The XML editor

Sebastien Ponce

1.0 Overview

The XML editor is a tool provided to edit the XML files of the detector description database without having to learn the XML syntax. It is provided as a separate Gaudi package and was written in Java. As all Gaudi packages, it runs both under Linux and Windows but since it is a Java native tool, it should also work on any platform where a Java virtual machine is provided.

This documentation corresponds to XmlEditor version 4 release 1.

2.0 Installation and usage

There are two ways of installing and using XmlEditor : either via AFS or locally

2.1 Via AFS

XmlEditor is installed in the standard LHCb software release area, in the `Det/XmlEditor` directory. In order to launch the application, just launch one of the scripts in the `scripts` subdirectory. `xmlEditor` runs under Linux while `xmlEditor.bat` runs under Windows.

Note that you must provide a Java runtime environment to use XmlEditor. This has to be version 1.2.2 or later (version 1.3.0 or later is strongly suggested).

2.2 Locally

In order to use XmlEditor locally, you'll have to install it. There are some requirements for this installation :

1. A Java compiler and runtime environment must be provided. Version 1.2.2 or later is needed and version 1.3.0 or later is suggested. You can find it easily on the web. Some providers are Sun (for Linux, Windows and Sparc), blackdown (only for Linux) and IBM (for Linux and Windows) .
2. An installation of xerces for java must be provided. This is a free XML parser that can be downloaded at *http://xml.apache.org/xerces-j/index.html*. Version 1.2.0 or later is needed and version 1.4.3 or later is strongly suggested.
3. An installation of dtdparser must be provided. This is a free dtd parser that can be downloaded at *http://www.wutka.com/dtdparser.html*. Version 1.13 or later is needed.
4. CMT is used to simplify the installation but this is not an absolute requirement.

All these tools are of course also available on AFS.

Once all tools are available, get the Det/XmlEditor package from the LHCb release area or from the LHCb CVS repository.

Provided that you have CMT, the installation and compilation is very simple. Go first to the "cmt" subdirectory of the XmlEditor package and adapt the requirements file to your needs (you may change the location of xerces and dtdparser). Then type :

```
cmt config
make
```

That's all, you can now use the scripts provided in the `scripts` subdirectory to launch the application. Use `xmlEditor` under Linux and `xmlEditor.bat` under Windows.

If you don't have CMT, the compilation is a bit more complicated but still manageable by non specialists. You first have to prepare your environment :

1. xerces.jar must be in your classpath
2. dtdparser.jar or equivalent (mine is dtdparser113.jar) must be in your classpath
3. java must be correctly installed which supposes that the java and javac commands are available (plus javadoc for documentation) and that a file called rt.jar is in your classpath
4. Finally, the subdirectories `src` and `classes` of the XmlEditor package must be in your classpath

Once this is done, go to the root directory of XmlEditor (where `src` and `classes` appear as subdirectories) and type the following :

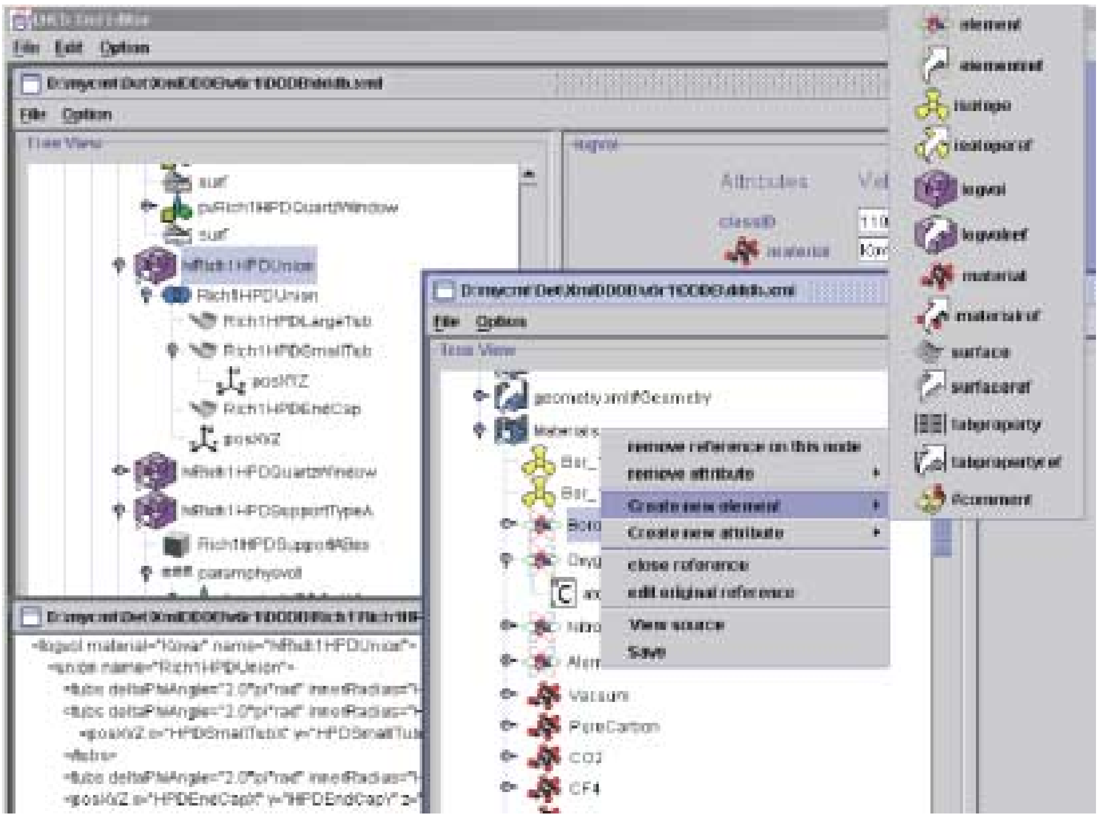
```
javac -d classes src/XmlEditor/Editor.java
```

