



GEG4019
Undergraduate Research Project
Geospatial Metadata Visualization

by Julien McArdle

GEG 4019

Undergraduate Research Project

Geospatial Metadata Visualization

Supervisor	Dr. Sawada
Second Reader	Dr. Bannari
Student	Julien McArdle
Student n.	3270515

Acknowledgments

I wish to express my sincerest thanks to Dr. Sawada for presenting me with the idea that became this project, as well as my parents for their incredible and never-ending support.

Globe cover image by Barun Patro
Reproduced under license.
<http://www.barunpatro.com/>

Table of Contents

Section 1: <i>Introduction</i>	5
1.1 Abstract	6
1.2 A Note on Writing Conventions	6
1.3 Introduction	7
Section 2: <i>Metadata Parser</i>	8
2.1 Metadata Parser Introduction	9
2.2 The Metadata Parser	10
2.2.1 Installing the Metadata Parser.....	10
2.2.2 Using the Metadata Parser.....	11
2.3 Application Development	17
2.3.1 Wapache vs. Alternatives.....	17
2.3.2 Changes to Wapache.....	21
2.3.3 Software Development Tools.....	22
2.3.4 Third Party Code and Resources.....	25
2.4 Analysis of the Metadata Parser	27
2.4.1 Source Code Structure.....	27
2.4.2 Inner-Functioning of the Metadata Parser...	30
2.4.3 Known Bugs.....	39
2.5 Discussion	41
2.5.1 Code Portability.....	42
2.5.2 Development of Plug-Ins.....	43
2.5.3 Future Development.....	44
Section 3: <i>Conclusion</i>	45
3.1 Conclusion	46
3.2 Notes	46
3.3 Reference	47
Appendix I: Pictographs	48
Appendix II: Code	54

SECTION 1
Introduction

1.1: *Abstract*

The dependence on text to present geospatial metadata files may hinder an individual's ability to quickly interpret the information. Presenting the same information as a series of simple symbolic pictures simplifies the absorption of information, which may be useful when evaluating a number of metadata files.

The Metadata Parser is computer software developed for the purposes of this research project. It is designed to automate the process of converting text metadata files into a visually rich format. It was conceptualized by Dr. Sawada, and was developed by myself over the period extending from October 2007 until February 2008. The application is Windows compatible, and at this stage reads only FGDC formatted XML metadata files. An integrated plug-in system, however, means that the program could also be expanded to incorporate more metadata standards in the future.

1.2: *Writing Conventions*

To reduce potential confusion during the reading of this paper, a few writing conventions have been adopted. They are explained in the table below.

Writing Conventions Table	
Hello World!	Default text formatting.
<code>Wapache.exe, parser</code>	Computer files and directories.
<code>foobar ()</code>	Computer code.

1.3: Introduction

“Metadata” is a term used to denote documentation that describes the characteristics of a data set (Michener et al. 1997). Within a geospatial environment, this extends to notions of location, resolution, acquisition methods, agencies responsible for maintaining the content, and so forth.

However, the dependence on text to convey that information can also make geospatial metadata unwieldy for quick interpretation. For instance, in order to assess the type of data at hand, a geomatics specialist may have to sift through line upon line of obscure metadata content to find what he/she needs.

An alternative to this traditional means of presenting metadata has been propositioned by Dr. Michael Sawada, director of the Laboratory for Applied Geomatics and GIS Science (LAGGISS) at the University of Ottawa. The idea consists of presenting the otherwise text metadata content as a series of simplified illustrations. To that extent, the research question for this undergraduate research paper becomes whether such an option is viable on a practical level.

The answer to this question is presented in the form of the *Metadata Parser*, geospatial metadata visualization software. The *Metadata Parser* is a computer program developed for the purposes of this research, which provides a functional demonstration for the conversion of text-based metadata files into a visually rich, picture-based, format.

SECTION 2

Metadata Parser

2.1: Metadata Parser Introduction

The *Metadata Parser* is the name of the computer program developed for the purposes of this undergraduate research project. It serves as a means to exemplify a tangible avenue through which metadata can be visualized.

The program's essential mode of operation is to interpret text geospatial metadata files and produce a visually rich document containing a series of pictographs. For instance, if a user selects a metadata file that describes a vector data file of the road network in Alberta, then the program will generate a new document containing symbolic images representing the notion of roads and the province of Alberta.

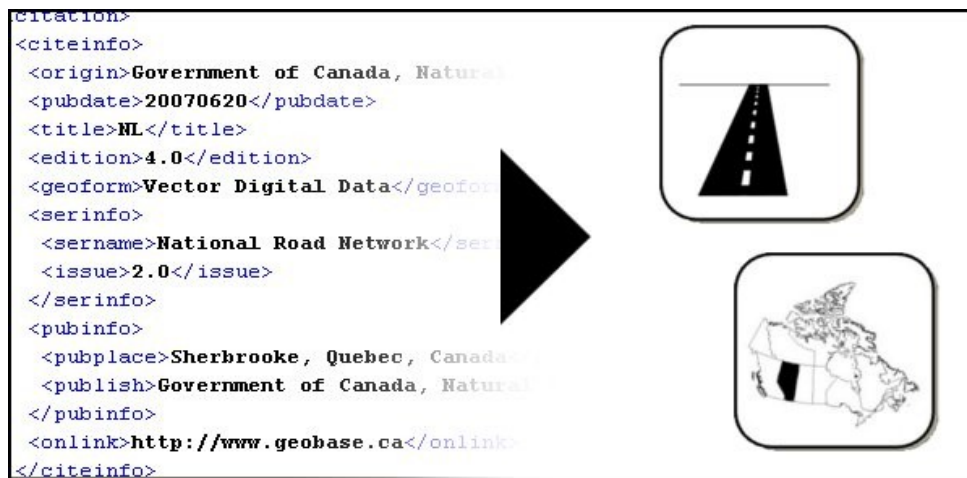


Illustration 1: The conversion from text metadata into a visual format by the Metadata Parser. Raw metadata content on the left, and the interpreted iconified representations of the content on the right.

The *Metadata Parser* is based on the concept of visualizing metadata, proposed by Dr. Sawada of the University of Ottawa. The program was developed over a period covering from October 2007 until February 2008, and the final revision, which includes an experimental plug-in, was released on March 1st, 2008.

2.2: The Metadata Parser

Much in the same way that the *Metadata Parser* was designed to simplify the interpretation of metadata files through their visualization, the general design philosophy was also to create an application that would be simple to use. As such, text in the application interface is large and kept at a minimum, a two pane comic is used to introduce the concept behind the purpose of the application, and the whole program is presented as an approachable wizard.

2.2.1 Installing the Metadata Parser

Installing the Metadata Parser is straight-forward. Once the user obtains the installation executable, which is also available in the DVD included with this paper, they simply need to start it to begin the installation wizard.

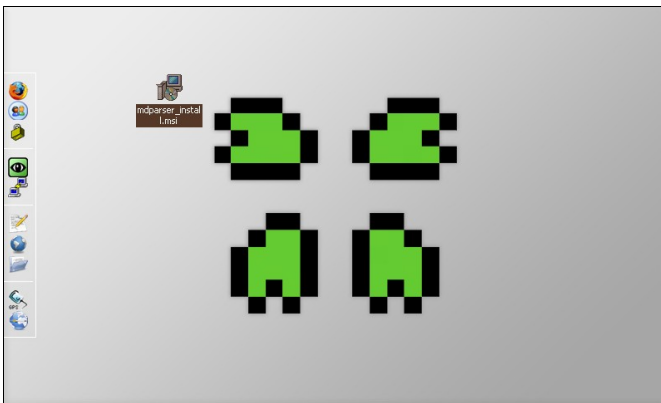


Illustration 2: Starting the Metadata Parser installer.

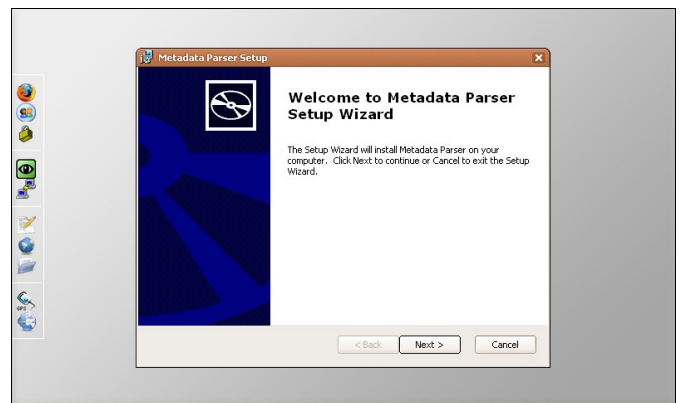


Illustration 3: Installation wizard guiding the user through the installation.

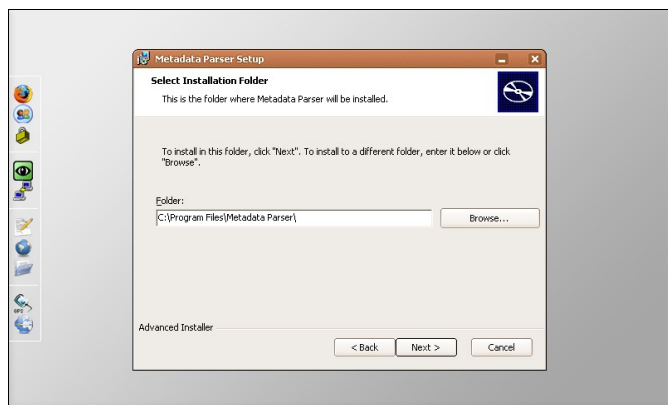


Illustration 4: Continuing the installation wizard.

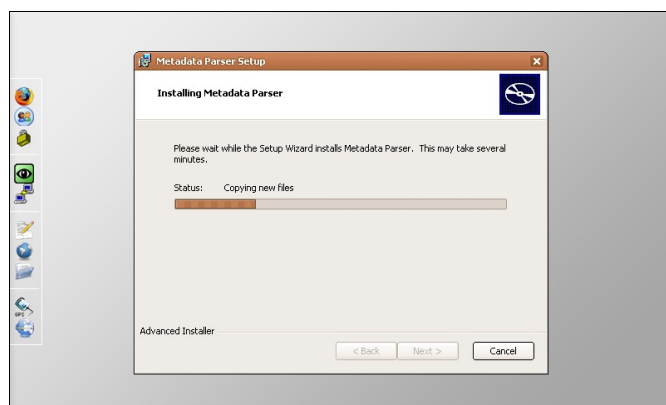


Illustration 5: The installer copying the files onto the computer.

After these steps have been completed, the *Metadata Parser* will be installed, and there will be icons present on both the desktop and the “Start” menu.

2.2.2 Using the Metadata Parser

The *Metadata Parser* is made up of two parts: the “wizard”, which guides users to input the proper information for processing, and the “report”, which displays the visualized metadata.

Starting the *Metadata Parser* is as easy as launching it from the desktop or the Start Menu. In this particular computer for the demonstration, the icon is stored in an application launcher on the left hand side of the screen.

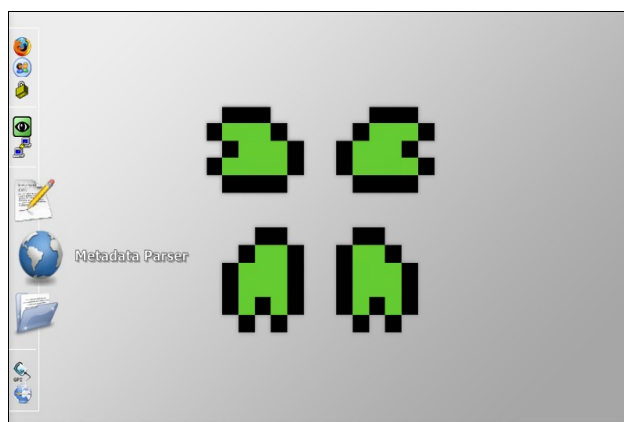


Illustration 6: Starting the Metadata Parser from the application launcher.

When the *Metadata Parser* is started, it will greet the user with an introductory two pane illustration visually explaining the purpose of the application. In the bottom of the window are also buttons that provide the user with either the option to see the credits for the program, or to begin the wizard.



Illustration 7: Introductory window for the Metadata Parser, including a two pane illustration visually describing the purpose of the application.

If the user chooses to begin the wizard, he/she is treated to the first step of the wizard. In this step, the user is asked to select a metadata file.



Illustration 8: The first step in the Metadata Wizard, which asks the user to select a metadata file.

Failure to select a metadata file, or to select an XML formatted file, will result in an error. Continuing with the philosophy of ease-of-use, the user is informed of the error in plain English.

