Gaudi Object Description v2r0 - Documentation

Stefan Roiser

January 17, 2002

Contents

Chapter 1

Introduction

Gaudi Object Description is a set of tools for the description of the transient event data in the Gaudi framework. For this purpose the data will be described with XML-files that can be compiled into many different representations (e.g. C++, Python, Java, ...).

This XML-description can also be used to produce some information so that the data-objects described can be investigated from an external point with an introspection tool (see Gaudi Introspection package[1]).

Other advantages of using Gaudi Object Description are:

- *Uniform layout* The files produced with Gaudi Object Description will all have the same behavior and layout.
- *No redundancies* Normally one has to type e.g. in a C++-header-file many times the same information about an object (e.g. for the set- and get-functions, etc.). In Gaudi Object Description from one piece of information a lot of redundant information can be produced, which will lead to a shorter description of the objects.
- *Coding conventions* The files produced by Gaudi Object Description comply to the C++ coding conventions[1].
- *Documentation* From the object-descriptions it is also possible to produce source-code-documentation with tools like doxygen[2].

[1] http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DataDictionary/default.htm
[2] http://www.doxygen.org

Chapter 2

The Structure

The structure of Gaudi Object Description can be seen in Figure 2.1. The source for all the stuff that can be produced is a XML-file, which can be seen at the top of the figure. Together with some generic rules, this file will be parsed by a XML-parser and feeds an internal C++ meta-model.

The information stored in this internal meta-model will be enough to produce any kind of desired output, like the different back-ends for different languages or the information for the Run Time Type Information. Additionally to the different back-ends special rules can be applied which have only a meaning in conjunction with the language that will be produced by this special backend. For the time being the back-ends for the C++-header-files and the meta-information for runtime-type-information have been implemented. In the future there are more back-ends to come, like converters or other languages.

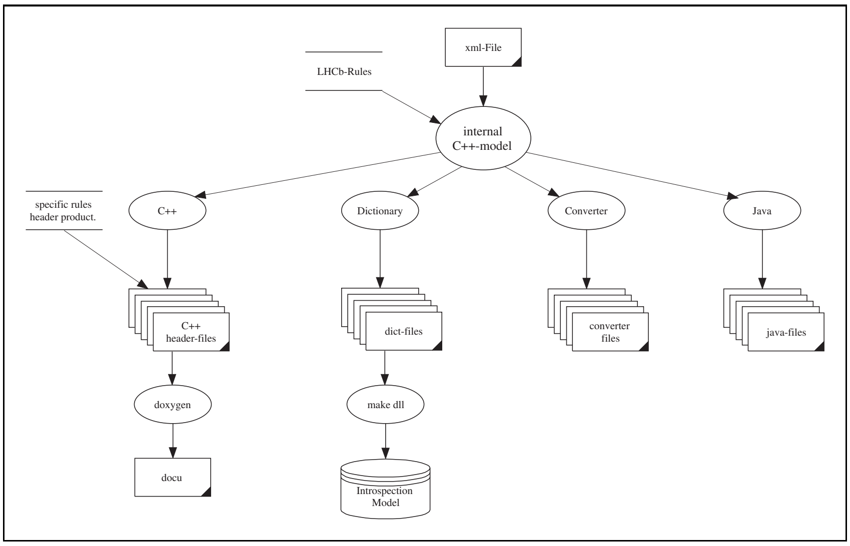

Figure 2.1: Overview of Gaudi Object Description

Chapter 3

Using Gaudi Object Description

3.1 Introduction to XML

The Gaudi Object Description language is based on XML. The main parts that are used for the description of Gaudi Objects with XML are elements and attributes. The elements define the syntactical order of the document, each element can have 0 or more attributes. Table 3.1 shows an element called <GAUDI> with an attribute 'version' which has the value '1.0'.

```
<GAUDI version='1.0'/>
```

Table 3.1: A simple XML-example

Table 3.1 also shows that the value of an attribute has to be assigned to it by '='. The value itself has to be enclosed either by single quotes (') or double quotes ("). We encourage people to use single quotes because the XML-files will be also used to produce C++-code and this will lead to less confusion, because most times one wants to use double quotes inside a C++ sequence.