You've just compiled the editor and created the source documentation. To run the editor, one last configuration is needed : the directory src/icons of the XmlEditor package must be in your classpath (otherwise, it works but without any icon). Now you can run the editor with the following command :

```
java XmlEditor/Editor
```

3.0 Overview of the Editor

XmlEditor is an Explorer like application that is able to display an XML document as a tree. Each node of this tree represents an element in the XML document. A snapshot of the GUI of the XmlEditor is shown in Figure 1



```
<logvol material="Kovar" name="lvRich1HPDUnion">
  <union name="Rich1HPDUnion">
    <tubs deltaPhiAngle="2.0*pi*rad" innerRadius="
    <tubs deltaPhiAngle="2.0*pi*rad" innerRadius="
      <posXYZ x="HPDSmallTubX" y="HPDSmallTu
    </tubs>
    <tubs deltaPhiAngle="2.0*pi*rad" innerRadius="
    <posXYZ x="HPDEndCapX" y="HPDEndCapY" z=
```

Figure 1 Overview of the Editor

The content of an XML file is displayed in a tree-like format on the left of the window while the right part is dedicated to the display of attributes of the currently selected node of the tree.

The basic functionality of an Explorer-like application are present here. The usage of the editor is thus pretty obvious. Here is a non-exhaustive list of the facilities :

1. "Open" menu to deal with files, including ability to create new files and a list of recently opened files
2. "Edit" menu with classical cut and paste. Note that the shortcuts Ctrl C for copy, Ctrl V for Paste, Ctrl X for cut and Ctrl B for Paste into parent are understood.
3. Global "Option" menu to change global display settings.
4. Per file "Option" menu to change per file display settings.
5. Popup menus on every node and on attributes : these menu are contextual and propose options like create subnode, remove node, create attribute, save file, close reference ...

6. Drag and Drop facilities to move, copy or link nodes. The default behavior is to move nodes, you can copy them by pressing Ctrl while dropping and create a reference on a node by pressing Ctrl and Shift while dropping. Take care that some java virtual machines have problems with the display of the drag and drop cursors (see Section 6.3)
7. Undo and Redo facilities (see Section 6.4). Ctrl Z can be used to undo while Ctrl Y can be used to redo. Undo all and Redo all are also provided.

4.0 Data specificity

Some features of this editor are related to the specificity of the detector description data. This is presented here.

4.1 References inside XML

Among the nodes appearing in the editor, you will notice that some have a big arrow across their icon. These special nodes are actually not data but references to data. They are a kind of pointers to real nodes.

In practice, these are special nodes with a `href` attribute defining what they are pointing to. The string contained by this href attribute is of the form `file#nodeName` where :

1. `file` is the file where the pointed node is located. The path can be absolute or relative to the current file and is given in the unix way (ie with '`/`'). It is highly recommended to put relative paths here since the data files may be used on different machines by different users.
2. In the case of a local reference (inside a single file), `file` should be empty and thus the reference of the form `#nodeName`.
3. `nodeName` is the name of the pointed node. What is called name here is the value of the `name` attribute of the pointed node.

As every single node, the reference nodes have a type. This type must be in accordance with the one of the pointed node. As an example, only a `catalogref` node can point to a `catalog`. This is a general rule that the type of a reference must be the type of the node pointed to plus `ref`. If this rule is not applied, it could happen that the editor be unable to open the reference, since it would be looking for the base node type.

When a reference appears in the XML tree, you can easily open it and edit its content by double clicking on the node. The icon will change showing that you no longer edit the reference but the underlying node itself. This implies that nodes from several files can be displayed in the same tree. However, icons are different for real nodes, references and open references. See Section 4.3.

To close a reference, just right click on the node to close and choose the "Close reference" item from the popup menu.

To create a reference, either create it by hand (as any other node) or use the drag and drop facilities (see Section 6.3).

4.2 Elements as attributes, values as attributes

There is a clear distinction in XML between elements and attributes : elements are part of the file structure and are displayed in the left part of the editor while attributes are attached to a given element and are displayed on the right part of the editor.

However, some elements are what we call *attribute-like*. This only means that they only have a single value, that they have no subelement, and that they don't have any attribute. These elements could have been replaced by an attribute of their parent. Even though this was not the case, by default they are displayed in the XmlEditor in this way. You can of course change this in the Option menu (see Section 7.0).