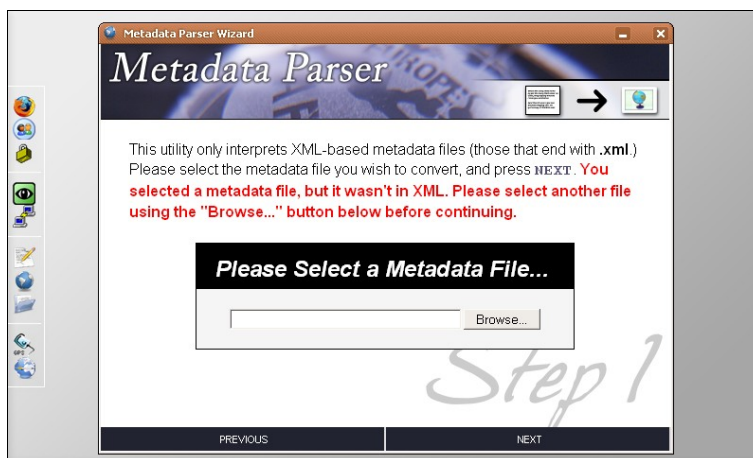


Illustration 9: The Metadata Parser producing an error to the user, who selected a file that wasn't formatted in XML.

With the successful selection of a metadata file, the user will be passed onto the metadata type selection. This is where the user tells the wizard what standard the metadata file uses, so that it can be properly interpreted by the program. Though the current release of the *Metadata Parser* only supports the **Federal Geographic Data Committee (FGDC)** metadata standard, an elaborate plug-in system allows developers to add support for additional metadata types.

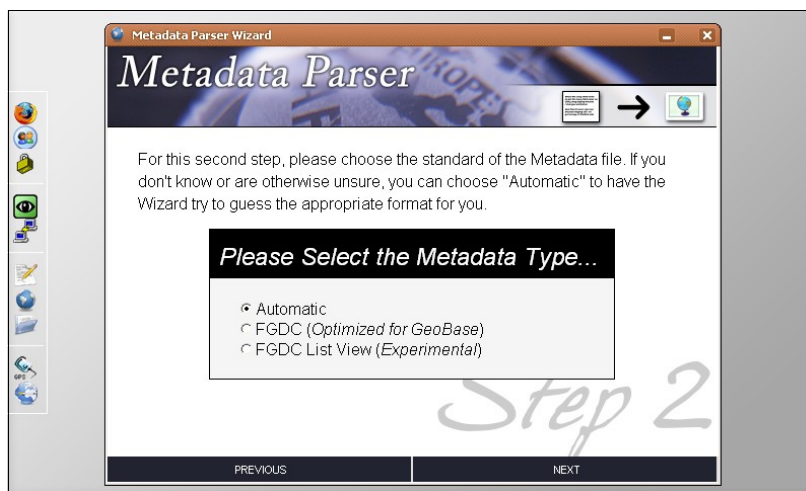


Illustration 10: Choosing the metadata standard. In the current experimental Metadata Parser release, there is a second FGDC parser which is absent in the standard edition.

The “Automatic” function seen in the illustration above is not yet fully implemented. Initially designed to analyze a metadata file to determine its type, its development was axed due to time constrictions. As it stands, using “Automatic” mode will default the user to the regular FGDC parser. Once the user selects the appropriate metadata type, they can then proceed to the final step in the wizard: confirmation. It is at this stage that the user verifies that the right file, and parser, were selected. Once everything checks out, the user can complete the wizard by pressing the “Finish” button.

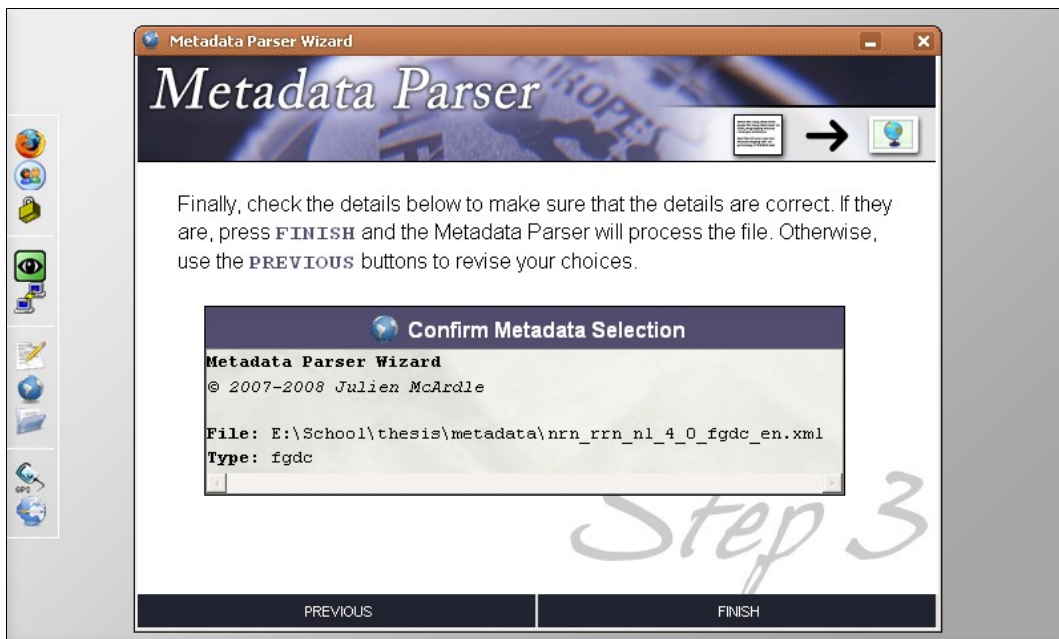


Illustration 11: The last step of the wizard for the Metadata Parser: the confirmation window .

Upon pressing the “Finish” button in the wizard, a new window will launch containing the visualized metadata. The contents of the window is dependent upon the metadata, and the selected type. Selecting the regular FGDC parser, for instance, will yield a window akin to the following illustrations.

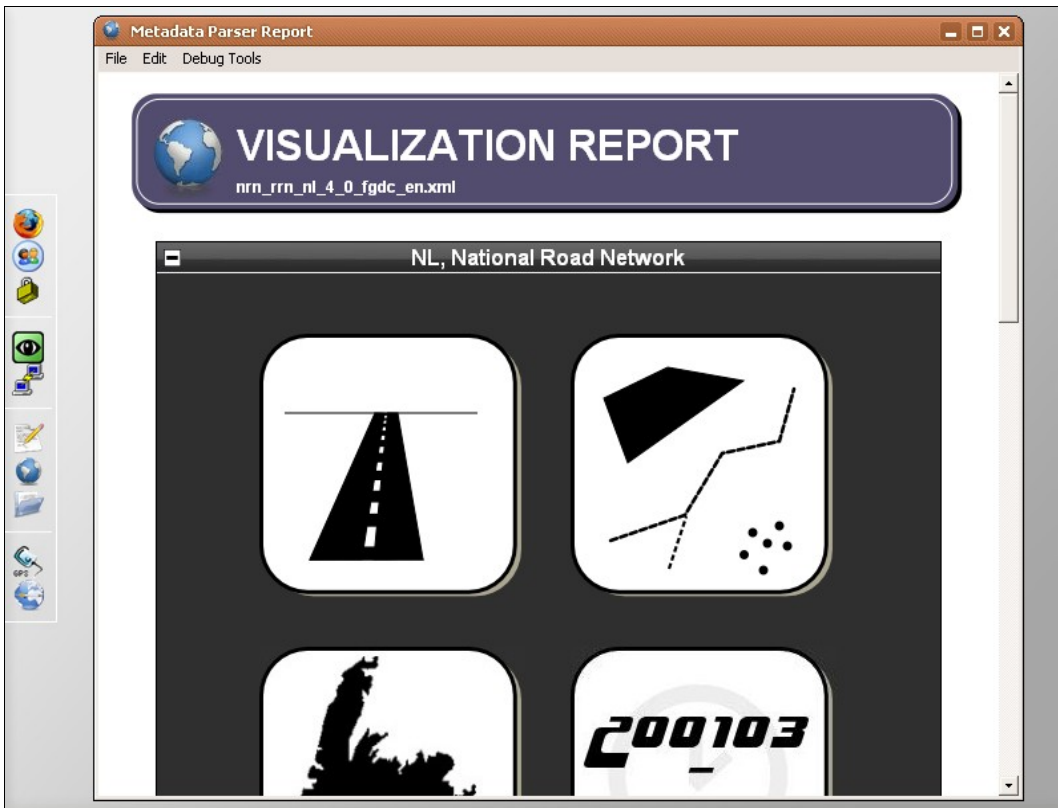


Illustration 12: The Metadata Parser's "report" window featuring the visualized metadata.

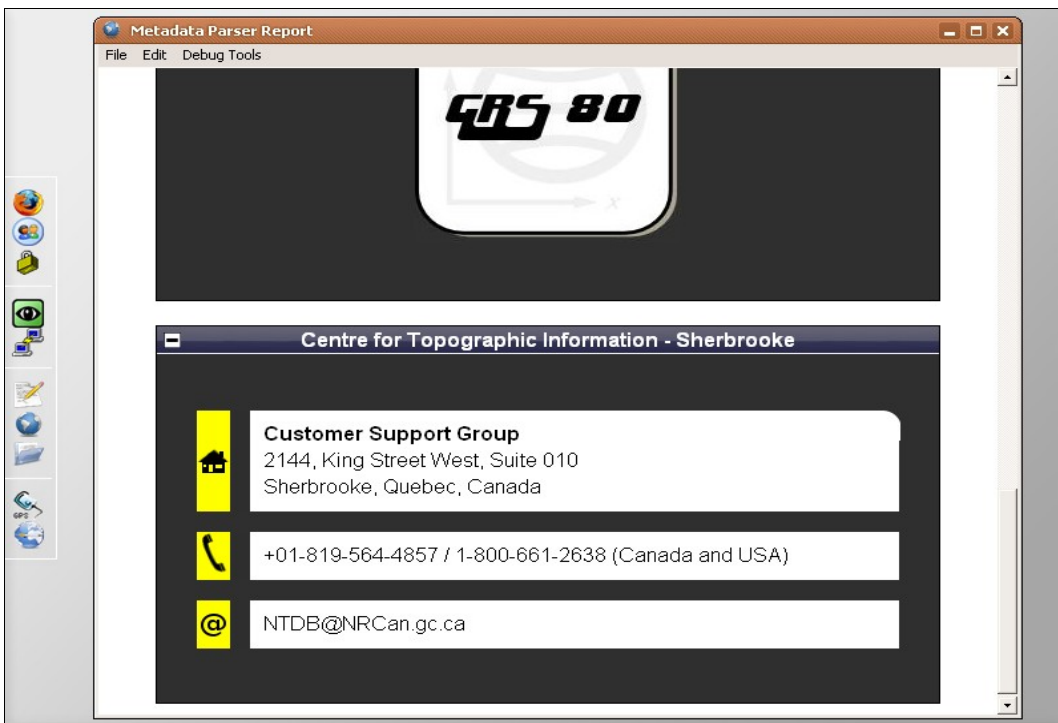


Illustration 13: The "report" window continued.

However, if the user selects the experimental parser, than they will see something more akin to the following image.

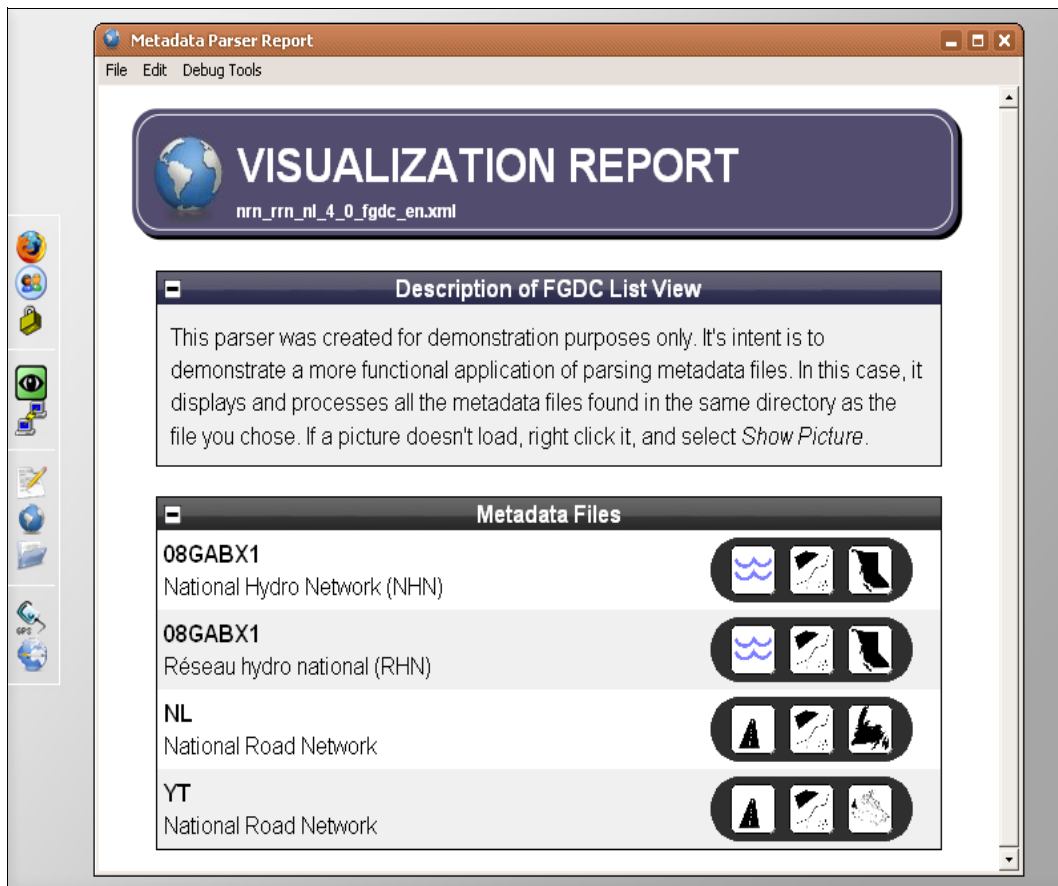


Illustration 14: Report window featuring the experimental FGDC parser. The experimental parser analyzes all metadata files that are found within the same directory as the file that was chosen in the wizard.