The XML-example of table 3.1 is closed at the end with '/>'. But XML-elements can also embed other elements or text. Table 3.2 shows an example of an XML-element <GAUDI> which contains another element called <LHCB>, which contains some text. Note that the <GAUDI>-element is closed after the child-element with </GAUDI>.

```
<GAUDI version='1.0'>
  <LHCB> These are just a few characters </LHCB>
</GAUDI>
```

Table 3.2: A second simple XML-example

These two examples now show only well-formed XML, which means that they conform to the rules setup for XML by the World Wide Web Consortium (W3C)[2]. To achieve valid XML-documents the text has to conform some rules. In our case these rules are provided by a DTD (Document Type Definition). This DTD specifies the syntax of the XML-document.

The DTD used for Gaudi Object Description is called 'gdd.dtd' and is stored in 'xml_files/gdd.dtd' in the GaudiObjDesc-package. The DTD must reside in the same directory as the xml-file with the object-descriptions. During a make this copying will be done automatically. Only if one does not use make from the start, the DTD-file has to be copied manually to the location of the xml-file.

3.2 Invoking Gaudi Object Description tools

For the time being there are two possibilities to invoke the tools of the Gaudi Object Description package. One way is to call the tools from the command-line. The other way is to execute the tools automatically with the make-command inside a Gaudi-package. The second approach is most probably the more often used one and if one wants to do that there are some changes which have to be done inside the requirements-file of the package.

The usage-statements of the Gaudi Object Description tools can be found in appendix B.

3.2.1 Changes to the requirements file

In order to compile a Gaudi-package with Gaudi Object Description, several changes have to be done in the requirements-file (see table 3.3).

```
1 use GaudiObjDesc v2r*
2
3 document obj2doth <Package>Obj2Doth ../xml/<Package>.xml
4
5 document obj2dict <Package>Obj2Dict ../xml/<Package>.xml
6 library <Package>Dict ../dict/*.cpp
7 macro <Package>Dict_shlibflags "$(use_linkopts) $(libraryshr\_linkopts)"
```

Table 3.3: Changes to the requirements-file

In the first line of table 3.3 you can see the use-statement for GaudiObjDesc. The latest version for the moment is v2r0, but in order to also use backward-compatible versions it is advised to use v2r* instead. Line 3 describes the call of the fragment to produce the C++-header-files. This statement will be executed at make-time only once, as long as the xml-source-file is not changed.

Lines 5 to 7 describe the statements which have to introduced in order to produce the meta-information for GaudiIntrospection. Line 5 produces the cpp-files which will be compiled to a dll in line 6. Line 7 introduces some flags for the compilation.

In order to run one of these two tools, before executing the make-command, one has also to execute the setup because the path to these tools has to be set.

It is also possible to pass any command-line-argument (see appendix B) to the tools by setting the variable $(GODFLAGS) inside the requirements file.

3.3 Some general comments about the Gaudi Object Description syntax

3.3.1 Elements

In section 3.4 one can find all the elements of the Gaudi Object Description language. Each element comes with a short description and a list of possible sub-elements, which can be placed between its opening- and closing-tag. The elements itself are put between < and > and some have also special characters attached which have been borrowed from the Backus Naur Form (BNF). Their meanings are:

- () grouping of elements
- | or-relation
- ? zero or one occurance of the element
- * zero or more occurances of the element
- + one or more occurances of the element

3.3.2 Attributes

In section 3.4 one can also find the detailed explanation of all attributes attached to an element. Every attribute has a table with at least three elements (Required, Values, Default). Their meanings are:

- **Required** means that this attribute has to be provided when the parser runs through the XML-document, otherwise it will complain. The only exception when no value has to be provided is, when there is already a default-value.