Another point is that the value of an element in XML is considered as a text subelement of the element whose value is the value of the element itself. The XmlEditor prefers considering that the value is an attribute of the node (with attribute name `Value`). Thus values are displayed this way by default. Once more, it is configurable in the Option menu (see Section 7.0).

4.3 Icons

As could be seen in Figure 1, most of the elements of the XML files have a specific icon that allows the user to distinguish them easily. These icons are part of the XmlEditor software but you can change them or add new ones easily (see Section 7.1.3).

There are actually three types of icons as shown in Figure 2:

1. regular icon
2. reference icon: by default, the regular icon with a big arrow on top. Used for reference nodes when these are closed
3. dereference icon: by default, the regular icon with a small arrow in the bottom left corner. Used for reference nodes once these are opened.

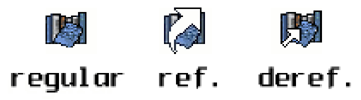


Figure 2 Three icons for an element

The icon to be associated to a given node is chosen as follows: by default, the icon associated to the type of the element is selected. The setup is used to get the actual image. This works for regular nodes and also for references whose type is the regular type plus "ref". But if the node is recognized as being the content of an open reference, then the icon used is the one associated to the type of the node plus "deref". Thus, the last icon in Figure 2 is associated to type "catalogderef" which actually doesn't exist.

Besides these icons, special icons are reserved for comments and nodes that are from a type not declared in the dtd of the file. See Figure 3

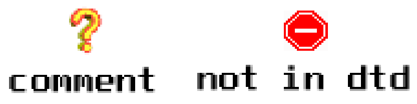


Figure 3 Special icons

4.4 Dtd considerations

The dtd is the description of the different elements that can be used inside a given XML file and of their attributes. It is either inside the file itself or in a separate file (the default case in the detector description database).

Without such a description, it is pretty hard to edit an XML file since the editor cannot have any clue on the existing nodes and their attributes. Thus, XmlEditor has very reduced functionality when no dtd is present or when the dtd is not valid.

Here are listed the non-working functionality in such a situation :

1. No use of icons
2. No creation of elements
3. No creation of attributes
4. Element names are always their type
5. In the per-frame "Option" menu, only the "Display Comments" and "Value as attribute" items are enabled
6. None of the attributes have a list of values proposed

5.0 File management

We deal here with file manipulation in the XmlEditor. This should be simple but the reference mechanism makes things less trivial.

5.1 File Opening

This works as expected with a nice file browser. There are however two little things to notice:

1. several files can be opened in several frames since XmlEditor is a multiwindow environment. Of course, cut-and-paste and drag-and-drop can be performed between different frames
2. the "File" menu has a "Recent" submenu containing every recently opened file. Thus, most of the time, one won't need to go through the Open item and browse the file system.

5.2 File creation

The creation of a new, empty Xml file supposes the knowledge of two things :

1. the name of the dtd used in this file : for those who don't know anything about XML, see it as a file giving the type of the XML file you're editing. For example, the XML files used for geometry descriptions are different from those used for materials. This actually defines the types of node you will be able to create inside your file and their attributes.
2. the name of the root node of the file : it has to be a type defined inside the dtd if you want to be able to add subnodes to it afterwards.

So, the File->New menu should ask you for these two data. It actually does it if you create a "Generic" XML file (see Figure 4). But since only few dtd files are used and each one with always the same root node, some predefined file types appear in the File->New menu that allows the user to create files without any question asked.

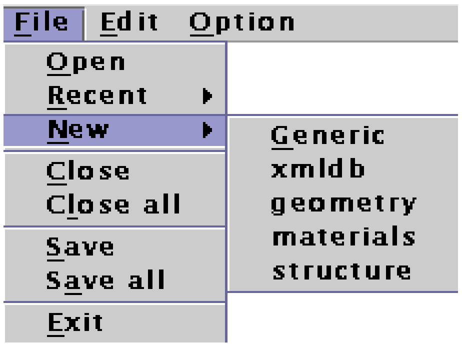


Figure 4 The File -> New menu

These predefined types are not cast in stone, they can be user defined through the setup of the editor (see Section 7.1). It actually stores types associated to a dtd and a root node name.

5.3 File Saving

Saving a file should be pretty easy, the user should just have to click on the "Save" button but it is not so simple and the user should better think before saving.

The problem comes from the fact that several files are displayed in a single tree, due to the reference mechanism. Thus, it is hard to say what will actually be saved : will it be a single file, every file in the current window or every file in every window ? It actually depends on which "Save" button you hit:

1. "Save" popup menu item : this "Save" button will only save the file containing the node on which the popup menu appeared. If this node is an opened reference, it will save the file containing the node pointed to by the reference and not the one that contains the reference itself.
2. "Save" item of each frame's "File" menu : this will save every file used inside this window. This means, in case of an open reference, that both the file containing the reference and the one containing the pointed node will be saved.
3. One must add that the closing of a reference does not close the file containing the pointed node and that this file still remains in the list of files to be saved. Take one example. The node totoref, in file toto.xml is a reference on the node titi, located in file titi.xml. Suppose that totoref is opened to make titi appear. Then some edition is done on titi and totoref is closed again. When the saving will happen, titi.xml will be saved as well as toto.xml.
4. "Save" item of the main window's "File" menu : hitting this item is strictly equivalent to hitting the File->Save item of the currently selected window
5. "Save All" item of the main window's "File" menu : this is equivalent to hitting every File->Save item of every opened window.