From this window, the user is also able to print the document, export it as an HTML file, or utilize debugging functions useful to the development of further parsers.

2.3: Application Development

The *Metadata Parser* was developed for the *Wapache* application framework, a development solution which allows programmers to code full-fledged applications using the same programming languages that are prevalent in web development¹. The end result is a flexible, portable, yet powerful, platform with which to create software.

2.3.1 Wapache vs. Alternatives

The power of *Wapache* as a development platform comes from the power afforded to it by the web development languages it supports. These languages are HTML, CSS, JavaScript, and PHP.

HTML stands for *HyperText Markup Language*, and serves as the backbone of the common web page. It contains the data to be presented, and handles the general formatting of things. CSS, or *Cascading Style Sheets*, is a newer language which has for its purpose the handling of the presentation of elements within HTML or other markup languages. JavaScript is a client-side scripting language, which despite its limitations brings a level of interaction with the web content that is absent from HTML and CSS. PHP (*PHP: Hypertext Preprocessor*) on the other hand is a robust server-side scripting language. It is able to read and open files, perform complex algorithms – in essence afford the developer the same kind of programming flexibility found in traditional software development. None of these aspects change within *Wapache* – HTML still handles the data and formatting, CSS the looks, Javascript the dynamic interactivity with the content, and PHP the processing.

What's important to note here is that all four of these languages are scripting languages. This is unlike traditional software programming languages, such as C and C++, which compile into executable binaries. Rather, the code here is parsed on-the-fly by *Wapache*. This allows the code to be modified and tested instantly, without the need for potentially lengthy compiling which is a necessity with other traditional software development languages.

To better understand the other benefits of *Wapache*, it is important to understand exactly just what *Wapache* is. The Windows® program is an integrated and portable combination of a web browser (*Microsoft Internet Explorer*), a web server (*Apache*), and a virtual machine (*Zend Engine*.) Each application running on *Wapache* includes these three elements.



Illustration 15: A typical Wapache window. The code that determines the appearance of the application are stored in different locations. The code for the general design of the window is held in the default.wcf file, while the contents are held in the root source code folder.

Much like a traditional web server-client relationship, the server acts to retrieve the source code and send it to the client. The client, in this case the embedded *Internet Explorer* browser, then parses and displays the code as the appropriate visual elements. If the need arises, code can also be transferred to the *Zend Engine* virtual machine for processing².

The manifestation of the source code as rendered by *Internet Explorer* is displayed in the Wapache application window (the blue area in the figure above.) The general window design (red area) is handled by a configuration file called `default.wcf`. It is in that file that characteristics such as the window size, the presence of scroll bars, menus, and so forth are defined. This last file does not use web development languages for its content, but rather follows a proprietary scripting format unique to *Wapache*, with tie-ins to *Internet Explorer's* **Dynamic HTML** (DHTML) application programming interface.

This means that the **graphical user interface** (GUI), with the exception of the overall window design, is actually implemented through the use of web development languages. It is this fact which underlies one of *Wapache's* greatest strengths. Creating new elements in the application GUI can be as simple as writing a short string of HTML. Making that element interactive is as simple as adding a bit more HTML or Javascript.

HTML and CSS are presentation-driven scripting languages geared towards web content. By supporting these, Wapache brings their ease of use to the program developer. To simplify matters even more, the adoption of these languages means that a developer could use, if so desired, WYSIWYG (**What You See Is What You Get**) editors tailored for HTML and CSS. Popular examples of such editors include the open-source *NVU* or Adobe's (formerly Macromedia's) *Dreamweaver*. Such

applications can greatly decrease user interface development time.

PHP, on the other hand, provides solid processing capabilities to a Wapache application. It is a high level programming language, which is more approachable than a lower-level programming language common in software development circles. Unlike lower-level programming languages, developers in PHP do not have to deal with time-consuming resource management, nor do they have to deal with an abstract machine-code oriented programming style. Instead, they can focus directly on software development that's more akin to pseudocode than to C or C++.

Furthermore, a number of software libraries are included with PHP that save even more time for the developer. Of particular interest for the purposes of this project are the GD Library, which allows for image creation and manipulation, as well as Expat Library, which allows for XML (eXtensible Markup Language) parsing. These files allow for complex operations to be accomplished in PHP without the need for complex code. Instead, the developer can simply call these functions, which then handle the relatively simple inputs submitted by the developer. The presence of these libraries provide a developer with approachable deep functionality while simultaneously cutting on development time.

Another perk that arises from the usage of these languages is the debugging process. Syntax in HTML is fairly loose, and programming mistakes may have minimal repercussions on the code's overall functionality, if any at all. Likewise with CSS. JavaScript is less forgiving, but the built-in error checking that's included in *Wapache's* embedded web browser identifies exactly where in the code faults are occurring, eliminating the possibility of a wild goose chase as the programmer attempts to find the error. The *Zend Engine* is likewise kind to developers, providing them with helpful error messages that greatly cut down on debugging time.

However, there are also a number of significant disadvantages that hinder *Wapache's* suitability as a replacement for traditional software development platforms. First of all, *Wapache's* high level language approach means that it is not as efficient in using the computer processor as a lower-language code that more directly in tune with machine architecture. Furthermore, despite the great boons afforded to development through the use of web development languages, there are some inherent limitations by this adoption. The content is not designed for continuous updates, which makes tasks such as incorporating video playback or basic 3D modeling exponentially more difficult than in any other traditional programming environment.

All-in-all, *Wapache* provides developers with an application development environment that's powerful, yet simple to use, easy to debug, and fast to code for. These are the reasons for which it was adopted for this research project. It is thus perhaps surprising to hear that *Wapache* is not terribly well known. According to the statistics provided by its web host, *SourceForge*, only eleven copies of the software were obtained during the month of its adoption for this project³.

2.3.2 Changes to Wapache

While the generic executable included with *Wapache* was suitable for this project's needs, a number of aesthetic changes were implemented to customize it to this project. The tool of choice to carry out these changes is called *ResHacker*. It was developed by Angus Johnson, an Australian programmer, and was last released in 2002. This program decompiles executables, and allows certain application elements to be altered⁴.

To that extent, *ResHacker* was used to modify the icon for the generic *Wapache* executable. The default feather icon was replaced with a more suitable Globe icon, which was obtained from a free icon library created by Bogdan Condurache.

The executable was also renamed from `Wapache.exe` to `mdparse.exe` (“Metadata Parser”.) The basic directory structure of *Wapache* also underwent a minor change: the name of the `htdocs` folder was changed to `source`, to better reflect the content held within.



Illustration 17: Old and new Wapache icons compared.

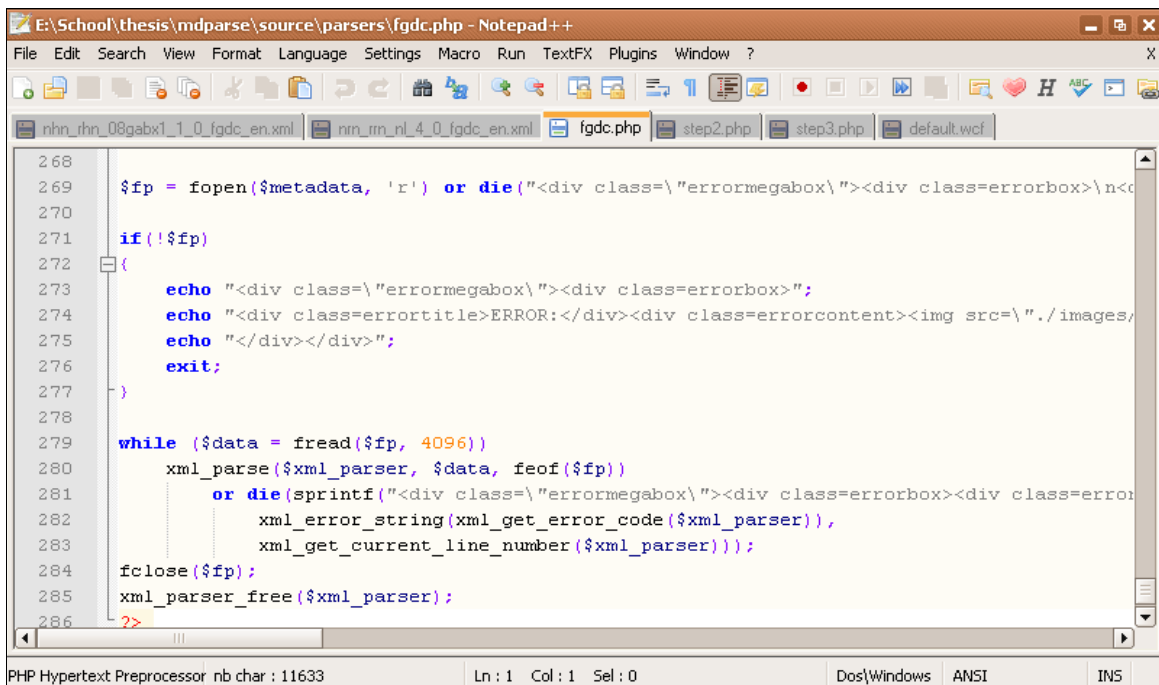
The Apache configuration file (part of `default.wcf`) was also edited to treat text files with the `.djpg` extension as PHP code to be processed. This is part of the **D**ynamic **J**PEG file format that was created for the purposes of this project, the details of which will be examined later in the paper.

2.3.3 Software Development Tools

*W*apache servers as a platform with which to *run* code, but it does not provide a means with which to actually *write* the code. For that, the developer needs to find a text editor. For basic code, the default text editor that comes bundled with Microsoft's Windows[®] operating system, *Notepad*, is sufficient.

However, *Notepad* does have a number of significant limitations. It is unable to open more than a single file at a time, it does not number the lines of text, its *undo* function is limited to a single undo/redo, and it does not provide syntax highlighting. The inability to open multiple simultaneous

files means that the developer is either forced to constantly close/open new files, or have multiple instances of *Notepad* open at the same time. A lack of line numbering makes it difficult to identify the a line containing erroneous code as identified in the debugging process, especially in files involving hundreds of lines of code. The poor implementation of the undo function means that multiple changes to the code cannot be reverted. Finally, a lack of syntax highlighting, which differentiates different constructs in code, reduces the general readability of the code.



```
268
269 $fp = fopen($metadata, 'r') or die("<div class=\"errormegabox\"><div class=errorbox>\n<
270
271 if(!$fp)
272 {
273     echo "<div class=\"errormegabox\"><div class=errorbox>";
274     echo "<div class=errortitle>ERROR:</div><div class=errorcontent><img src=\"./images/
275     echo "</div></div>";
276     exit;
277 }
278
279 while ($data = fread($fp, 4096))
280     xml_parse($xml_parser, $data, feof($fp))
281     or die(sprintf("<div class=\"errormegabox\"><div class=errorbox><div class=error
282         xml_error_string(xml_get_error_code($xml_parser)),
283         xml_get_current_line_number($xml_parser));
284 fclose($fp);
285 xml_parser_free($xml_parser);
286
287 >>
```

Illustration 18: Notepad++ in action.

For that reason, an alternative text editor was adopted for the development of the *Metadata Parser*. It is an open-source tool developed by Don Ho that is called *Notepad++*, and that is freely distributed under the General Public License. Unlike Notepad, it supports multiple document tabbing, multiple undos, syntax highlighting, and other features that simplify development.

To create the graphics used in this project, an additional set of programs were used: *Inkscape*, *Gimp*, and Adobe's *Photoshop*TM. The first two programs can be downloaded for free, and as they are open-source and distributed under the General Public License, software developers are free to modify the code and improve the program.