- **Values** desribes all possible values for this attribute, if values is set to 'any' this means that any UTF-8 character (in our case ASCII-character) can be used.

- **Default** means that there is already a default value for this attribute. If one is happy with this value, it doesn't need to be set explicitly in the XML-file. The XML-parser will always look at the DTD and take this default-value if no otherone is provided.

- If **Fixed** is set to yes this means that there is also a default-value and this value will be taken by the XML-parser. Trying to set this attribute to another value will cause a complaint by the parser.

3.4 Detailed Gaudi Object Description syntax

In this section all elements and their corresponding attributes will be declared in depth.

3.4.1 Element <GDD>

• Subelements: (<IMPORT>* <PACKAGE>+) | (<IMPORT>* <CLASS>)

The XML-declaration defines, that there must not be more than one root-element. The <GDD>-element is the root-element of all Gaudi Object Description files. There are two possiblities of subelements. Either zero or more imports followed by one or more packages, or zero or more imports followed by one class.

Attribute — version

- Required: yes
- Values: any
- Default: 1.0

The version-attribute will be used in future versions of Gaudi Object Description to distinguish between different versions of files. For the moment it is set to '1.0' and should not be changed.

3.4.2 Element <IMPORT>

• Subelements: none

The <IMPORT>-element denotes that some additional information has to be included to the file. In the sense of C++ this would mean for example that an #inlude-statement or a forward-declaration of a class will be added to the file. In general you will not have to bother with import-statements because the Gaudi Object Description tools will take care for this in many places (e.g. non-simple types of attributes, arguments of methods, etc.). The import may occur in three different places, which are inside <GDD>, <PACKAGE> or <CLASS>. Depending on where the import-statements occurs, this scope will be taken (e.g. an import under <PACKAGE> will occur in every class. For the time being <GDD> and <PACKAGE> define the same scope).

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the file to be imported. If this file is inside the Gaudi-, LHCb- or CLHEP-area, you will also not have to bother about the path to this file, because it will be retrieved for you automatically. If you want to import a file which by chance has the same name as a file which would be retrieved automatically and you want to use the other one, you have to provide the full path to it.

Attribute — std

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Std stands for 'standard import'. This means that in case of a language where a difference between a normal import and an import of a standard import can be made, this will be taken into account. (e.g. C++ will do an '#import<filename>).

Attribute — soft

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Soft import means that the file itself will not be included. Instead of that the import will be made known to the compiler. In the case of C++ this would result in a forward declaration.

3.4.3 Element <PACKAGE>

- Subelements: <IMPORT>* <CLASS>*

The <PACKAGE>-element is the container for all classes. Every Import that is printed inside the package-element will again occur inside every class.

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the package.

3.4.4 Element <CLASS>

- Subelements: <DESC>? <BASE>* <IMPORT>* <CONSTRUCTOR>* <DESTRUCTOR>* <METHOD>* <ATTRIBUTE>* <RELATION>*

The <CLASS>-element is the heart of the Gaudi Object Description language. This is where all the essential class-information will be.

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the class.

Attribute — author

- Required: yes
- Values: any
- Default: none

The author of the class.

Attribute — desc

- Required: yes
- Values: any
- Default: none

This is a short description of the class, and should not exceed one sentence. If more explanation is needed one can always put it inside the <DESC>-subelement (see section 3.4.5).

Attribute — filename

- Required: no
- Values: any
- Default: none

Filename should not be used for the description of Gaudi objects. This attribute was invented, because the same DTD is also used to build up a database of all classes and their location in the Gaudi-framework.

Attribute — id

- Required: no
- Values: any
- Default: none

The id-attribute defines the Class-ID. If the id is set to a value, it will be assumed that this class is an event-class which will have its representation in the Gaudi-stores. Setting this attribute to a value also triggers the production of a definition of this class-id inside the class and methods to retrieve it.

Attribute — templateVector

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

All event-classes derived from ContainedObject will have a templated vector set up. If one does not want this, the value of templateVector has to be set to FALSE.