5.4 File closing

This is less complicated than file saving but still needs some explanation.

Basically, every "Close" item only closes the current frame (the one where the item is or the selected one) and the "Close All" command closes every frame.

This is for frames but what about the files ? As a matter of fact, one file can be opened in several frames. So when will a file be closed ? The rule is simple: a file stays open as long as any frame that once opened it is not closed. Whenever every such frame is closed then the file is closed and opening it again will perform a full parsing of it.

Actually, all this means that one file, if displayed in several frames, is only opened once in the editor, in the sense that every frame is editing the same file and that every edition in one frame eventually modifies the others.

6.0 Edition

Here the edition features of the XmlEditor are described.

6.1 Regular Edition

6.1.1 Element creation and deletion

The only way to create an element is to add it as child of an existing one. A right click on a given node will give the ability to create such a child node, see Figure 5.

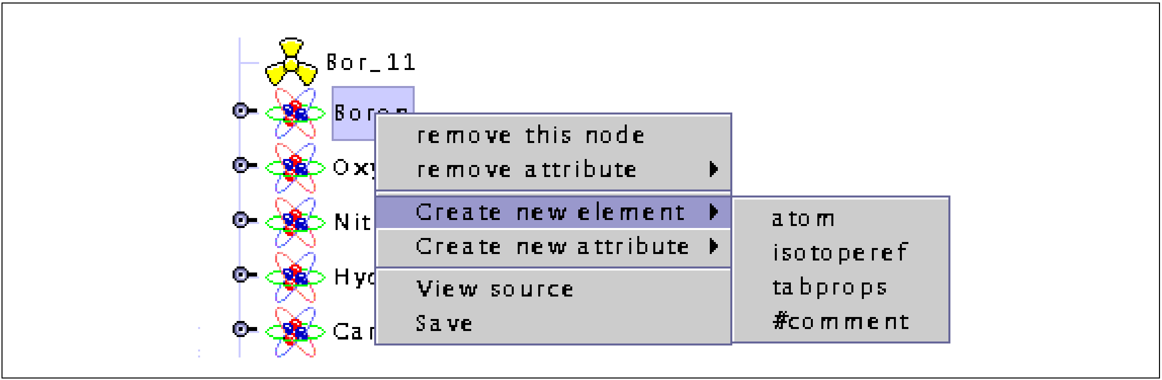


Figure 5 Creating a new element

The user can choose the type of node he wants to create among those possible for the selected node. Note that the new node will be created with the string "Default Value" in place of every editable value and that it will already have some attributes (the required one) but with the same default value (except for those where a default value is given in the dtd).

To remove a node, simply right click on it and choose the right item. Note than you cannot remove an opened reference. Close it first.

Concerning comments and value nodes, they can be created and removed the same way, use the #comment and #value items of the popup menu (see Figure 5).

6.1.2 Attribute creation and deletion

The creation and deletion of attributes is done via a right click on the parent node. As for element creation, the user must choose in a reduced list of attributes accepted by the node. Note that an attribute is unique under a node. Thus, the creation of an already existing attribute will be canceled and the user has to remove an attribute before recreating it.

The attributes that are created can be of two kinds : text ones and list ones. The user doesn't have any choice here since the kind of an attribute is defined by the dtd. In the case of a "list" attribute, the value of the attribute can only be selected in a list of predefined values. In case of a "text" attribute, any value can be entered. Thus, the display of these two kinds of attribute is different, the former are displayed in a combo box while the later use a text field.

To delete an attribute, either use the "Remove attribute" item in the popup menu that appears after right clicking on a node or right click directly on the attribute's name in the left part of the frame and select "Remove attribute". Note that some attributes are defined as "required" in the dtd. You can delete them, but at your own risk!

6.1.3 Editing

Editing an attribute value or a node name is very easy. For attributes, you just have to click on it. For names of elements, you have to click on it while selected. In this last case, you may get a message telling you that the name is not editable since it does not exist. This is due to the particular definition of the element name, see next section.

6.1.4 Editing element names

The name of an element is not defined in XML. The only given data is the type of the element. Since this type is symbolized by the icon, and since it is common to many nodes, the XmlEditor does not in general use this type as a name for nodes.

Most of the time, the name of a node is actually the value of one of its attributes. The rule is the following : if a "name" attribute exists in the node, its value is the name of the node. Else, if a "href" attribute exists, its value is the name of the node (that's for references). Else the node has no name and its type is used. That's where you can get an error message when you try to edit the name since the type is not editable.

You can of course change the definition of the name for a given type of node. This is the object of the "Element Name Displaying" item in the "Option" menu of each frame (see Section 7.2.6).