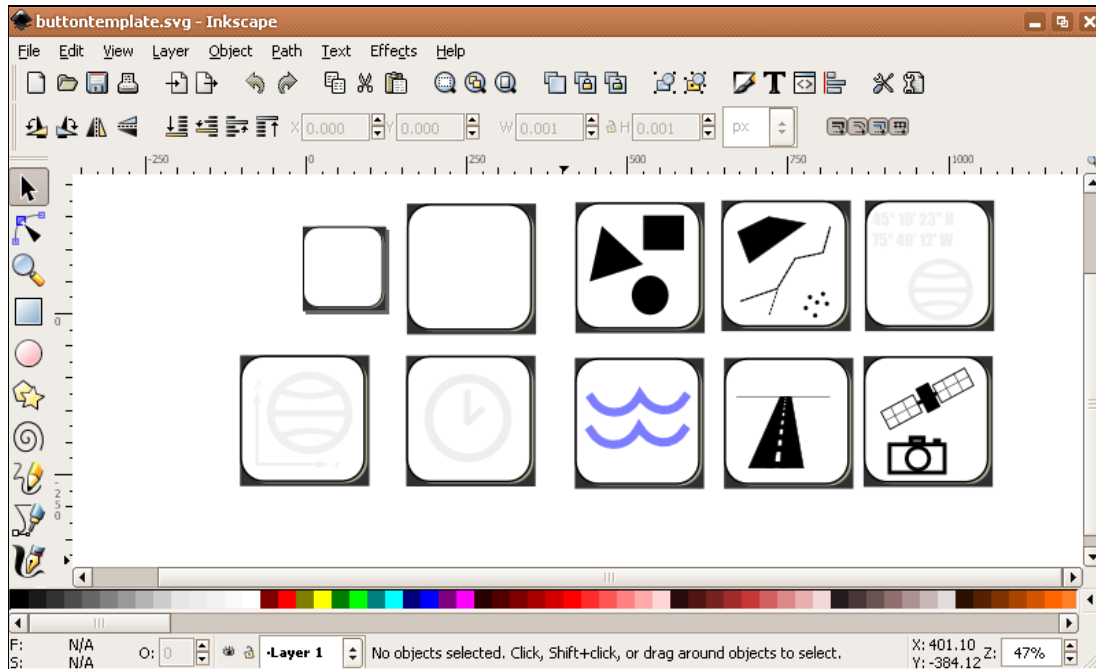


Illustration 19: Inkscape in action during the development of the Metadata Parser.

Inkscape is a tool to create and edit vector images. In the development of the *Metadata Parser*, it was used to create most pictographs (see illustration above), as well as a number of other incidental images used in the program. *Gimp* is a raster graphic editing program, functionally similar to Adobe's *Photoshop*TM. It was used to create more pictographs, icons, as well as backgrounds. *Photoshop*TM was to be the raster graphics editor, but its use was quickly supplanted by the free *Gimp* in the early stages of development. The push for the switch was due to licensing issues with Adobe's product.

2.3.4 Third Party Code and Resources

The current version of the *Metadata Parser* is made up of 1,855 lines of code. To facilitate development, and to fill the knowledge gaps with PHP, some of this code was borrowed from other sources. The borrowed code is listed in the table below, in order of descending importance:

External Code Assets		
Item	Author	Overview
XML Parsing Code	Kevin Yank	While the current XML parsing code in the <i>Metadata Parser</i> is unrecognizable from Kevin Yank's implementation, his code was the figurative Rosetta Stone in understanding how PHP handles XML. The parsers included in the program are based on his work.
Hide/Show Script	Will Bontrager	A hybrid of JavaScript and CSS, this code allows segments of the visualized materials to be shown or hidden at the behest of the user. This code was condensed for the purposes of this project.
Random Character Generator	Bill Pellowe	This code is integral to the functioning of the Hide/Show Script.
File Extension Identification	Angela Bradley	This code is used in different parts of the program, including for error detection, as well as identifying plug-in files.

Images from the royalty free stock photography website, *stock.xchnng* (www.sxc.hu) were also used in the development of the *Metadata Parser*. Modified versions of these images served as the backgrounds for the credits window in the program, as well as the title graphic in the wizard.

Other third-party resources are delineated in the table below:

External General Assets		
Item	Author	Overview
Fonts	Ray Larabie & Hans Zinken	The fonts developed by these two individuals are integrated into the static and dynamically generated images of the <i>Metadata Parser</i> .
Application Icon	Bogdan Condurache	The main program icon for the <i>Metadata Parser</i> is a globe graphic designed by Bogdan Condurache.
Sounds	AT&T Labs & Partners in Rhyme	The synthesized voice provided by AT&T Labs' Text-to-Speech engine serves as the vocal declaration upon the successful completion of the <i>Metadata Parser's</i> wizard. Sound effects by Partners in Rhyme are used to alert users to program errors.

2.4: Analysis of the Metadata Parser

The *Metadata Parser* can be thought of as being made up of two components: the Wapache application framework, and the source code. The Wapache framework is the generic environment which allows the source code to run and be turned into Windows[®]-compatible programs. Without Wapache, the source code is just a collection of innate text and picture files. Without the source code, Wapache has nothing to process and therefore nothing with which to produce programs.

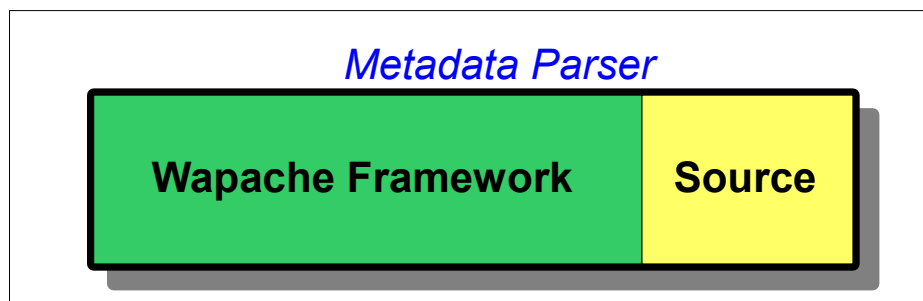


Illustration 20: An abstracted representation of the Metadata Parser, application and it's two major sub-components: the Wapache framework, and the source code.

The term “source code” is a bit of a misnomer, because source code does not necessarily entirely consist of code. While the executable portion of the program is indeed generated from the code, it may also rely on additional resources such as images and sounds for its creation.

While the previous section focused more on *Wapache*, this section will focus on the source code which truly makes the *Metadata Parser* what it is.

2.4.1 Source Code Structure

The source code in the *Metadata Parser* can be broken into three components: *pages*, which are the fundamental building blocks that produce the content seen in the windows and call

additional resources, *called code*, which are code segments called on-demand by the *pages*, and *audiovisual resources*, such as pictures and sound effects.

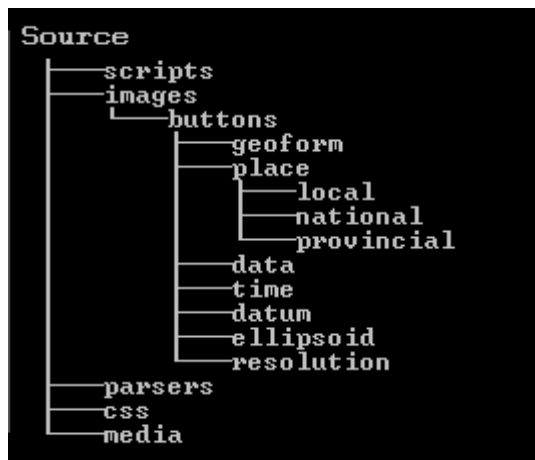


Illustration 21: Directory Layout of the Metadata Parser's source code, as generated by Microsoft's tree program.

With respect to the directory structure of the source code, the pages reside in the root directory of the source code. The called code are stored in separate directories, depending on their purpose. These directories are **css**, **parsers**, and **scripts**. The audiovisual resources are stored into two directories: **images** for the pictures, and **media** for the sound effects.

When a user is looking at the program credits, or one of the steps in the wizard, or any other window in the *Metadata Parser*, they are looking at the product from the Wapache framework parsing of one of the source code's eight rudimentary pages. These pages are the fundamental building blocks of the program: they set up the basic layout of the window, .include the code that processes events and files, and calls in images and further code if necessary. The page are typically made up of a mix of HTML, PHP, and JavaScript.

The pages are also independent. They do not interact with each other at all, though they can link (send the user) to one another. This complicates the matter of sending information from one page to another, the solution to which will be discussed later on.

When the Metadata Parser begins, the first page the user will see will always be the *index* page. The term is derived from the web development world, in which the “index” page is the default web page to be called in a directory. The index page in the Metadata Parser is that introductory window seen in *Illustration 7*.

It is important to note that the user is only able to navigate between these pages by the links provided on those pages. This constricts the user to follow a certain program flow, as delineated in the following illustration.

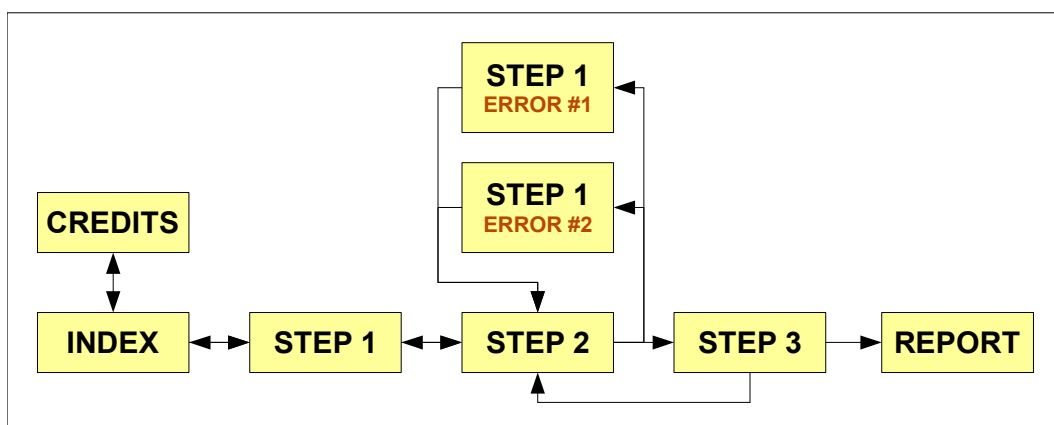


Illustration 22: Links between the eight pages that make up the Metadata Parser.

As an example, a user who is on the page “Step 1”, that is to say the page that represents the first step of the wizard, can only go on to “Step 2” or backtrack to the index page.

External to these pages are the called code. These external files contain additional segments of code which can be integrated into the pages on demand. There are many advantages to externalizing the code. For one, this reduces the complexity of the pages by having lines of code outsourced to other files. Furthermore, repeating portions of code can be represented a single time through the use of one of these external files, further cutting down the size of the pages. Other practical applications exist as

well. For instance, take the report page, which is responsible for visualizing the metadata. This page calls the code for the right parser from a matching external file, and integrates that code into itself to then process the metadata file. This means that a single report page doesn't have to contain all the parsers in its own code for it to work, it can just get the one it needs from one of those files.

2.4.2 Inner-Functioning of the *Metadata Parser*

Analyzing all the 1,855 lines of code verbatim that make up the source code for the *Metadata Parser* would be both unwieldy and a confusing means through which to explore the operation of the application. Instead, this report will speak generally of the source code and the inner-functionality of the program. The full source code can be found in the appendix of this report.

2.4.2.1 The Metadata Parser's Wizard and the Report Window

The *Metadata Parser's* wizard is made up of an index page, a credits page, and three steps. The index page serves as the introduction to the wizard, providing the user with basic information as to the purpose of the application. The index page contains links to the credits, and the first step of the wizard (see *Illustration 22.*)

The credits page serves to credit the work of those on which this project was reliant. These listed names are then all scrolled upwards, similarly to a end-credits scene in a movie. This list of credits, however, is not static. While fourteen basic names will always appear, more names can be dynamically loaded from the list of parsers. This is part of the plug-in framework employed by the application, which aims to create a platform whereby users can “drag 'n drop” new parser files into the application, without requiring any additional programming in order to integrate them into the program.

The plug-in framework isn't a consolidated set of code, but rather bits of code here and there in the overall application that combine to create the “drag 'n drop” functionality. In the case of the credit page, a segment of code will check the `parser` directory for parsers. If it finds any, it will check every parser and obtain it's author, and will then use that name in conjunction with the parser's filename to generate the appropriate credits.



Illustration 23: Credits window. The "Sounds By" credit is static, the credits for the two parsers are dynamically loaded into the application.

Moving on from the credits are the three steps of the wizard. The first step is tasked with having the user identifying the metadata file to be visualized. The second step is tasked with identifying the metadata type, and thus the parser which will process it. Finally, the third step serves to confirm the details.

In terms of its code, the page representing the first step is much like the index page – relatively short and straightforward. To store the information given to the application by the user, the program uses the

POST method to pass the data onto the page of the second step, which then stores the information for later retrieval by using PHP sessions. The POST method is the traditional means by which to submit data to websites, and is applicable to the *Wapache* framework as a means to submit information into itself. Sessions are a means to store data on a per-user basis⁵. This circuitous approach of using both the POST and session techniques are necessary in order to circumvent the limitations placed by using the HTML construct that serves to browse and select our metadata file for visualization.

The second step takes the metadata file's location submitted to it and stores it. However, it will also analyze the submitted file to make sure that it is valid. This comes in the form of two checks. The first is to determine whether the user submitted any file at all for analysis. If the user did not, the program will evaluate whether that's because a metadata file was actually already selected beforehand, a possibility if the user was backtracking to the second step from the third. The second check is to determine whether the file in question is an XML file. It does so purely by reading the extension of the file, those three letters following the period at the end of the file name. If those checks fail, the user is automatically forwarded to an error page, the content of which is reflective of the nature of the problem. These error pages are identical to the first step in every respect, with the exception that they contain additional text detailing to the user the nature of the error.