Attribute — templateList

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

All event-classes derived from ContainedObject will have a templated list set up. If one does not want this, the value of templateList has to be set to FALSE.

3.4.5 Element <DESC>

- Subelements: #PCDATA

The <DESC>-element is one of two elements, the other one is <CODE> (see section 3.4.12), which allow simple text between the opening- and the closing-tag. The text which is set between these two tags will be taken as is and put into the comment which describes the event class at the top of the header-file. Take care that necessary characters have to be escaped (see section 4.2).

Attribute — xml:space

- Required: yes
- Values: default, preserve
- Default: preserve
- Fixed : yes

The xml:space-attribute is set to Fixed, so any change to this attribute will result in a complaint of the compiler. The reason for setting this attribute to Fixed is that otherwise some XML-editors cannot cope with the preserving of spaces and line-feeds.

3.4.6 Element <BASE>

- Subelements: none

The <BASE>-element describes the baseclass of the current object.

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the baseclass.

Attribute — virtual

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Has to be set to TRUE if the class is derived virtually.

Attribute — access

- Required: yes
- Values: PUBLIC, PRIVATE, PROTECTED
- Default: PUBLIC

Defines the accessor to the baseclass.

3.4.7 Element <CONSTRUCTOR>

- Subelements: <ARG>* <CODE>?

If a special constructor is needed it can be done with this element. If this special constructor takes no arguments it will replace the standard constructor which would have been set up otherwise. The automatically produced standard constructor sets all attributes to its default-values, which is the init-attribute of the <ATTRIBUTE>-element (see section 3.4.13) or 0 resp. 0.0 for numbers which do not have an init-attribute. The special constructor never uses default-values and all initialization-values have to be set explicitly.

Attribute — desc

- Required: yes
- Values: any
- Default: none

A short description of the special constructor.

Attribute — argList

- Required: no
- Values: any
- Default: none

A list of arguments separated by commas (,), containing at least a pair of type and name with a possible 'const' in front of the type. The argList-argument is one of two alternatives to describe the arguments of a constructor, destructor or method. If the type is not a simple type (e.g. int, char) the argument will be treated as 'const argument&'. If one wants to avoid that or have some special treatment, the use of the <ARG>-subelement (see section 3.4.10) should be considered.

Attribute — argInOut

- Required: no
- Values: any
- Default: none

Not used in this version of Gaudi Object Description and will be implemented in a future release.

3.4.8 Element <DESTRUCTOR>

- Subelements: <ARG>* <CODE>?

Similarly to special constructors one can also specifiy special destructors, which behave the same as special constructors. If no special destructor with zero arguments is defined, one will be setup automatically. Like for the special constructor the possible sub-elements are for arguments and code.

Attribute — desc

- Required: yes
- Values: any
- Default: none

A short description.

Attribute — argList

- Required: no
- Values: any
- Default: none

Argumentlist behaving the same as for the <CONSTRUCTOR>-element (see section 3.4.7).

Attribute — argInOut

- Required: no
- Values: any
- Default: none

Not implemented yet.

3.4.9 Element <METHOD>

- Subelements: <ARG>* <RETURN>? <CODE>?

Most of the methods will be produced automatically (e.g. the set- and get-methods for arguments, handling of classID, etc.). If there is the need to specify special methods which are not covered by these automatic procedures one can do that by using the <METHOD>-element. Special methods behave similarly to special constructors and destructors. The main difference is, that one can also define a return-value.

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the method.

Attribute — desc

- Required: yes
- Values: any
- Default: none

A short description.

Attribute — access

- Required: yes
- Values: PUBLIC, PROTECTED, PRIVATE
- Default: PUBLIC

The handling of this attribute is not yet implemented. In future releases the access-attribute specifies where the method should be placed inside the class. For the time being all methods will be put into the public area.

Attribute — const

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Defines wether the method should be const or not.

Attribute — virtual

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Defines wether the method should be virtual or not. For the time being pure virtual methods are not supported.