6.1.5 Editing references

A reference can be edited as a regular node but has some special edition facilities on top of that. You can double-click on a reference to open it. Once it is open, you can close it by right clicking on the node displayed and choosing the "Close reference" item. You can also edit the original reference by right clicking on the node and choosing the "Edit original reference" item. This will first close the reference and then start edition.

6.2 Cut and Paste

The Cut and Paste mechanism allows the user to move, cut and paste nodes in the tree. It only deals with the currently selected node. There are two ways to use it : the "Edit" menu of the main window or the key bindings : Ctrl-C to copy, Ctrl-V to paste, Ctrl-X to cut and Ctrl-B to Paste into Parent.

Cut and Paste from and into other applications is also provided. Since nodes as defined in the xmlEditor do not exist in these application, the cut and paste deals with text in these cases, ie with the xml representation of the imported / exported nodes. Thus, importing xml code will create the corresponding node and pasting outside the xmlEditor some nodes will generate the corresponding xml code.

6.3 Drag and Drop

The drag and drop facility allows to copy nodes, move nodes or create references using the mouse. The default behavior of the drag and drop is to move the nodes concerned. But one can also copy nodes by hitting Ctrl while dropping and even create references on nodes by hitting Ctrl and Shift while dropping.

Note that you can drop nodes either inside a node, before the node or after the node. You drop inside when the node is the node is surrounded, before the node if it is overlined and after the node if it is underlined.

6.4 Undo/Redo facilities

The use of the Undo/Redo facilities is not as obvious as it could be due (once more) to multiple files displayed in one single window.

The functionnalities provided are Undo (Ctrl Z) to undo the last edition, Undo all to undo every edition done so far, Redo (Ctrl Y) to redo the last undone edition and Redo all to redo every undone edition. What is considered to be an edition is changing the value of an element or an attribute, creating or deleting an element or an attribute and opening or closing a reference.

As I said previously, the presence of references causes some troubles in this perfect undo/redo mechanism. Obviously, once the user closed a reference, the editions made in the file that was pointed to by the reference cannot be kept in the undo list since the file was more or less closed. Thus, undoing the opening of a reference erases all editions done in the corresponding file from the redo list, losing them. You’re warned !

6.5 View Source

This is not really an edition feature. It is just a way to see the result of your edition in the XML file itself. This item (located in the popup menu of each node) displays the content of the XML file containing the currently selected node as it would be if the user saved the file. The View Source window is not editable nor does it change when you edit the tree. However, you can open several View Source windows and see the differences that your edition made to the XML.

7.0 Configuration

This section deals with the possible configurations of the editor. There are two configuration facilities :

1. the "Setup" : this is a high level configuration that will be saved on disk and retrieved when the XmlEditor will be run again. This allows configuration of the icons used and of the File->New menu.
2. the "Option" menu in each frame : this more or less configures the display of the current frame.

Let's see all that in detail.

7.1 Setup

Since the Setup is persistent, it is saved in a file in the home directory of the user. This file is named ".xmleditorrc.xml". As suggested by its name, it is an XML file and that's why it is associated to its dtd, saved in the ".xmleditorrc.dtd" file.

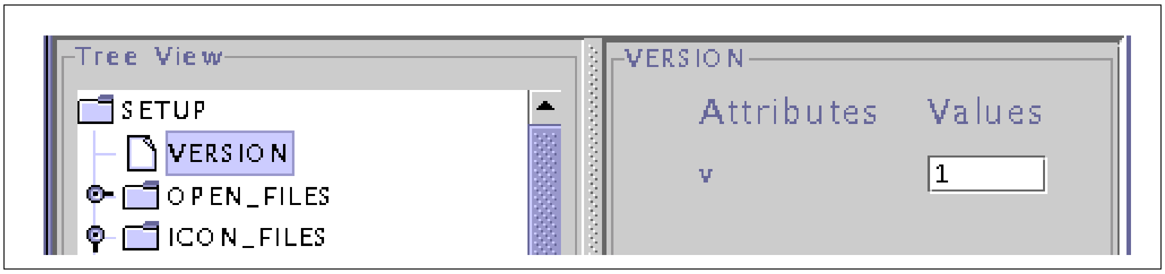
Since the Setup is an XML file, it is very easy to edit : just choose "Edit Setup" in the global "Option" menu and this file will be opened as a regular XML file. Thus all edition facilities are provided to create new nodes or remove others.

In order to take the modifications of the setup into account, just use the "Reload setup" item in the global "Option" menu. Every frame displayed will be updated then.

Those who know XML can look at the dtd file to understand the structure of the Setup file. For the others, we provide a simple view of it in the remainder of this section together with details on how it is used in the editor. The file is divided into three main categories : "OPEN_FILES", "ICON_FILES", and "DETECTOR_DATA". On top of that, a VERSION element exists. We detail further each category here.

7.1.1 Version

The reason for having introduced a VERSION element in the ".xmleditorrc.xml" file is pretty obvious : managing the evolution of this specification file in a clever way. Up to now, the dtd of this file was changing with each release of the xmlEditor with no backward compatibility. From this version (excluded), the backward compatibility will be supported between configuration files and creation of a new configuration file from an old one will be provided.