The second step is also tasked with having the user selecting the parser to process the metadata file. The parser listings are entirely dynamic. The program will look in the **parser** directory for parsers, and list them. The name for the parsers that is displayed on this page is pulled from the files themselves, at specified locations within those files which is standardized across all parsers designed for the *Metadata Parser*. Furthermore, the very display of the page is dynamic. The presence of scroll bar will automatically be added to the window that lists the parsers if the program detects that more than three

parsers are to be present.

The third step takes the new information from the second step and stores it. It will then retrieve the stored information of the metadata file's location, as well as the parser selected, in order to present it to the user for the purposes of confirming their choice. A link is present to launch the report window.

The report window presents the user with the visualized metadata file. In order to visualize the metadata, however, it first loads in the metadata file's location and the parser that the user chose from the stored data in the PHP session as variables. It will use the parser variable to load in the code for the appropriate parser into the report page, which will then use the variable for the metadata file's location to actually process through the metadata file. It will also use the metadata file's location to extract the metadata filename, which is then displayed beneath the main header of the report window. This is exemplified in *Illustration 24* where the filename is `nrn_rrn_nl_4_0_fgdc_en.xml`.



Illustration 24: The Metadata Parser's report window.

The report window also loads the code for a random generator and a script to hide and show content on-demand, which may be of use to parsers. There is also a script that checks the overall size of the window, for unlike the window of the *Metadata Parser's* wizard, the report window is resizeable. Therefore, it must ensure that the dimensions of the window aren't too small, or risk distorting the contents of the window. As such, the program makes sure that the window is a minimum of 530 pixels wide. Failure to meet this requirement will produce an error, which will be corrected upon the widening of the window (see *Illustration 25.*)

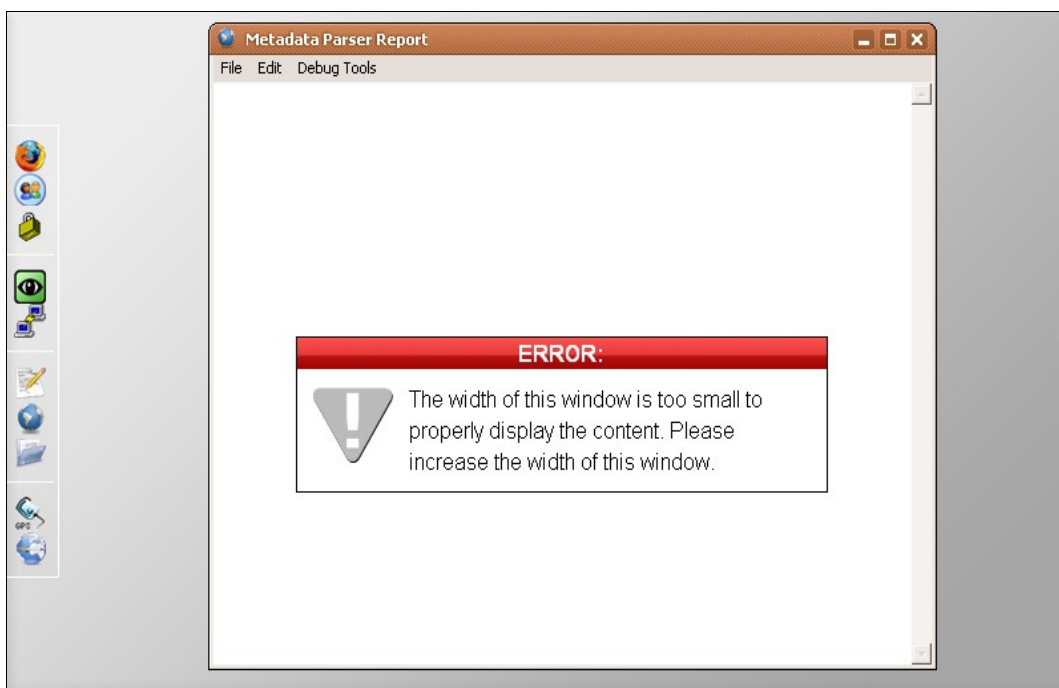


Illustration 25: Error obtained when the window is resized to a width that is considered too small for the proper viewing of the report's content. Resizing the window to a larger "safe" resolution will automatically remove the error and return the regular contents of the report.

2.4.2.2 Metadata Visualization and the FGDC Parsers

The are two FGDC parsers in the experimental release of the *Metadata Parser*, but they both interpret and visualize the metadata in identical ways. The only thing that changes between the two is the fact that one parser parses a single file at a time (see *Illustration 12*), while another will

process and display multiple files simultaneously (see *Illustration 14*.) The presentation is altered, but the underlying mechanics remain the same.

The FGDC parsers have been geared towards the metadata content published by *GeoBase*, a distributor of free geomatics datasets located at the Earth Science Sector of Natural Resources Canada⁶. It should be noted that other producers of metadata might follow very different approaches to the content, while still abiding by the FGDC standard. As this author's experience shows, even with the material produced solely by *GeoBase* there's a certain level of heterogeneity with respect to the formatting and treatment of content. As such, this parser was tailored to the files produced by *GeoBase* to increase this applications effectiveness with regards to those files, and therefore enhance its ability to serve as a successful technical demonstration.

Looking first at the FGDC parser that analyzes single metadata files, it breaks down the file into three logical sections. The first section contains general information about the file: the title for the dataset, the series to which it belongs, the nature of the data (ie. road network), the type of data (ie. vector, raster, or tabular), the location of the dataset, the range of dates in which the data was collected for the set. The second section contains more technical details: the resolution of the dataset, the datum and ellipsoid used. The third section contains contact information for the office that published the data. The other FGDC parser that analyses multiple files skip on the second and third sections.

The title and series of the dataset are not initially visualized, but rather presented in the title bar of the first section. It should be noted that all three sections have a title bar, all of which feature a “collapse/expand” button to hide or show that section at the behest of the user. To visualize the nature

of the data, the program calls a critical function called `findbutton`. This function takes two input variables: the location of a library of images, and a string of text. It will then take each filename of the pictures found in that directory of images, minus their extension, and try to match it to one of the words in the string of text. If it finds a match, it will return the picture that represented that match. So for visualizing the nature of data, the library of images in question is a repository of pictures representing different types of data, such as road networks, river systems, satellite imagery, digital elevation models, and so forth. The string to which they will be matched will be the title of the data set, and failing that, the series. So for instance, if the series is entitled “The Road Networks in Alberta”, the program will find that the picture “**road.png**” matches a word in the series name, and will therefore display that image as the selected pictograph by the *Metadata Parser*. Adding new pictures to the library is as simple as dragging and dropping new pictures into the appropriate directories. The program will automatically include them upon its next search.

The visualization for the *type* of data, that is to say whether the data is in a vector, raster, or tabular format, follows the same formula as the previous visualization process. The string of text in the metadata file for the type of data is extracted by the XML processing functions of the parser. This string is then matched to a series of images in a pre-defined repository of images. Successful matches are presented to the user. Any external format not provisioned for are represented by an additional image indicating the lack of success.

Visualizing the place for the dataset also follows a similar process, only the search through the image repositories are done on a hierarchical basis. Three levels exist in the hierarchy: local, provincial/state, and national/continental. On a local basis are cities such as Ottawa (represented by the city outline) and Toronto (visualized as the CN Tower). On a provincial/state basis are all the Canadian provinces and

territories, plus a few recognizable American states. All these entities are represented by the outline of their territory. On a national/continental level are a number of flags and outlines representing their nation and/or continent. The *Metadata Parser* will first try to find a match for the string representing the location of the dataset on the local level. If it fails to find a match, it will try to find one on a provincial/state level, before moving on to the greater national/continental level. So for instance, a dataset located in the community of Red Deer, Alberta, would have a pictograph with the outline of Alberta representing it. A second dataset located in Sydney, Australia, would be represented with an image of the outline of Australia.

Visualizing the timescale for the source data is done in a different manner. For that, the *Metadata Parser* directly extracts the dates in question from the metadata file, and superimposes them onto a base image. The combination is then exported as a new picture: the pictograph. To ensure legibility, the size of the text is auto adjusting – it will try to be as large as possible, without exceeding the width of the image. To do this, the program first superimposes text at the full font size of 36 points. It will then measure the width of the resulting text. If it's too wide, it will reduce the font size by one unit, and remeasure the dimensions. It will keep doing this until an appropriate width is found. Also to note is that the code to do all of this is located in the image file itself. Unlike the wealth of static images that are located in the image libraries, these are *dynamic JPEGs*. They are made up entirely of code, with a traditional, static, image at its side to serve as the base picture.

The pictograph representing the resolution is also a *dynamic JPEG*. The pictograph can be one of three images: a very pixelated satellite image, representing low-resolution, a mildly pixelated satellite image, representing mid-resolution, and a crisp satellite image, representing high-resolution. The algorithm determining which is selected is found within the *dynamic JPEG* itself. The process to determine which

resolution-level is appropriate to present is, however, extremely subjective. For the notion of resolution is really dependent upon the geographic properties of the dataset. For instance, it would be fair to deem ten meter resolution satellite imagery covering all of Ontario as high-resolution. However, the same resolution applied to a dataset of satellite imagery focused on downtown Ottawa would not so easily be considered high-resolution. The whole concept is relative. The *Metadata Parser's* system looks at the resolution's units, as well as unit type. The attribution of units to resolution is defined by the following table:

Resolution Interpretation by Units					
METERS		KILOMETERS		DECIMAL DEGREES	
Units	Resolution	Units	Resolution	Units	Resolution
< 10	High	< 1	High	< 0.01	High
10-200	Medium	1-10	Medium	0.01-1	Medium
> 200	Low	> 10	Low	> 1	Low

One might wonder how 500 *meters* can be considered low-resolution, while 0.5 *kilometers* is considered high resolution, given that both represent the same actual distance. This is because the *Metadata Parser* pays heed to the connotation of the *type* of units involved. If the environment is being measured in kilometers, it will use that as its frame of reference to determine resolution. The assumption is the dataset is being measured in those units because it is most reflective of the general dimensions of the data, and will therefore use that as a basis to set notions of resolution. It is a flawed system, because of the unknowns involved. However, it's inclusion into the *Metadata Parser* was substantiated on the basis of its value for demonstration purposes.

Both the datum and ellipsoid are presented much in the same way as the timescale, using dynamic images that superimpose text onto a base picture. However, the text here is first processed to be

shortened. Long names such as “Geodetic Reference System” are recognized and represented as the acronym “GRS.” With the datum, only the first, relevant word construct is taken and incorporated into the pictograph to be shown to the user.

2.4.3 Known Bugs

There are two identified bugs in the *Metadata Parser*. The first occurs if the report is reprocessed via the “Debug Tools” menu, when the user has simultaneously nulled parts of the information stored in the PHP session. This nullification can be achieved by going back a number of steps in the wizard window, to the page of the first step. In that particular page's code, there's a clause to reset the information stored in the session. This resetting allows the user to start afresh and submit a new metadata file for analysis, in line with the primary function of the page. Meanwhile, if this nullification of the session occurs, the report window will display the error demonstrated in the following illustration.

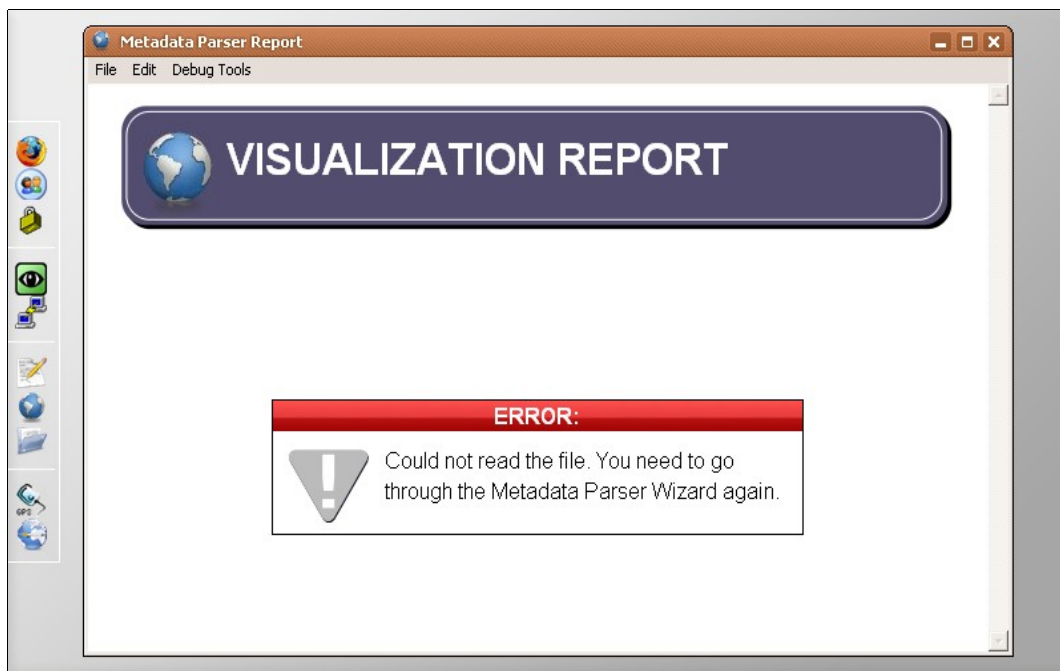


Illustration 26: Error produced by the Metadata Parser if the information stored in the session is invalid.