Attribute — static

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Defines wether the method is static.

Attribute — inline

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Left-over from a previous version and should not be used anymore.

Attribute — friend

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Defines wether this is a friend-method.

Attribute — type

- Required: yes
- Values: any
- Default: void

The type defines the return-type of the method. If no return-type is set and there is also no <RETURN>-subelement (see section 3.4.11), 'void' will be assumed.

Attribute — argList

- Required: no
- Values: any
- Default: none

The argList behaves completely similar to the argList of the <CONSTRUCTOR>-element (see section 3.4.7).

Attribute — argInOut

- Required: no
- Values: any
- Default: none

Not used in this version.

3.4.10 Element <ARG>

- Subelements: none

The <ARG>-element defines an argument of a special constructor, destructor or method. The order of the <ARG>-elements will be taken from the XML-file. In principal this element has only to be used when some special treatment of the arguments is needed.

Attribute — type

- Required: yes
- Values: any
- Default: none

The type of the argument. If the argument is not a simple type it will be treated as 'argument&' to avoid unnecessary copying of objects in the code.

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the attribute.

Attribute — const

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Defines wether the attribute is const.

Attribute — inout

- Required: yes
- Values: INPUT, BOTH
- Default: INPUT

The inout-argument will be in most cases the reason for using the <ARG>-element instead of the argList-attribute. Setting this attribute to 'BOTH' will treat the argument as an input-output-argument, which means that the reference of argument will be taken.

3.4.11 Element <RETURN>

- Subelements: none

The <RETURN>-element has the same as meaning as the type-attribute of <METHOD> (see section 3.4.9), which is the definition of the return-type of a method.

Attribute — type

- Required: yes
- Values: any
- Default: none

Type of the return-value.

Attribute — const

- Required: yes
- Values: TRUE, FALSE
- Default: FALSE

Defines wether the return-value is const.

3.4.12 Element <CODE>

- Subelements: #PCDATA

The <CODE>-element is one of two elements which allow normal text between the opening- and the closing-tag. The text between these two tags will be taken as is, except the escaping of characters (see section 4.2).

Attribute — xml:space

- Required: yes
- Values: default, preserve
- Default: preserve
- Fixed : yes

The xml:space-attribute is set to Fixed, so any change to this attribute will result in a complaint of the compiler. The reason for setting this attribute to Fixed is that otherwise some XML-editors cannot cope with the preserving of spaces and line-feeds.

3.4.13 Element <ATTRIBUTE>

- Subelements: none

<ATTRIBUTE>-elements are beside the <RELATION>-elements (see section 3.4.14) the meat of the Gaudi Object Descriptions. From these elements most of the automatic code will be produced.

Attribute — type

- Required: yes
- Values: any
- Default: none

The type of the attribute. If the type is not a simple one, Gaudi Object Description will try to import automatically a description of this type so the compiler can deal with it.

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the attribute. Although the Gaudi C++ coding conventions[1] define members of classes to be defined with a preceding 'm_', this is not necessary in Gaudi Object Description, as these letters will be added to the string whenever necessary.

Attribute — desc

- Required: yes
- Values: any
- Default: none

A short description of the member.

Attribute — init

- Required: no
- Values: any
- Default: 0, 0.0 for numbers, none otherwise

The init-attribute defines the initial value of the member. The default for numbers (e.g. int, float), will be set to 0 respectively 0.0. If another value is specified it will be taken. The init-attribute can also be used to specify initial values for non-numbers (e.g. char, std::string). Take care that the init-values will only be used for the default-constructor that is setup automatically by Gaudi Object Description, for special constructors all the values have to be set by hand.

Attribute — access

- Required: yes
- Values: PUBLIC, PROTECTED, PRIVATE
- Default: PRIVATE

The handling of this attribute is not yet implemented. In future releases the area inside the class where the member resides will be specified with the access-attribute. For the time being all class-members will be put in the private area.

Attribute — setMeth

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

Defines wether a setter-method for this member should be created
(e.g. 'void setOscillationFlag(bool value);'). The implementation will also be inline.