Attributes	Values
v	1

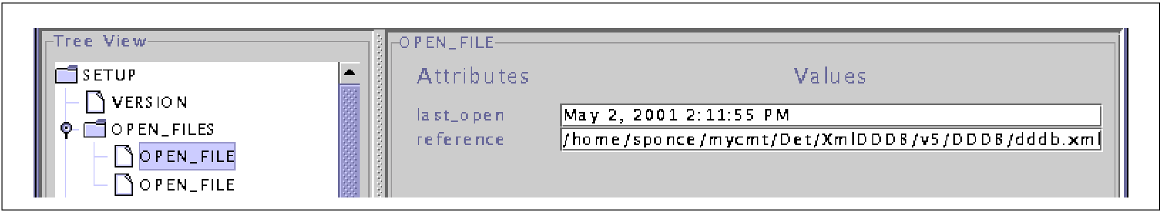
Figure 6 VERSION field in the setup

Technically speaking, the use of the VERSION element is very simple : it has a single attribute called v and giving the version number.

7.1.2 Open_files

This part of the Setup file is dedicated to the storage of a list of the most recently opened files. This is basically what is used to build the "Recent" submenu of the main "File" menu.

You shouldn't have to edit this part of the file but you could if necessary. Figure 7 shows the content of a node of this section. It is pretty obvious : you get a file name and the time of its last opening.



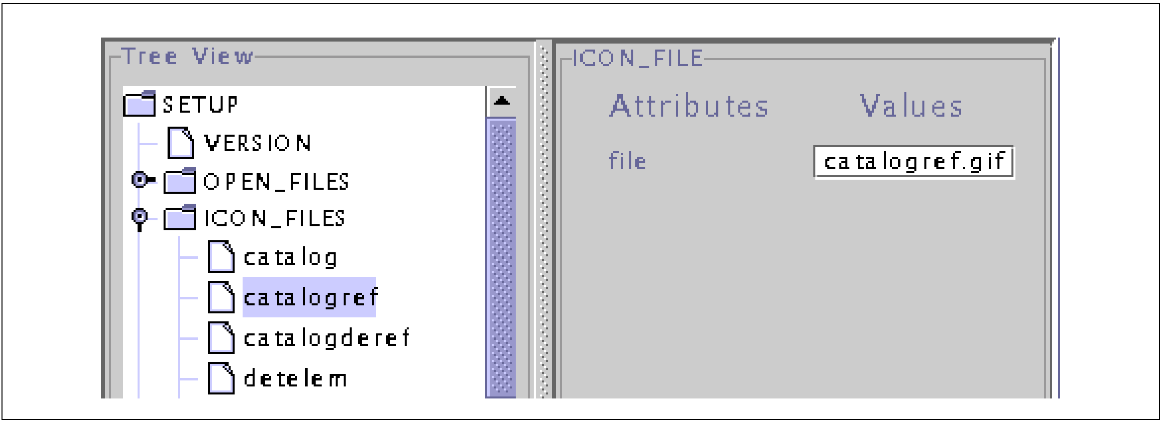
Attributes	Values
last_open	May 2, 2001 2:11:55 PM
reference	/home/sponce/mycmt/Det/XmlDDDB/v5/DDDB/dddb.xml

Figure 7 OPEN_FILE field in the setup

7.1.3 Icon_files

This part is dedicated to the definition of the icons used to display nodes.

Every node in this section defines an icon and associates it to a given node type. The node type is given as the name attribute of the ICON_FILE node (and displayed in the tree on Figure 8) while the icon file associated is given as a file attribute.



Attributes	Values
file	catalogref.gif

Figure 8 ICON_FILE field in the setup

Note that the name of the icon file is searched in the directories and jar files contained by the CLASSPATH environment variable. Thus, take care to add the corresponding directory in your CLASSPATH if you define icons.

Also note the presence of element types that don't actually exist for "dereferenced" nodes. See Section 4.3 for more details.

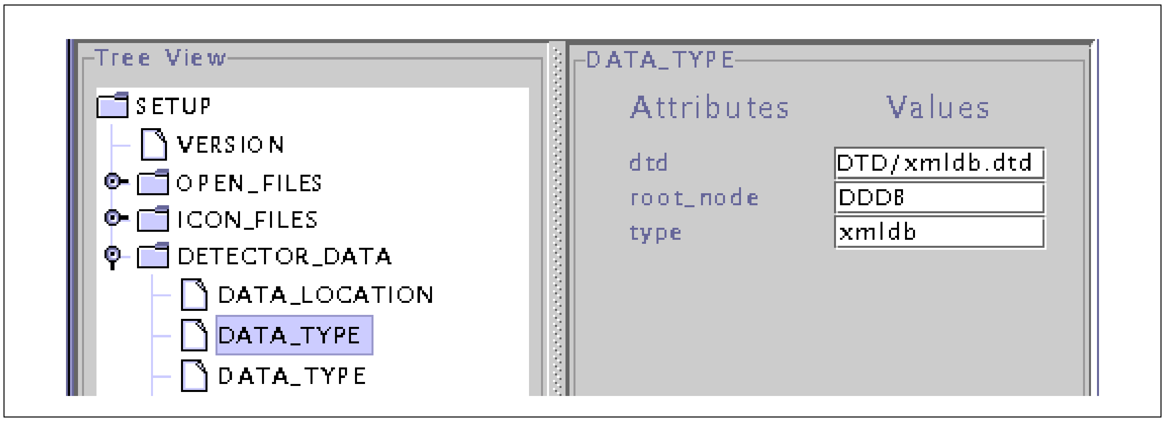
You may think here that every user has to define its own set of icons. Actually, most icons are predefined and already present in the Setup when you run XmlEditor for the first time. The only usage of the ICON_FILES section should be to add new icons if you define new types or to change some unpleasant icons.

