit="unit">0</d>
    <d id="ktts" type="double" unit="unit">0</d>
    <d id="k2tts" type="double" unit="unit">0</d>
    <d id="ks" type="double" unit="unit">0</d>
    <d id="k2s" type="double" unit="unit">0</d>
</ble>
```

ThinSteering

```
<ble id="TS0060" type="thinSteering" model="rasterMagnet">
    <d id="xkick" type="double" unit="Tm">0.65</d>
    <d id="ykick" type="double" unit="Tm">0.0</d>
    <d id="r" type="double" unit="mm">40</d>
    <d id="kickType" type="int" unit="unit">0</d>
</ble>
```

Appendix B. FieldMap Format

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE fieldProfile SYSTEM "FieldProfile.dtd" >
<fieldProfile title="MB_F2F" storedEnergy="1.0" length="1258.8" lengthUnit="mm" storedEnergyUnit="Joules" fieldUnit="Volt/m">
    <d id="0">0.0</d>
    <d id="1">-0.002403</d>
    <d id="2">-0.004807</d>
    <d id="3">-0.00721</d>
    <d id="4">-0.009614</d>
    <d id="5">-0.012017</d>
    <d id="6">-0.014421</d>
    <d id="7">-0.016824</d>
    <d id="8">-0.019227</d>
    <d id="9">-0.021631</d>
    <d id="10">-0.024034</d>
    <d id="11">-0.026438</d>
    <d id="12">-0.028841</d>
    <d id="13">-0.031245</d>
    ...
    ...
    ...
    <d id="4413">0.004813</d>
    <d id="4414">0.002407</d>
    <d id="4415">0.0</d>
</fieldProfile>
```

Appendix C. LinacLego.dtd

```
<!ELEMENT linacLego (header,linac)>
<!ATTLIST linacLego
    xmlns:xi   CDATA       #FIXED    "http://www.w3.org/2001/XInclude"
      revNo       CDATA       #IMPLIED
      title CDATA #REQUIRED>
<!ELEMENT header (xi:include|slotModels|cellModels|controlPoints)* >
<!ELEMENT xi:include (#PCDATA)>
<!ATTLIST xi:include
    href       CDATA       #IMPLIED
    parse      (xml|text)  "xml"
   xpointer   CDATA       #IMPLIED>
<!ELEMENT slotModels (slotModel*) >
<!ATTLIST slotModels
      xml:base CDATA #IMPLIED
      xmlns:xi CDATA #IMPLIED
      id CDATA #REQUIRED>
<!ELEMENT slotModel (var*,ble+)>
<!ATTLIST slotModel
      id CDATA #REQUIRED>
<!ELEMENT var (#PCDATA) >
<!ATTLIST var
      type CDATA #REQUIRED
      id CDATA #REQUIRED>
<!ELEMENT ble (d+) >
<!ATTLIST ble
      type CDATA #REQUIRED
      model CDATA #IMPLIED
      id CDATA #REQUIRED>
<!ELEMENT d (#PCDATA) >
<!ATTLIST d
      unit CDATA #IMPLIED
      type CDATA #IMPLIED
      id CDATA #REQUIRED>
<!ELEMENT cnpt (d*) >
<!ATTLIST cnpt
      section CDATA #REQUIRED
      cell CDATA #REQUIRED
      slot CDATA #REQUIRED
      ble CDATA #REQUIRED
      devName CDATA #REQUIRED
      type CDATA #REQUIRED
      model CDATA #IMPLIED
      id CDATA #REQUIRED>
<!ELEMENT cellModels (cellModel*) >
<!ATTLIST cellModels
      xml:base CDATA #IMPLIED
      xmlns:xi CDATA #IMPLIED
      id CDATA #REQUIRED>
<!ELEMENT cellModel (var*,slot+) >
<!ATTLIST cellModel
      id CDATA #REQUIRED>
<!ELEMENT slot (d|ble)* >
<!ATTLIST slot
      model CDATA #IMPLIED
      id CDATA #REQUIRED>
<!ELEMENT controlPoints (cnpt*) >
<!ATTLIST controlPoints
      xmlns:xi CDATA #IMPLIED
      xml:base CDATA #IMPLIED
      id CDATA #REQUIRED>
<!ELEMENT linac (linacData,(xi:include|section)*) >
<!ELEMENT linacData (d*) >
<!ELEMENT section (cell+) >
<!ATTLIST section
      xml:base CDATA #IMPLIED
      xmlns:xi CDATA #IMPLIED
      rfHarmonic CDATA #REQUIRED
      id CDATA #REQUIRED
      type CDATA #IMPLIED>
<!ELEMENT cell (slot|d)* >
<!ATTLIST cell
      id CDATA #REQUIRED
      model CDATA #IMPLIED>
```

Appendix D. FieldProfile.dtd

```
<!ELEMENT fieldProfile (d+)>
<!ATTLIST fieldProfile
      title        CDATA      #REQUIRED
      storedEnergy CDATA      #REQUIRED
      length       CDATA      #REQUIRED
      storedEnergyUnit CDATA  #REQUIRED
      lengthUnit   CDATA      #REQUIRED
      fieldUnit    CDATA      #REQUIRED >
<!ELEMENT d (#PCDATA) >
<!ATTLIST d
      id CDATA #REQUIRED>
```