The second bug is a visual offsetting of the contents of the wizard for the Metadata Parser. The contents can be displaced if a user clicks on a segment of text, and then drags the mouse off the application to the right. A resulting white area of approximately ten pixels at the right edge of the screen will appear. The cause of this bug has not been exactly located, but the nature of the error suggests unknown properties of a styled HTML element.

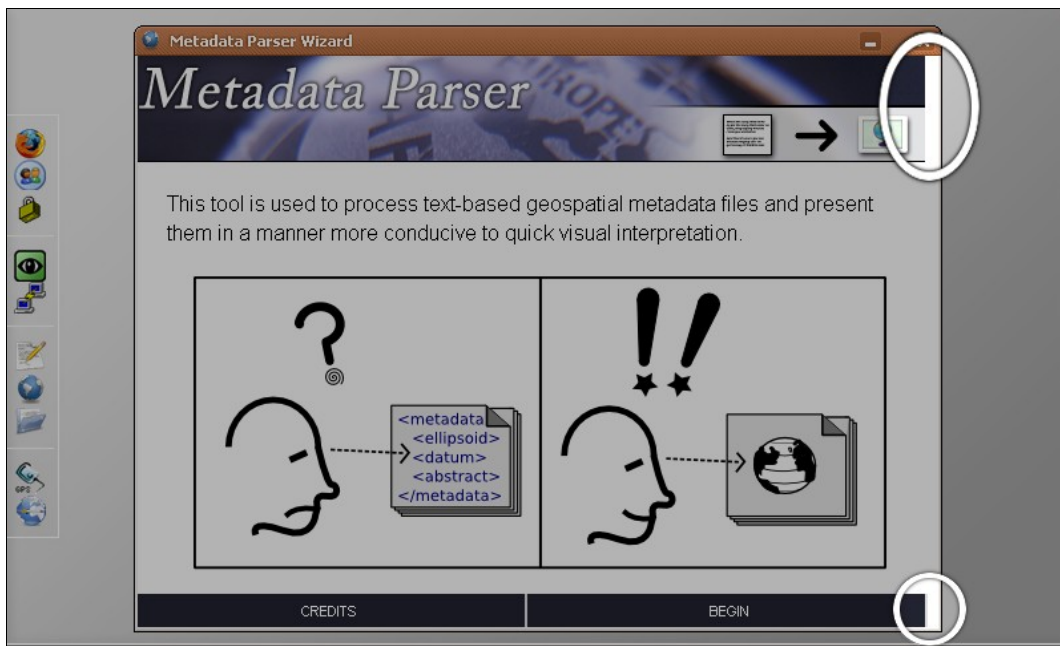


Illustration 27: White space visible from the visual offsetting bug present in the Metadata Parser.

There is another issue worth discussing, although not technically a bug. The *Metadata Parser* only seeks to find one image to represent the nature of the data, the location, and so forth. Therefore if there's more than one location in a metadata file, the *Metadata Parser* will only show the first one it can find a match to in its library of images. Such an adherence to only one item for visualization might induce an inaccurate representation of the dataset to the user. However, the display of all potential matches might at the same time over-saturate the content. There is no simple solution in the matter.

2.5: Discussion

While the standard FGDC parser included with the *Metadata Parser* demonstrated the feasibility of visualizing geospatial metadata, the second multi-file experimental parser, demonstrated potential usage for such a system. An individual using this latter parser would have absorbed the major characteristics of the datasets represented by the metadata files in a directory, without undergoing the arduous task of reading through those metadata files individually. The applicability of such a system becomes immediately apparent. For instance, if an individual is seeking to pick and choose a number of datasets that are appropriate for a project, they can quickly and easily discount datasets of little relevance to their task. All the datasets residing outside the area of study can be identified in one fell swoop. All vector-based datasets can likewise be as easily identified, and kept for their appropriateness.

The parsing of XML files was a significant hindrance in the development of the *Metadata Parser*. It took weeks of research into the PHP language before a workable solution was found, and even then it took months to create workable code. The end result is that despite a relatively polished *Metadata Parser* and single-file parser, the multiple-file parser is lacking. The pictographs sported by the multiple-file parser are high-resolution images reduced in size after the fact by the Internet Explorer drawing engine, which uses a poor resampling technique. The end result is an esthetically displeasing sight. With more development time oriented on the multiple-file parser, this project could have commercial potential.

2.5.1 Code Portability

The *Wapache* framework enables code in languages designed for the web to run independently on a personal computer as a full-fledged application. A positive repercussion such development is that the code can easily be ported to a web server.

In the case of the *Metadata Parser*, a few changes would have to be implemented before the code would be ready to run independently on a web server. For one, the whole design of the wizard would have to be revamped. The code for the wizard was designed to fit in a window of a set resolution. Having this same code displayed in a regular browser window completely changes this dynamic, and the contents as-is would look completely distorted. Furthermore, all of the CSS code would have to be altered to be cross-browser compatible. This is the code that determines how things *look*. In the case of the *Metadata Parser*, it was geared for how Internet Explorer interprets CSS code, which is substantially different from how other browsers such as Mozilla's *Firefox* and Apple's *Safari* interpret cascading style sheets.

Part of the underlying PHP code would also have to change. As it stands, the parser simply reads the metadata file that the user selected directly from the hard drive. The PHP *Zend Engine* is able to do this because it has direct access to the hard drive. However, a web server does not have access to a surfer's hard drive, and therefore would be unable to process the file. The code would have to be altered: first have the user upload the file to the server, which would then have the ability to read it off of its own disk drive.

2.5.2 Development of Plug-Ins

The *Metadata Parser* supports two types of plug-ins: *parsers*, and additional *images* for the pictograph libraries. For their installation, both of these can simply be placed in the appropriate directories within the *Metadata Parser*, and they will automatically be integrated into the program when it's run.

For the development of pictographs, a base image (`blank.png`) is provided in the `/images/buttons/` directory. Additional documentation on the matter (`about.txt`) is also provided in the same folder. The text explains the nuances in the directory structure, the types of images, etc.

Parser development is covered by a separate text file, in the `parsers` directory. The file containing the documentation is also called `about.txt`, and it contains information on the naming conventions of the parser, the structure of the file, the variables provided to it by the *Metadata Parser*, and it explains the purpose of the *called code* files in the `scripts` directory, as well as how to interact with dynamic JPEGs.

A third `about.txt` file is also located inside the root directory of the *Metadata Parser*, and serves to explain the general directory structure of the program. Combined, this information provides a developer with a small pool of documentation from which to begin work on the program.

Users ought to be cautious when running the code of other developers, in the form of plug-ins for the *Metadata Parser*. These plug-ins have direct access to the hard drive, and ill-written or malicious code could cause irreversible changes to the user's computer.

2.5.3 Future Development

The *Metadata Parser* is a relatively polished product, but it is one that is geared towards single-file parsing. While there is nothing inherently wrong with single-file parsing, it is clear to this author that the future of this kind of technology rests in multiple-file parsing. The multiple-file parser included in the experimental release of the *Metadata Parser* is just that: an experiment, an exercise to prove the usefulness of such visualization techniques. Its incorporation into the greater program remains lackluster. For instance, the program is designed to allow the user to choose single metadata files to pass on to the parser, not directories nor sets of files.

The next version of the Metadata Parser should incorporate these lessons, and create an environment more suitable to multiple-file parsing. The correction of bugs would also be a necessity, as well as a look at the current issues with regards to the limitations of displaying but one pictograph per item in the metadata file. In terms of new additions, a feature whereby users could dynamically download new parsers on-the-fly, from within the *Metadata Parser* itself, would be a desirable improvement to work on. Finally, the ability to export the current visualizations into a PDF-compatible format would also be a desirable addition.

SECTION 3
Conclusion

3.1: Conclusion

The *Metadata Parser* is able to interpret FGDC-compliant XML geospatial metadata and produce a document representing the written data as a set of pictographs. The capability for the program to interpret these metadata files is dependent upon the parsers developed for the application. Creating new parsers could make the Metadata Parser visualize metadata files following different data standards, or interpret the same standards in new ways. A framework, consisting of documentation as well as little facilitators within the *Metadata Parser*, are provided to encourage this kind of development.

The *Metadata Parser* is susceptible to a few computer glitches, or *bugs*, but these are minor problems overall. The program is limited by its inability to present more than one pictograph per content item in the metadata, as well as its adherence to a single variant of the FGDC metadata standard. Further development could correct these problems, and enhance the general functionality of the program.

3.2: Notes

1. The description of the *Wapache* framework is extracted from the official site for the project. The link is provided in the reference below.
2. The inner workings of *Wapache* was determined through analysis of the software by this paper's author. Tools such as Microsoft's *Process Explorer* were used for the purposes of the investigation.
3. The statistics on monthly downloads of *Wapache* were provided by SourceForge.net, its web host. The link is provided in the reference below.
4. The information on *ResHack's* author were taken from the application's README file.
5. Sessions, and other concepts of the PHP language are documented on the official Canadian

version of the PHP website. The link is provided in the reference below.

6. General information on *GeoBase* is documented on its own website. The link is provided in the reference below.

3.3: References

Johnson, Angus. *ResHacker*. Vers. 3.4.0. Computer Software. 2002. Windows-Compatible, download.

Michener, William; Brunt, James; Helly, John; Kirchner, Thomas; Stafford, Susan. "Nongeospatial Metadata for the Ecological Sciences." *Ecological Applications* 7.1 (1997): 330-342.

"GeoBase – Home" GeoBase.ca. Natural Resources Canada. 27 March 2008.
<<http://www.geobase.ca/geobase/en/index.html>>

"PHP: Sessions" php.net. The PHP Group. 27 March 2008. <<http://ca.php.net/session>>

"Project Statistics for Wapache" SourceForge.net. SourceForge, Inc. 27 March 2008.
<http://sourceforge.net/project/stats/?group_id=129600&ugn=wapache>

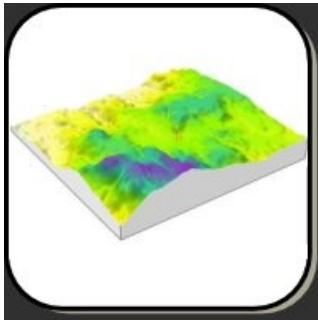
"Wapache – Power of Apache to Go" SourceForge.net. SourceForge, Inc. 27 March 2008.
<<http://wapache.sourceforge.net/>>

Trademark symbols have been attributed where deemed appropriate. However, the lack or presence of such a symbol should not be understood as a legal statement upon the status of the trademark(s) under scrutiny.

APPENDIX I
Pictographs

The following are the pictographs designed for, and included with, the Metadata Parser.

