Error Parser Manuel

Daniel Prévost

Error Parser Manuel

by Daniel Prévost

Table of Contents

Chapter 1. Installation

The installation procedure of the software itself is relatively simple if you use the pre-packaged version of the software (RPM, Inno setup, etc.) or if you build the software from scratch and use make install.

There are some potential complications however in some cases. For example: this software requires libxml2 - on Unix/Linux, libxml2 will usually already be installed (or you can find packages that will do it painlessly). On Windows, this is often not the case - the Inno Setup installation can installed the run-time environment needed by Error Parser but you may prefer to install everything yourself (this will include the development environment if you want to integrate xml in your application).

This section will look into installation issues in some details.

1.1. The xml catalog on Linux/Unix

The two DTD (Document Type Definition) for Error Parser can be installed in the global xml catalog (usually /etc/xml/catalog). Is this needed? Depends on your exact situation.

A DTD can be identified using a PUBLIC clause (in the xml file), for example:

```
<!DOCTYPE errorlist PUBLIC
   "-//Error Parser project//DTD Error Parser XML V1.2//EN"
    "http://errorparser.sourceforge.net/xml/errorParser12.dtd">
```

In this case the DTD will be downloaded from the Error Parser web site.

You can also use SYSTEM instead:

```
<!DOCTYPE errorlist SYSTEM
   "/usr/local/share/xml/errorParser/errorParser12.dtd">
```

Using SYSTEM is perfectly ok if you have a single developer per workstation. If development is done on a shared box however, using the PUBLIC clause is one way to make sure that every developer uses the same copy of the DTD.

However you don't want to download the DTD every time you use the software and in this situation, you'll want to add the Error Parser catalog to the main xml catalog. The program used to do this (xmlcatalog) is included in the libxml2 package. Furthermore a bash script is provided by Error Parser

(possibly installed in /usr/local/share/xml/errorParser/install_catalog.sh if Error Parser is installed in /usr/local). You'll need to edit this script to tailor it for your installation.

Recommendation: using the SYSTEM clause is perfectly ok and there is likely no real needs to use the xml catalog.

1.2. The xml catalog on Windows

The previous section already discussed the use of PUBLIC or SYSTEM clause for identifying a DTD. The biggest difference between Microsoft Windows and for example Linux, is the absence of a centralized way of managing xml DTD and Schemas - each software manages its XML on its own. It makes even more sense to use SYSTEM here.

If you still want to install an xml catalog on Windows, the only thing you need to know is that libxml2 uses the environment variable XML_CATALOG_FILES to find the main xml catalog.

There is an example of a main catalog in the DTD directory (main_catalog_win32.xml). If the main catalog already exists you can use the script DTD\install_catalog.vbs. In both cases you will need to edit the files to tailor them to your installation.

1.3. libxml2 on Windows

You can find binaries for the complete libxml2 package and other associated packages (libxslt, iconv, etc.) here (http://www.zlatkovic.com/libxml.en.html). You will need the full package if you build Error Parser from scratch.

As mentioned previously, the Error Parser package will give you the option of installing the run-time environment needed to run errorParser.exe. This include iconv.exe, iconv.dll, libxml2.dll and xmlcatalog.exe. Do not install them if you're installing the complete libxml2 package.

Chapter 2. Building the Software

The only prerequisite software needed to build and run Error Parser is the library libxml2 (the Gnome project xml parser). If it is not installed on your system, you can download it at xmlsoft.org (http://xmlsoft.org/downloads.html).

2.1. Building on Linux/Unix from the repository

To retrieve the latest version of vdsf, use the following command:

```
svn export https://errorparser.svn.sourceforge.net/svnroot/errorparser/trunk my_local_dir
```

Of course, this is a generic command, you can omit the sub-directory `trunk` to retrieve everything (furthermore, if `my_local_dir` is left empty, the current directory will be used).

> **Warning**
>
> The code in the repository is not always stable, be careful.

The next step is to run the shell script autogen.sh (in the trunk directory). This will generate everything you need (the makefiles, the configure script). The script will also run configure for you but you might want to rerun it again if you want to change its default options.

The remaining instructions are identical to the instructions for building the software from a tar file.