7.1.4 Detector_data

This last part stores kind of metadata concerning the detector description data.

The node named "DATA_LOCATION" only contains the path to the root of the detector data repository. Take care that the value provided here by default may not be correct for your particular case and that you may have to change it in order to make the File->New menu work.

Every other node is a definition of a data type. A data type is defined by a name, a dtd and a root node as shown in Figure 9. It defines the type of XML file to be created for edition. Valid types are those which appear in the File->New submenu, after the "Generic" item (see Figure 4). Note that the dtd file name must be relative to the DATA_LOCATION given in the first node of the DETECTOR_DATA section. This allows to change the location of the data without having to change every data_type description.



Attributes	Values
dtd	DTD/xmldb.dtd
root_node	DDDB
type	xmldb

Figure 9 DATA_TYPE field in the setup

7.2 Option menu

We will describe here every item in this menu, one by one, except for the "View Source" item that was described in Section 6.5.

7.2.1 Global considerations

As suggested by their location, one could imagine that the items of the "Option" menu deal with the configuration of the current frame. Actually, this is not always the case since several files are displayed inside one frame.

To be precise, the "Value as Attribute" and "Display Comments" items are associated to the frame (and thus change the behavior of the frame, just the frame and all the frame) while the other items are more or less file related and only apply to the currently selected file, ie to the file containing the currently selected node.

This last assumption has the strange implication that using the "Option" menu of one frame could modify the display in other frames (all those displaying the currently selected file) but will not modify the display in the whole current frame. It is actually even more complicated, as described in the following.

You know that each XML file is associated to a dtd. This dtd defines the possible node types in the XML file and their attributes among other things. The decision to display or to not display one type of elements or of attribute has thus to be associated to the dtd, not to the file. The tricky thing is that several files can share a common dtd (usually many ones). In that case, a change in the display configuration will actually modify several files and thus potentially even more frames displays.

On top of that, take care that a given element type, say "detelem", can be defined in several dtds (and actually is in the detector description database). Thus, when you modify the display of the "detelem" elements in one dtd, you don't modify the display of elements defined in other dtds and thus you don't change the display of all nodes of this type but only of those associated to the right dtd.

To make matters worse, XmlEditor has a bug here (since its author was not fully conscious of all that before writing this documentation...) and thus, the updating of frames other than the current one after a change in one of the items of the "Option" menu is not done. This leads to the following behavior : when you change something for a given dtd, the corresponding dialog box will be updated in every frame but not the display of the frames themselves. These will only be updated the next time you use one item of the "Option" menu in them. On the other hand, the opening of a new file (using the concerned dtd) inside one of these frames (via the opening of a reference) will be done using the new display policy which will definitely lead to some nasty things... To be fixed.

7.2.2 Value as attribute

This item is actually a checkbox that says whether the value of a given node should be displayed as a text subnode of this node or as an attribute named "Value" of this node. The default is to display values as attributes.

See Section 4.2 for further discussion concerning this point.

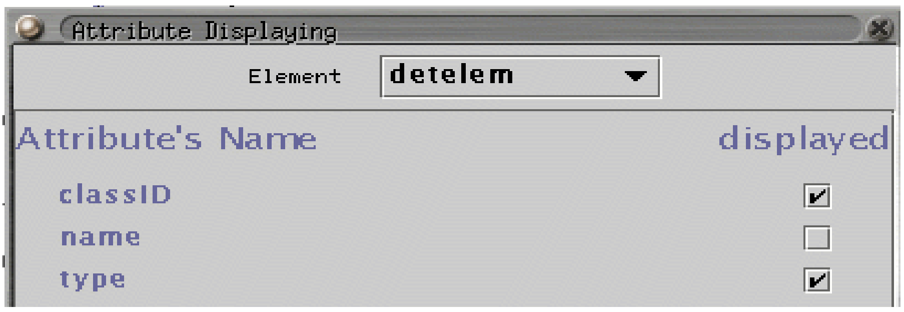
Note that a click on this item immediately updates every node inside the current frame.

7.2.3 Display Comments

This is also a checkbox that says whether comments should be displayed or not. The default is to not display them.

7.2.4 Attribute Displaying

This item opens a dialog box where you can choose for each element type which attribute will be displayed on the left part of the frame and which will not. Figure 10 shows a view of the dialog box, where the "detelem" element type is being edited.