BASIC DATA SETS



Digital Elevation Model



Lakes, Rivers, Hydro



Political



Roads



Satellite

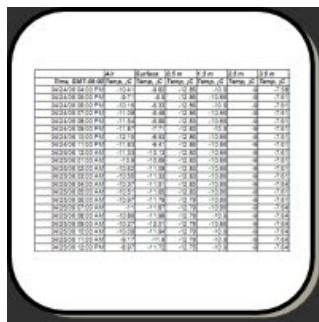


Unknown

DATA TYPES



Raster



Tabular



Vector



Unknown

PLACES: CITIES



Ottawa



Toronto



Unknown

PLACES: PROVINCES/STATES



Alberta



*British Columbia, B.C.,
Colombie-Britannique*



California



Florida



Manitoba



*New Brunswick,
Nouveau Brunswick*



*Newfoundland, Terre
Neuve*



*Northwest Territories,
Territoires du Nord-
Ouest*



Nunavut



Ontario



*Prince Edward
Island, Île-du-Prince-
Édouard*



Quebec, Québec



Saskatchewan



Texas



Yukon



Unknown

PLACES: COUNTRIES/CONTINENTS



Africa



Australia



*Britain, England,
Scotland, United
Kingdom*



Canada



China



Finland



France



Germany



Japan



(South) Korea



Mexico



Russia



South America



United-States, USA



Unknown

RESOLUTION



Low-Resolution



Medium-Resolution



High-Resolution



Unknown

BASE PICTURES FOR DYNAMIC JPEGs



Datum



Ellipsoid



Timescale



General Blank Template

APPENDIX II
Source Code

CREDITS.PHP

```
<HTML>
<HEAD>
<TITLE>Metadata Parser Wizard</TITLE>
<link rel="stylesheet" type="text/css" href="./css/steps.css" />
</HEAD>

<BODY>

<div id="container">
<div id="logo"></div>
<div id="contents">

<span style="width:550px; position: relative; left: -20px;">
This tool is used to process text-based geospatial metadata files and present
them in a manner more conducive to quick visual interpretation. <br /><br />
</span>



</div>
</div>

<ul class="nav_half">
  <li><a href="credits.php">CREDITS</a></li>
  <li><a href="step1.php">BEGIN</a></li>
</ul>

</BODY>
</HTML><html>
<head>
<title>Metadata Parser Wizard</title>
<link rel="stylesheet" type="text/css" href="./css/steps.css" />
</head>

<body>

<div id="container">
<div id="logo"></div>



<div align="center" id="creditsbox">
<marquee scrollamount="2" direction="up" loop="true" width="380">
<center>

<br /><br />

<strong>Developed By</strong> <br />
Julien McArdle <br /><br />

<strong>Original Concept By</strong> <br />
Dr. Mike Sawada, M.A., Ph.D. <br /><br />

The Metadata Parser runs on the Wapache application development environment. You
can find more information on Wapache at the official site:<br />
```

```

<a href="http://wapache.sourceforge.net/">
http://wapache.sourceforge.net/
</a><br /><br />

<strong>Work Based on Code By</strong> <br />
<a href="http://www.willmaster.com/blog/css/show-hide-div-layer.php">
Will Bontrager
</a> (CSS DIV Hide/Show) <br />
<a href="http://php.about.com/od/finishedphp1/qt/file_ext_PHP.htm">
Angela Bradley<
/a> (File Extention IDing)<br />
<a href="http://www.i-fubar.com/random-string-generator.php">
Bill Pellowe
</a> (Random Generator)<br />
<a href="http://www.sitepoint.com/article/php-xml-parsing-rss-1-0">
Kevin Yank
</a> (XML Parsing)<br />
<br />

<strong>Introductory Comic By</strong> <br />
Julien McArdle <br /><br />

<strong>Stock Photography By</strong> <br />
The members of <a href="http://www.sxc.hu/">SXC.HU</a>. <br />
<span style="font:75%/1.4 sans-serif;"><em>Reproduced with
permission.</em></span>
<br /><br />

<strong>Non-System Fonts By</strong> <br />
<a href="http://www.larabiefonts.com/">Ray Larabie</a> <br />
<a href="http://www.zinken.net/">Hans J. Zinken</a><br />
<span style="font:75%/1.4 sans-serif;"><em>Used under license.</em></span>
<br /><br />

<strong>Globe Icon By</strong> <br />
<a href="http://bogdanc.deviantart.com/">Bogdan Condurache</a><br />
<span style="font:75%/1.4 sans-serif;"><em>Free for public release.</em></span>
<br /><br />

<strong>Digital Elevation Model By</strong> <br />
Australian Commonwealth Scientific and Research Organization<br /><br />

<strong>Graphics By</strong> <br />
Julien McArdle <br /><br />

<strong>"Processing Complete" Voice By</strong> <br />
<a href="http://www.research.att.com/~ttsweb/tts/demo.php">AT&T Labs Text-To-
Speech</a><br />
<span style="font:75%/1.4 sans-serif;"><em>Sanctioned for limited non-commercial
use.</em></span>
<br /><br />

<strong>Sounds By</strong> <br />
<a
href="http://www.partnersinrhyme.com/pirsounds/WEB_DESIGN_SOUNDS_WAV/INSTRUME.sh
tml">Partners in Rhyme</a><br />

```



```

<span style="font:75%/1.4 sans-serif;"><em>Used under license.</em></span>
<br /><br />

<?php
/*
This script opens the parser files, and displays their authors in the credits.
*/

$cwd = getcwd();
$dir = "$cwd/parsers";

// Open a known directory, and proceed to read its contents
if (is_dir($dir)) {
if ($dh = opendir($dir)) {
    while (($file = readdir($dh)) !== false) {
        if ($file != "." && $file != ".." && $file != "Thumbs.db" && $file !
= "about.txt" && $file != "auto.php") {
            $nakedfile = substr($file, 0, strrpos($file, '.'));
            echo "<strong><span style=\"text-transform:
uppercase\">\">$nakedfile</span> Parser By</strong><br />\n";
            $readline = file('./parsers/'.$file);
            if ($readline[6] == "") {
                echo "Unidentified";
            } else {
                echo $readline[6];
            }
            echo "<br /><br />\n";
        }
    }
    closedir($dh);
}
}
?>

<strong>Special Thanks</strong> <br />
Dr. Sawada, as well as the development crews behind
<a href="http://www.inkscape.org/">Inkscape</a>,
<a href="http://notepad-plus.sourceforge.net/">Notepad++</a>, and
<a href="http://www.gimp.org/">GIMP</a>.<br />
<br /><br />

Any original materials found within this application
created by Julien McArdle are released free for
non-commercial use in accordance with the terms
dictated by the University of Ottawa. <br /><br /><br />

Additional inquiries can be made at the <br />
following e-mail address:<br />
<a href="mailto:julien@jmcardle.com">julien@jmcardle.com</a>

</center>
</marquee>
</div>
</div>

<ul class="nav_full">

```

```

        <li><a href="index.php">GO BACK</a></li>
</ul>

</body>
</html>

```

INDEX.PHP

```

<HTML>
<HEAD>
<TITLE>Metadata Parser Wizard</TITLE>
<link rel="stylesheet" type="text/css" href="./css/steps.css" />
</HEAD>

<BODY>

<div id="container">
<div id="logo"></div>
<div id="contents">

<span style="width:550px; position: relative; left: -20px;">
This tool is used to process text-based geospatial metadata files and present
them in a manner more conducive to quick visual interpretation. <br /><br />
</span>



</div>
</div>

<ul class="nav_half">
    <li><a href="credits.php">CREDITS</a></li>
    <li><a href="step1.php">BEGIN</a></li>
</ul>

</BODY>
</HTML>

```

REPORT.PHP

```

<?php
session_start( );
?>

<html>
<head>
    <title>Metadata Parser Report</title>
    <link rel="stylesheet" type="text/css" href="./css/report.css" />
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" >

<script type="text/javascript" language="JavaScript">
function HideShow(d) {
if (document.getElementById(d).style.display == "none")

```

```

{ document.getElementById(d).style.display = "block"; }
else { document.getElementById(d).style.display = "none"; }
}

function HideDiv(d) {
document.getElementById(d).style.display = "none";
}

function ShowDiv(d) {
document.getElementById(d).style.display = "block";
}
</script>

<? include("./scripts/idgen.php"); ?>

</head>

<body onResize="DetectSize();">
<bgsound src="./media/complete.wav">

<script type="text/javascript" language="JavaScript">
function DetectSize() {
    if (document.body.clientWidth < 530) {
        HideDiv('parserdiv');
        ShowDiv('widtherrorbox');
    } else if (document.body.clientWidth > 530) {
        ShowDiv('parserdiv');
        HideDiv('widtherrorbox');
    }
}
</script>

<div id="widtherrorbox" class="widtherrorboxclass">
<div class="errorbox"><div class="errortitle">ERROR:</div>
<div class="errorcontent">
The width of this window is too small to properly display the content.
Please increase the width of this window.
</div></div>
</div>

<div id="parserdiv">

<div class="reportheadertext">
<? include("reportbg.php") ?>
<div class="reportheadertext" >

VISUALIZATION REPORT

<div class="reportheadersubtext">
<?php
session_register("metadatainfile");
$metadatafiledisp = str_replace("\\\\", "\\", $metadatainfile);
$wholefilename = strtolower($metadatafiledisp) ;
$filename = split("[\\]", $wholefilename) ;

```

```

        $n = count($filename)-1;
        $filename = $filename[$n];
        echo $filename;
        ?>
    </div>
</div>

<div style="text-align: center;">

<?php
session_register("metadatatype");
if ( $metadatatype == "Automatic" ) {
    include("../parsers/auto.php");
} elseif (file_exists("../parsers/$metadatatype.php")) {
    include("../parsers/$metadatatype.php");
} elseif (file_exists("../parsers/$metadatatype.PHP")) {
    include("../parsers/$metadatatype.PHP");
} else {
echo "<div class=errormegabox><div class=errorbox><div class=errortitle>ERROR:</div>";
echo "<div class=errorcontent><img src=\"../images/error.gif\" style=\"float:left;margin-right:10px;\">";
echo "Could not load the parser. Try going through the Wizard again. If that fails, the parser might not be properly formatted.";
echo "</div></div></div>";
}
?>

</div>
</div>

</body>
</html>

```

REPORTBG.PHP

```

<div class="reportbg_left"></div>
<div class="reportbg_right"></div>
<div class="reportbg_top"></div>
<div class="reportbg_bottom"></div>
<div class="reportbg_topleft"></div>
<div class="reportbg_topright"></div>
<div class="reportbg_bottomleft"></div>
<div class="reportbg_bottomright"></div>

```

STEP1.PHP

```

<?php
session_start( );
?>

<HTML>

```

```

<HEAD>
<TITLE>Metadata Parser Wizard</TITLE>
<link rel="stylesheet" type="text/css" href="./css/steps.css" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
</HEAD>

<BODY>

<?php
session_register("metadatafile");
$metadatafile = "";
?>



<div id="container">
<div id="logo"></div>
<div id="contents">

<span style="width:550px; position: relative; left: -20px;">
This utility only interprets XML-based metadata files (those that
end with <strong>.xml</strong>.) Please select the metadata file you wish
to convert, and press <span class="technicaltext">NEXT</span>.
<BR><BR>
</span>

<div id="selmetabox">
<form name="step1form" action="step2.php" method="POST">

<div id="steps"><em>Please Select a Metadata File...</em></div>
<div id="stepscontents">

<input name="metadatafile" type="file" size="35" />
</div>

</form>
</div>

<br /><br />

</div>
</div>

<ul class="nav_half">
<li><a href="index.php">PREVIOUS</a></li>
<li><a href="javascript:document.step1form.submit();">NEXT</a></li>
</ul>

</BODY>
</HTML>

```

STEP1ERROR1.PHP

```
<HTML>
<HEAD>
<TITLE>Metadata Parser Wizard</TITLE>
<link rel="stylesheet" type="text/css" href="./css/steps.css" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
</HEAD>

<BODY>
<bgsound src="./media/error.wav">



<div id="container">
<div id="logo"></div>
<div id="contents">

<span style="width:550px; position: relative; left: -20px;">
This utility only interprets XML-based metadata files (those that
end with <strong>.xml</strong>.) Please select the metadata file you wish
to convert, and press <span class="technicaltext">NEXT</span>.
<strong><span style="color: red;">You did not select a metadata file.
Please select one using the "Browse..." button below before continuing.</span>
</strong><BR><BR>
</span>

<div id="selmetabox2">
<form name="step1form" action="step2.php" method="POST">

<div id="steps"><em><strong>Please Select a Metadata File...</strong></em></div>
<div id="stepscontents">

<input name="metadatafile" type="file" size="35" />
</div>

</form>
</div>

<br /><br />

</div>
</div>

<ul class="nav_half">
<li><a href="index.php">PREVIOUS</a></li>
<li><a href="javascript:document.step1form.submit();">NEXT</a></li>
</ul>

</BODY>
</HTML>
```

STEP1ERROR2.PHP

```
<HTML>
<HEAD>
<TITLE>Metadata Parser Wizard</TITLE>
<link rel="stylesheet" type="text/css" href="./css/steps.css" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</HEAD>

<BODY>
<bgsound src="./media/error.wav">



<div id="container">
<div id="logo"></div>
<div id="contents">

<span style="width:550px; position: relative; left: -20px;">
This utility only interprets XML-based metadata files (those that
end with <strong>.xml</strong>.) Please select the metadata file you wish
to convert, and press <span class="technicaltext">NEXT</span>.
<strong><span style="color: red;">You selected a metadata file, but it wasn't
in XML. Please select another file using the "Browse..." button below
before continuing.</span>
</strong><BR><BR>
</span>

<div id="selmetabox2">
<form name="step1form" action="step2.php" method="POST">

<div id="steps"><em><strong>Please Select a Metadata File...</strong></em></div>
<div id="stepscontents">

<input name="metadatafile" type="file" size="35" />
</div>

</form>
</div>

<br /><br />

</div>
</div>

<ul class="nav_half">
<li><a href="index.php">PREVIOUS</a></li>
<li><a href="javascript:document.step1form.submit();">NEXT</a></li>
</ul>

</BODY>
</HTML>
```

STEP2.PHP

```
<?php
session_start( );
?>

<html>
<head>
    <title>Metadata Parser Wizard</title>
    <link rel="stylesheet" type="text/css" href="./css/steps.css" />
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" >

    <script type="text/javascript" language="JavaScript">
        function ErrorScreen1() {
            window.location = 'steplerror1.php'
        }
        function ErrorScreen2() {
            window.location = 'steplerror2.php'
        }
    </script>

</head>

<?php
/*
This is the error-checking code. For the first error, it checks to see whether a
metadata file was selected at all. If none was, it'll send the user to a
modified first step page, which contains details on the error.

For the second error, it checks to see whether it was an XML file that was
selected. Likewise to the first error, if this test fails, the user will be sent
to a specially modified first step page. Previously, if the user had had an
error, but then corrected the situation, went onwards to the following pages
... but then went back for some reason, then the error page would still show.
Additional code was added to correct that situation. If the user fucks up,
corrects himself, and goes back for whatever reason step-wise, then the correct
non-error page will be displayed.

All basic variables are stored in the session. This is how they are remembered
from page to page.
*/

session_register("metadatafile");
if ( $_POST["metadatafile"] == "" ) {
    if ( $metadatafile == "" ) {
        echo "<body onLoad=\"ErrorScreen1();\">\n";
    }
    if ( $metadatafile != "" ) {
        $filename = strtolower($metadatafile) ;
        $exts = split("/\\.", $filename) ;
        $n = count($exts)-1;
        $exts = $exts[$n];
        if ($exts == "xml" ) {
            echo "<body>\n";
        } else {
            echo "<body onLoad=\"ErrorScreen2();\">\n";
        }
    }
}
```



```

    }
} else {
    $metadatafile = $_POST["metadatafile"];
    $filename = strtolower($metadatafile) ;
    $exts = split("/\\\.]", $filename) ;
    $n = count($exts)-1;
    $exts = $exts[$n];
    if ($exts == "xml" ) {
        echo "<body>\n";
    } else {
        echo "<body onLoad=\"ErrorScreen2();\">\n";
    }
}
?>

```

```

```

```
<div id="container">
    <div id="logo"></div>
    <div id="contents">
```

```

        <span style="width:550px; position: relative; left: -20px;">
For this second step, please choose the standard of the Metadata file. If
you don't know or are otherwise unsure, you can choose "Automatic" to have the
Wizard try to guess the appropriate format for you.<BR><BR>
        </span>

```

```
<div id="selmetabox">
```

```

    <form action="step3.php" name="step2form" method="POST">
        <div id="steps"><em>Please Select the Metadata
Type...</em></div>

```

```

        <?php
        /*

```

This script counts the amount of files in the parser directory. If there are only two parsers or less, then we don't want the unsightly scroll bar to show. If there are more than two parsers, then we need the scroll bar to show.