2.2. Building on Linux/Unix from the tar file

Step by step instructions to build the package from a tar file:

- Run configure (./configure). Here is the list of options that you might want to use to tailor the package to your needs:
  - --prefix=PREFIX To change the default installation directory (default is /usr/local). Example: --prefix=/usr
- make
- make check (to run the test suite - optional)
- make install

2.3. Building on Windows from the repository

If you want to retrieve the software directly from the repository, you will need a subversion client for Windows. Tortoise (http://tortoisesvn.tigris.org/) is one possible choice. The repository is:

```
https://errorparser.svn.sourceforge.net/svnroot/errorparser/trunk
```

> **Warning**
>
> The code in the repository is not always stable, be careful.

The remaining instructions are similar to the instructions for building the software from a zip file.

2.4. Building on Windows from a zip file

Very simple, to build the software, just use the workspace file parser\parser.dsw. If you use a modern version of Visual Studio, it will likely ask you to convert the workspace.

One additional note: you will need to add the directory for the header files for libxml2 to the list of directories in Visual Studio. For VS 6, you can set this in Options (in the Tools menu).

To run the tests, from a command shell run cscript RunTests.vbs (in the tests directory). You will also need Gnu diff (http://gnuwin32.sourceforge.net/packages/diffutils.htm) GNU diff for Windows.

To build the package, you will need Inno Setup (http://www.jrsoftware.org/isinfo.php). The Inno Setup file to open is installation\errorParser.iss.

Chapter 3. User Guide

3.1. Introduction

The program errorParser is very easy to use. From the command line just enter:

```
errorParser -o option_file error_file
```

where `option_file` and `error_file` are the names of two xml files specifying the options and all the data associated with the errors. The real complexity is of course in the xml files themselves. There are examples of these files in the source directory and their syntax is explained below.

3.2. The xml error file

The xml file starts with the usual xml header (version of xml, encoding of the file, the location of the DTD, etc.). If you're not familiar with xml and want more information on this, you might want to read additional documentation, for example the website of the W3 Schools (http://www.w3schools.com/xml/default.asp) .

Each of the xml tag specific to Error Parser is explained in the list below. Unless specified, each tag can only appear once (for each of its parent) and in the order listed here. Example: the <years> tag must be the first sub-tag of <copyright> and must be unique - however, if you have more than one <copyright> tag, each will have its <years> sub-tag.

- <errorlist version="0.3">

  This tag starts the document itself. The version attribute is a string of your choosing and is there to help you synchronize your work (to make sure that the output files match the version of your software, for example). This attribute is optional.

- <copyright_group>

  This is an optional tag. You use it if you want to generate copyright information in the output files. This tag has only one sub-tag, <copyright>, which can be repeated multiple times (if you have multiple authors with possibly different licenses).

  Note: from an xml point of view, this tag is unneeded - the <copyright> tag could be used directly without this "dummy" container. However, the code of errorParser is simplify because of it, so...

- <copyright>

Allows you to enter copyright information in the output files. It has multiple sub-tags. As mentioned previously, multiple <copyright> tags are allowed.

- <years>

  The interval of years for the current copyright notice.

- <authors>

  The name(s) of the owner(s) of the copyright.

- <notice>

  The text of the license. More exactly, a paragraph of the license. Use multiple <notice> if your license has multiple paragraphs. You must have at least one <notice> for each <copyright>.

- <errgroup>

  <errgroup> is used to organize your errors in groups, if you so desired (for example, a group of i/o errors, a group of network errors, etc). You can have multiple <errgroup>, you must have at least one.

  If you don't want to use this feature, only use a single group and do not include the <errgroup_ident> sub-tag.

- <errgroup_ident>

  This sub-tag of <errgroup> is optional. The sub-tags associated with this tag are used to generate comments in your output header file (and will be used to generate the docbook documentation eventually).

  You can have multiple <errgroup_ident> - one for each language supported by your application. This is done with an optional attribute, xml:lang. If the xml:lang attribute is missing the default attribute will be used (currently 'en' but you can modify the shipped DTD to use a different default tailored for your needs).

  Note: the parser will use the first <errgroup_ident> matching the language selected in the option file. If the selected language is not found, the first <errgroup_ident> is used.

  - <errgroup_name>

A name for this group of errors. For example, Network errors.

- <errgroup_desc>

  Description of this group of errors. You must have at least one <errgroup_desc> but you can have multiple ones, as needed. Each will be mapped to a paragraph.