Element: detelem

Attribute's Name	displayed
classID	✓
name	
type	✓

Figure 10 The Attribute Displaying dialog box

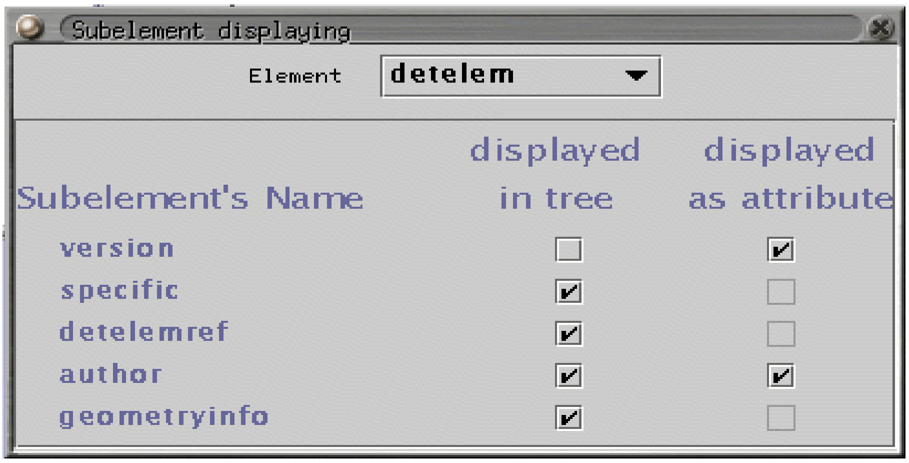
By default, every attribute is displayed except those used as name of the element type, as for the "name" attribute in Figure 10.

Note that a modification in this dialog box will be taken into account in the corresponding frame only at the closing of the box.

7.2.5 SubElement Displaying

This works more or less like Attribute displaying but there are two columns of options in the dialog box.

The first column says whether subelements of a given type should be displayed in the tree as children of elements of the selected type. Figure 11 shows the example of element type "detelem". Here all subelements are displayed in the tree except those of type "version".



Element: detelem

Subelement's Name	displayed in tree	displayed as attribute
version	☐	✓
specific	✓	☐
detelemref	✓	☐
author	✓	✓
geometryinfo	✓	☐

Figure 11 The SubElement Displaying dialog box

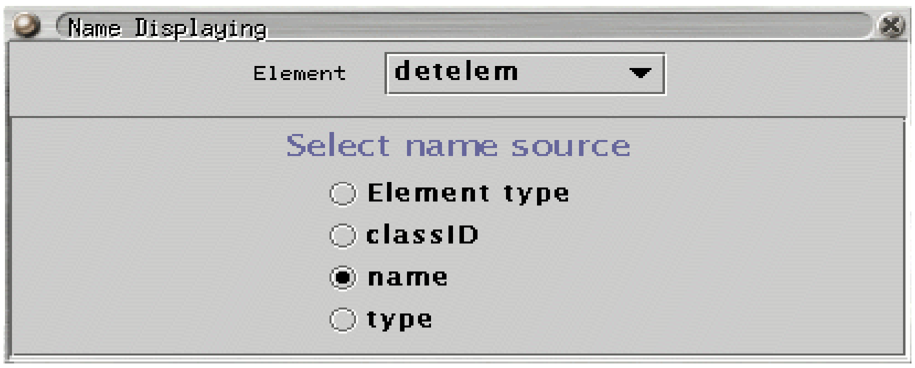
The second column says whether a given subelement type should be displayed as attribute or not. Figure 11 shows for example that the author subelements are displayed as attributes as well as in the tree.

Of course, the second column is only available for "attribute-like" elements. See Section 4.2 for further discussion on this matter.

The default configuration here is to display every node in the tree except for the attribute-like ones that are displayed only as attributes.

7.2.6 Element Name Displaying

This last item opens a dialog box where the user can, for each element type, choose the source of the name of the element. This source can be either one of the attributes of the element or its value or its type. Figure 12 shows this dialog for the "detelem" element type.



Element: detelem

Select name source

- Element type
- classID
- [✓ name]
- type

Figure 12 The Name Displaying dialog box

The default configuration is to use the "name" attribute if it exists, as shown here. If not, the "href" attribute is used (for references). If it neither exists the element type is used.

8.0 Source code documentation

This documentation is automatically generated from the source code (using javadoc) at compilation time. You'll find it in the doc subdirectory of the XmlEditor package. Just look at index.xml with a browser.

9.0 Known bugs and limitations

Due to its recent birth, the XmlEditor still has many limitations and a few known bugs. All of them are related in this list but other may appear soon. Please don't hesitate to report them to the LHCb computing group.

Here are the known bugs and limitations :

3. Cut and Paste and Drag and Drop functionality allow to add nodes that should not be accepted since they are not in conformity with the underlying dtd.
4. Required attributes can be deleted without being recreated
5. There is no way to get a kind of "report" of an XML file in a more understandable way than the XML itself
6. There is no way to distinguish an attribute that is not in conformity with the dtd
7. The default state of every configuring menu should be included into the setup file
8. There is no Save-As facility
9. An icon bar with the most useful options should be added
10. The frames are not always well redisplayed in case of a display configuration change, see Section 7.2.1 for details