Attribute — getMeth

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

Defines wether a getter-method for this member should be created (e.g.'bool oscillationFlag() const;'). The implementation will also be inline.

3.4.14 Element <RELATION>

- Subelements: none

Besides the <ATTRIBUTE>-element (see section 3.4.13) the <RELATION> is also a very important element. <RELATION>-elements handle the connection to other objects in the Gaudi stores. For this purpose the concept of SmartRef and SmartRefVector will be used.

Attribute — type

- Required: yes
- Values: any
- Default: none

The type of the relation. Only the type of the relation is needed. The SmartRef resp. SmartRevVector will be added whenever necessary.

Attribute — name

- Required: yes
- Values: any
- Default: none

The name of the relation. The string 'm_' (conforming to the Gaudi C++ Coding Conventions[1]) will be inserted whenever necessary.

Attribute — desc

- Required: yes
- Values: any
- Default: none

A short description of the relation.

Attribute — access

- Required: yes
- Values: PUBLIC, PROTECTED, PRIVATE
- Default: PRIVATE

The handling of this attribute is not yet implemented. In future releases the area inside the class where the relation resides will be specified with the access-attribute. For the time being all class-members will be put in the private area.

Attribute — multiplicity

- Required: yes
- Values: 1, M, m, N, n
- Default: 1

The multiplicity-attribute is specific to the <RELATION>-element. It defines the relation to the other object. The default is '1' which will set a 'SmartRef'. All the other possible values 'M,m,N,n' are equal and define a one-to-many-relation which will end up in a SmarRefVector. There is also a difference in the amount of methods produced for the relation. While a one-to-one-relation produces the setter-, getter- and clear-method, for a one-to-many-relation also the addTo- and removeFrom-method will be created if desired.

Attribute — setMeth

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

Defines wether a set-method should be created
(e.g. 'void setDecayMCVertices(const SmartRefVector<MCVertex>& value);').
The implementation will also be inline.

Attribute — getMeth

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

Defines wether a get-method should be created
(e.g. 'const SmartRefVector<MCVertex>& decayMCVertices() const;'). The implementation will also be inline.

Attribute — addMeth

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

Defines wether addTo-methods should be created
(e.g. 'void addToDecayMCVertices(MCVertex* value);' and
'void addToDecayMCVertices(SmartRef<MCVertex>& value);). The implementation will also be inline.

Attribute — remMeth

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

Defines wether removeFrom-methods should be created
(e.g. 'void removeFromDecayMCVertices(MCVertex* value);' and
'void removeFromDecayMCVertices(SmartRef<MCVertex>& value);'). The implementation will also be inline.

Attribute — clrMeth

- Required: yes
- Values: TRUE, FALSE
- Default: TRUE

Defines wether a clear-method should be created
(e.g. 'void clearDecayMCVertices();'). The implementation will also be inline.

Chapter 4

Tips & Tricks

4.1 Editing and producing xml-files

Although the xml-files can also be written with a normal text-editor, it is highly recommended that a xml-editor should be used for producing and editing the xml-files.

One of these Xml-Editors, written by Sebastien Ponce, is also available as a package from the LHCb-cvs-server. The package is called *'Det/XmlEditor'* and the current release is *v4r2*. The advantage of this xml-editor is that it is specially tailored to the handling of Gaudi objects.

4.2 Escaping of characters

In some special cases the text cannot be put into the XML-file as is, because special characters, which are reserved for the steering of XML, need to be escaped. These characters and their escaping can be found in table 4.1. With some XML-editors this escaping of characters will be done automatically, so the user does not have to bother with it.

XML-character	Escape-sequence
&	&
<	<
>	>
'	'
"	"

Table 4.1: XML-characters to be escaped

The first three characters (<, > and &) have to be escaped no matter where in the XML-document. If not, a complaint from the XML-parser will be the result. The two latter ones (' and ") only have to be escaped if they are used as a value for an argument, and only if they both are used at the same time.