- <error>

  This tag is also a sub-tag of <errgroup>. You need at least one <error> for each <errgroup> but you can have as many as needed.

  - <errnumber>

    The numerical value of the error code.

  - <errname>

    The generic name of the error. This name is going to be used to generate either an enum entry or a #define in the output header file.

    Note: if you add a prefix to your errors (to avoid namespace conflicts), it is recommended to leave the prefix out. For example, VDS_OBJECT_IS_DELETED would become <errname> OBJECT_IS_DELETED </errname> - the prefix can be added by specifying it in the option file.

  - <message_group>

    This tag is used to group the error messages and the documentation for the error - which are encapsulated in the <message> tag. Multiple <message> tags are supported (you must have at least one), one for each supported language.

    Note: from an xml point of view, this tag is unneeded - the <message> tag could be used directly without this "dummy" container. However, the code of errorParser is simplify because of it, so...

  - <message>

    Container for both the error message and the documentation for this error. This tag has an optional attribute, xml:lang.If the xml:lang attribute is missing the default attribute will be used (currently ’en’ but you can modify the shipped DTD to use a different default tailored for your needs).

You must have at least one <message>, multiple ones are supported (one per language used in your application).

Note: the parser will use the first <message> matching the language selected in the option file. If the selected language is not found, the first <message> is used.

- <errormsg>

  The text of the error message that will be retrieved by the function *yourprefix_ErrorMessage()*.

  Note: the inclusion of some characters ('%' for example) or chain of characters (escape sequences) might be problematic. You can control how to handle these occurrences by setting the appropriate options in the option file.

- <errordoc>

  Each <errordoc> is a paragraph of the documentation. Therefore, you can have multiple <errordoc>. You must have at least one.

3.3. The xml option file file

The number of potential options to the program errorParser was getting quite large and it made more sense to put all these options in a configuration file (written in xml). These options are described below.

- <options>

  The root element.

- <enumname>

  Optional. The name of the enum. If absent, #define will be generated instead.

- <header_name_dir>

  Optional. The name of the directory where the error header file is to be created. If absent, the current directory will be used.

Alternatively, you can put the directory name (or part of the directory name) directly in the <header_name> tag. But if you do, the "guard" will include that directory name (guard: the small preprocessor code use to avoid including the same header file twice).

Example: the output file name is *my_dir/my_sub_dir/my_sub_project/errors.h* and you want to include *my_sub_project* in the guard. The option file would look like:

```
<header_name_dir>my_dir/my_sub_dir</header_name_dir>
<header_name>my_sub_project/errors.h</header_name>
```

The generated guard would look like this:

```
#ifndef MY_SUB_PROJECT_ERRORS_H
#define MY_SUB_PROJECT_ERRORS_H
...
#endif /* MY_SUB_PROJECT_ERRORS_H */
```

- <header_name>

  The name of the header file for errors. See the previous discussion (<header_name_dir>) for more details.

- <errmsg_dir>

  Optional. The name of the directory where the header file and "C" file (for the code to retrieve the error messages) will be created. If absent, the current directory will be used.

  See the previous discussion (<header_name_dir>) for more details.

- <errmsg_header_name>

  The name of the header file for the code to retrieve the error messages. See the previous discussion (<header_name_dir>) for more details.

- <errmsg_c_name>

  The name of the "C" file for the code to retrieve the error messages. See the previous discussion (<header_name_dir>) for more details.

- <prefix_error>

The prefix to be used for the error codes (enum or #define).

- <prefix_variable>

  The prefix to be used for all variables in the code to retrieve the error messages.

- <err_msg>

  This tag has no content but its attributes are important - they define the allowed characteristics (and the transformations, if needed of the error messages defined in the xml error file (<errormsg>).

  The problem: the generated C strings will be use, eventually, by standard C/C++ libraries and some sequence of characters could have some bad side effects (possibly some security issues?). The most problematic issue is likely the %symbol - it is interpreted differently by libc (printf()) and the iostream library (cout).

  The attributes:

  - allow_escapes

    An enumeration with two values, yes and no. The default is no, escape sequences are not permitted (a more fine grain approach could be written if the need arise).

  - allow_quotes

    An enumeration with two values, yes and no. The default is yes, quotes (’ and ") are allowed (and will be escaped as needed).

  - percent

    The text to use to replace the % symbol. No default. I recommend the textual "percent".