Arguments of XML-elements have to be surrounded by either ' or ". So if the other character is used inside the value it has to be escaped.

4.3 Additional information

General information about the Gaudi-framework can be found at:

http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/Gaudi/

Additional information about the Gaudi Object Description package can be found at:

http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DataDictionary/default.htm

Appendix A

The Syntax

A.1 The elements

The syntactical tree of the Gaudi data definition language can be found in Table A.1. The corresponding arguments for each element in this tree are shown in Table A.2. A more precise explanation of the elements and their attributes can be found in section 3.4

The <DESC>- and <CODE>-element are the only two elements in the Gaudi Object Description language which can contain normal text between their opening and closing tag.

```
<GDD>         := (<IMPORT>* <PACKAGE>+) | (<IMPORT>* <CLASS>)
<PACKAGE>     := <IMPORT>* <CLASS>*
<CLASS>       := <DESC>? <BASE>* <IMPORT>* <CONSTRUCTOR>*
                 <DESTRUCTOR>* <METHOD>* <ATTRIBUTE>*
                 <RELATION>*
<DESC>        := #PCDATA
<CONSTRUCTOR> := <ARG>* <CODE>?
<DESTRUCTOR>  := <ARG>* <CODE>?
<METHOD>      := <ARG>* <RETURN>? <CODE>?
<CODE>        := #PCDATA
```

Table A.1: Syntax of Gaudi Object Description

A.2 The attributes

Element	Attribute	Value	Required	Default	Description
GDD	version	text	yes	1.0	version of the xml-file
IMPORT	name	text	yes	-	name of import-file
	std	TRUE, FALSE	yes	FALSE	define standard import
	soft	TRUE, FALSE	yes	FALSE	define soft-import
PACKAGE	name	text	yes	-	name of package
CLASS	name	text	yes	-	class-name
	author	text	yes	-	class-author
	desc	text	yes	-	short description
	filename	text	no	-	filename of class[1]
	id	text	no	-	class-id
	templateVector	TRUE, FALSE	yes	TRUE	vector of class
	templateList	TRUE, FALSE	yes	TRUE	list of class
DESC	xml:space	default, preserve	yes	preserve	fixed, can't be changed
BASE	name	text	yes	-	name of base-class
	virtual	TRUE, FALSE	yes	FALSE	derived virtually
	access	PUBLIC, PROTECTED, PRIVATE	yes	PUBLIC	accessor to base-class
CONSTRUCTOR	desc	text	yes	-	description of constructor
	argList	text	no	-	optional argument-list
	argInOut	text	no	-	not used in this version

[1]Is used to setup the database of packages, should not be used in normal xml-files

Element	Attribute	Value	Required	Default	Description
DESTRUCTOR	desc	text	yes	-	description of constructor
	argList	text	no	-	optional argument-list
	argInOut	text	no	-	not used in this version
METHOD	name	text	yes	-	method-name
	desc	text	yes	-	method-description
	access	PUBLIC, PROTECTED, PRIVATE	yes	PUBLIC	accessiblity
	const	TRUE, FALSE	yes	FALSE	const-method
	virtual	TRUE, FALSE	yes	FALSE	virtual-method
	static	TRUE, FALSE	yes	FALSE	static-method
	inline	TRUE, FALSE	yes	FALSE	inline-method (obsolete)
	friend	TRUE, FALSE	yes	FALSE	friend-method
	type	text	yes	void	return-type of method
	argList	text	no	-	optional argument-list
	argInOut	text	no	-	not used in this version
ARG	type	text	yes	-	argument-type
	name	text	yes	-	argument-name
	const	TRUE, FALSE	yes	FALSE	const-argument
	inout	INPUT, BOTH	yes	INPUT	input- or input-output-argument
RETURN	type	text	yes	-	return-type
	const	TRUE, FALSE	yes	FALSE	const-type
CODE	xml:space	default, preserve	yes	preserve	fixed, can't be changed

Element	Attribute	Value	Required	Default	Description
ATTRIBUTE	type	text	yes	-	attribute-type
	name	text	yes	-	attribute-name
	desc	text	yes	-	attribute-description
	init	text	no	-	initial value of attribute
	access	PUBLIC, PROTECTED, PRIVATE	yes	PRIVATE	accessiblity
	setMeth	TRUE, FALSE	yes	TRUE	create set-method
	getMeth	TRUE, FALSE	yes	TRUE	create get-method
RELATION	type	text	yes	-	relation-type
	name	text	yes	-	relation-name
	desc	text	yes	-	relation-description
	access	PUBLIC, PROTECTED, PRIVATE	yes	PRIVATE	accessiblity
	multiplicity	1, M, m, N, n	yes	1	relation to other class[2]
	setMeth	TRUE, FALSE	yes	TRUE	create set-method
	getMeth	TRUE, FALSE	yes	TRUE	create get-method
	addMeth	TRUE, FALSE	yes	TRUE	create addTo-method
	remMeth	TRUE, FALSE	yes	TRUE	create removeFrom-method
	clrMeth	TRUE, FALSE	yes	TRUE	create clear-method

Table A.2: Arguments for Gaudi Object Description

[2] $(M \equiv m \equiv N \equiv n)$

Appendix B

Usage statements

The following two sections show the usage statements for the two tools which are currently implemented in the Gaudi Object Description package. In order to use the tools inside a cmt-requirements-file the environement variable $(GAUDIOBJDESCROOT) should be set to the directory which defines the version of the package. Normally this variable is set when 'setup.bat' or 'source setup.csh' is executed.

Care has to be taken that all xml-files have the extension '.xml' and all directories that are passed as arguments end with a slash or back-slash.

B.1 GODWriteCppHeader.exe

```
Usage: GODWriteCppHeader.exe [-h] [-v] [-i] [-o [path]] [-x [path]] xml-file(s)
Produce .h-files out of xml-files

   -h            display this help and exit
   -v            display version information and exit
   -i            add additional file-package-information from './AddImports.txt'
   -o [path]     define possible output-destination with following precedence
                    -o path     use 'path'
                    -o          use environment-variable 'GODDOTHOUT'
                     default    use local directory
   -x [path]     define location of 'GaudiCppExport.xml' which holds information
                 about include-file<->package dependencies, with this precedence
                    -x path     use 'path'
                    -x          use environment-variable 'GODXMLDB'
                     default    use '$(GAUDIOBJDESCROOT)/xml_files'
   xml-file(s)   xml-file(s) to be parsed (must have extension '.xml')
```

B.2 GODWriteCppDict.exe

```
Usage: GODWriteCppDict.exe [-h] [-v] [-o [path]] [-x [path]] xml-file(s)
Produce .cpp-files for the Gaudi Dictionary

   -h             display this help and exit
   -v             display version information and exit
   -i             add additional file-package-information from './AddImports.txt'
   -o [path]      define possible output-destination with following precedence
                     -o path     use 'path'
                     -o          use environment-variable 'GODDICTOUT'
                      default    use local directory
   -x [path]      define location of 'GaudiCppExport.xml' which holds information
                  about include-file<->package dependencies, with this precedence
                     -x path     use 'path'
                     -x          use environment-variable 'GODXMLDB'
                      default    use '$(GAUDIOBJDESCROOT)/xml_files'
   xml-file(s)    xml-file(s) to be parsed (must have extension '.xml')
```

List of Tables

List of Figures

Bibliography

[1] Callot O.; Revised C++ coding conventions; 30 April 2001; http://weblib.cern.ch/format/showfull?uid=1301345_1951&base=LHBLHB&sysnb=000041

[2] Bray T., Paoli J., Sperberg-McQueen C.M., Maler E.; Extensible Markup Language (XML) 1.0 (Second Edition); W3C Recommendation 6 October 2000; http://www.w3.org/TR/2000/REC-xml-20001006