This script is part of the greater drag 'n drop parser idea. While the next script down assists with listing the parsers, this script assists with the display of the grey box that encompasses them.

```

        */
        $cwd = getcwd();
        $dir = "$cwd/parsers/";

        /* To make up for glitches in the count() function */
        if (count(glob($dir . "*.php")) == "1") {
            $count_lowercase = "0";
        } else {
            $count_lowercase = count(glob($dir . "*.php"));
        }

        if (count(glob($dir . "*.PHP")) == "1") {
            $count_uppercase = "0";

```

```

    } else {
        $count_uppercase = count(glob($dir . "*.php"));
    }

    $countparsers = $count_lowercase + $count_uppercase;

    if ($countparsers >= 4 ) {
        echo "<div id=\"stepscontents2overflow\">\n";
    } else {
        echo "<div id=\"stepscontents2\">\n";
    }
    ?>

    <INPUT TYPE="RADIO" NAME="metadatatype" VALUE="Automatic"
CHECKED>Automatic<BR>

    <?php
    /*
This is the script that makes adding new parsers via 'drag and drop' possible.
Essentially, it looks in the parsers folder for all files. It lists all files
here, except for the special automatic parser file. It then reads a specific
line of the parser file (line #4) to get the HTML code to insert here as the
description.

The filename of the parser becomes the value of the type of parser used behind
the scenes by the application. The parser description line inside the parser is
what is used to be publicly displayed as the English-readable
title.
*/

    $cwd = getcwd();
    $dir = "$cwd/parsers";

    // Open a known directory, and proceed to read its contents
    if (is_dir($dir)) {
    if ($dh = opendir($dir)) {
        while (($file = readdir($dh)) !== false) {
            if ($file != "." && $file != ".." && $file !=
"Thumbs.db" && $file != "about.txt" && $file != "auto.php") {
                $nakedfile = substr($file, 0, strpos($file, '.'));
                echo "<INPUT TYPE=\"RADIO\" NAME=\"metadatatype\"
VALUE=\"\$nakedfile\">";

                $readline = file('./parsers/'.$file);
                echo $readline[3];
                echo "<br />\n";
            }
        }
        closedir($dh);
    }
    }
    ?>

    </div>
</form>

```

```

        </div>

    </div>
</div>

<ul class="nav_half">
    <li><a href="step1.php">PREVIOUS</a></li>
    <li><a href="javascript:document.step2form.submit();">NEXT</a></li>
</ul>

</body>
</html>

```

STEP3.PHP

```

<?php
session_start( );
?>

<html>
<head>
<title>Metadata Parser Wizard</title>
<link rel="stylesheet" type="text/css" href="./css/steps.css" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
</head>
<body>



<div id="container">
    <div id="logo"></div>
    <div id="contents">

        <span style="width:550px; position: relative; left: -20px;">
            Finally, check the details below to make sure that the details are
            correct. If they are, press <span
class="technicaltext">FINISH</span>
            and the Metadata Parser will process the file. Otherwise, use the
            <span class="technicaltext">PREVIOUS</span> buttons to revise your
            choices.<BR><BR>
        </span>

        <div class="confirmdetailsbox">
            <div class="confirmdetailstitle">
                 &nbsp;   Confirm Metadata Selection
            </div>

            <div class="mdfileform" nowrap="true">
                <strong>Metadata Parser Wizard</strong><br />
                <em>&copy; 2007-2008 Julien McArdle</em><br /><br />

                <strong>File: </strong>
            <?php
session_register("metadatafile");

```

```

        $metadatatype = str_replace("\\\\", "\\", $metadatatype);
        echo $metadatatype;
        ?>

        <br />

        <strong>Type: </strong>
        <?php
        session_register("metadatatype");
        $metadatatype = $_POST["metadatatype"];
        print $metadatatype;
        ?>
    </div>
</div>
</div>
</div>

<ul class="nav_half">
    <li><a href="step2.php">PREVIOUS</a></li>
    <li><a href="report.php" target="Report">FINISH</a></li>
</ul>

</body>
</html>

```

CSS/STEPS.PHP

```

body {
    background-color:#fff;
    font:100.01%/1.4 sans-serif;
    cursor: default;
}

#container {
    position: absolute;
    background-color:#fff;
    top: 0px;
    left: 0px;
    width: 600px;
    border: 0px solid #000;
    text-align:left;
}

#logo {
    position: absolute;
    top: 0px;
    height: 80px;
    color: fff;
    background-color: #000050;
    background-image: url("../images/logo.png");
    width: 600px;
    margin: 0px auto;
    border: 0px;
}

```

```
#contents {
  position: absolute;
  top: 80px;
  padding: 20px 10px 10px 50px;
}

#selmetabox {
  position: absolute;
  left: 100px;
  top: 100px;
  width: 400px;
  background-color: #f5f5f5;
  border: 1px solid #000;
}

#selmetabox2 {
  position: absolute;
  left: 100px;
  top: 130px;
  width: 400px;
  background-color: #f5f5f5;
  border: 1px solid #000;
}

#steps {
  color: #fff;
  font: 150.01%/1.4 sans-serif;
  background-color: #000;
  width: 100%;
  padding: 10px 0px 4px 0px;
  text-align: center;
}

#stepscontents {
  padding: 20px 0 0 0;
  text-align: center;
}

#stepscontents2 {
  padding: 20px 0 0 25px;
  text-align: left;
  height: 70px;
}

#stepscontents2overflow {
  padding: 20px 0 0 25px;
  overflow-y: scroll;
  text-align: left;
  height: 70px;
}

#bottomcredits {
  position: absolute;
  bottom: 0px;
}
```

```

    left: 0px;
    width: 604px;
    font: 7pt/1.4 sans-serif;
    background-color: #eee;
    text-align: right;
}

#creditsbox {
    position: relative;
    top: 120px;
}

ul.nav_half{
    list-style:none;
    background:#222330;
    width: 602px;
    position: absolute;
    bottom: 0px;
    left: -40px;
}

.nav_half li{
    border-right:2px solid #DDD;
    text-align: center;
    float:left;
    display:block;
    width:50%;
}

.nav_half li a{
    font-size:11px;
    outline:none;
    color:#fff;
    text-decoration:none;
    display:block;
    padding:5px 0 5px 5px;
    width:100%;
}

.nav_half li a:hover{
    background:#524d6e;
    color:#fff!important;
    width:100%;
}

ul.nav_full{
    list-style:none;
    background:#222330;
    width: 610px;
    position: absolute;
    left: -50px;
    bottom: 0px;
}

.nav_full li{

```

```

        text-align: center;
        float:left;
        display:block;
        width:100%;
    }

.nav_full li a{
    font-size:11px;
    outline:none;
    color:#fff;
    text-decoration:none;
    display:block;
    padding:5px 0 5px 5px;
    width:100%;
}

.nav_full li a:hover{
    background:#524d6e;
    color:#fff!important;
    width:100%;
}

img.credits_bg {
    position: absolute;
    top: 80px;
    right: 0px;
}

img.steps_img {
    position: absolute;
    right: 0px;
    bottom: 0px;
}

.technicaltext {
    font-weight: bold;
    font-family: "Courier New" Courier monospace;
    font-size: 100%;
    color: #524d6e;
}

.mdfileform {
    height:110px;
    width:480px;
    overflow-x: scroll;
    display:block;
    border: 1px solid #888;
    font-family: "Courier New" Courier monospace;
    font-size: 80%;
}

.confirmdetailsbox {
    border: 1px solid #000;
    width: 482px;
    background-color: #fff;
}

```

```

        background-image: url("../images/confirm_bg.jpg");
    }

    .confirmdetailstitle {
        width: 100%;
        color: #fff;
        padding: 0 0 4px 0;
        text-align: center;
        font: 100.01%/1.4 sans-serif;
        font-weight: bold;
        background-color: #524d6e;
    }

    IMG.tinyglobe {
        position: relative;
        top: 5px;
        left: 5px;
    }

```

CSS/REPORT.PHP

```

body {
    background-color:#fff;
    font:100.01%/1.4 sans-serif;
    cursor: default;
}

.widtherrorboxclass {
    text-align: center;
    display: none;
    position: relative;
    top: 40%;
}

.reportheadertext {
    background-color:#524d6e;
    width: 95%;
    height: 90px;
    position: relative;
    left: 15px;
    z-index: 1;
}

.reportheadertext {
    font:200.01%/1.4 sans-serif;
    font-weight: bold;
    text-align:left;
    position: absolute;
    left: 16px;
    top: 16px;
    color: #fff;
    z-index: 5;
}

```



```
.reportheadersubtext {
    font:100.01%/1.4 sans-serif;
    font-weight: bold;
    font-size: 12px;
    color: #fff;
    z-index: 5;
}

.reportbg_left {
    background-image: url("../images/reportbg_left.gif");
    background-repeat: repeat-y;
    width: 32px;
    height: 100%;
    position: absolute;
    left: 0px;
    z-index: 2;
}

.reportbg_right {
    background-image: url("../images/reportbg_right.gif");
    background-repeat: repeat-y;
    width: 32px;
    height: 100%;
    position: absolute;
    right: -1px;
    z-index: 2;
}

.reportbg_top {
    background-image: url("../images/reportbg_top.gif");
    background-repeat: repeat-x;
    width: 100%;
    height: 32px;
    position: absolute;
    top: 0px;
    z-index: 3;
}

.reportbg_bottom {
    background-image: url("../images/reportbg_bottom.gif");
    background-repeat: repeat-x;
    width: 100%;
    height: 32px;
    position: absolute;
    bottom: 0px;
    z-index: 3;
}

.reportbg_bottomleft {
    background-image: url("../images/reportbg_bottomleft.gif");
    background-repeat: no-repeat;
    width: 32px;
    height: 32px;
    position: absolute;
    bottom: 0px;
    left: 0px;
}
```

```

        z-index: 4;
    }

    .reportbg_bottomright {
        background-image: url("../images/reportbg_bottomright.gif");
        background-repeat: no-repeat;
        width: 32px;
        height: 32px;
        position: absolute;
        bottom: 0px;
        right: -1px;
        z-index: 4;
    }

    .reportbg_topleft {
        background-image: url("../images/reportbg_topleft.gif");
        background-repeat: no-repeat;
        width: 32px;
        height: 32px;
        position: absolute;
        top: 0px;
        left: 0px;
        z-index: 4;
    }

    .reportbg_topright {
        background-image: url("../images/reportbg_topright.gif");
        background-repeat: no-repeat;
        width: 32px;
        height: 32px;
        position: absolute;
        top: 0px;
        right: -1px;
        z-index: 4;
    }

    .errorbox {
        width: 400px;
        border: 1px solid #000;
        text-align: left;
    }

    .errortitle {
        background-image: url("../images/redtitlebg.jpg");
        background-repeat: repeat-x;
        width: 100%;
        background-color: #9f0000;
        text-align: center;
        color: #fff;
        font-weight: bold;
    }

    .errorcontent {
        padding: 10 10 10 10px;
    }

```

```

img.reportglobe {
    float: left;
    margin-right: 10px;
}

.metablock {
    border: 1px solid #000;
    margin-top: 20px;
    position: relative;
    width: 90%;
}

.metatitle {
    font-weight: bold;
    color: #fff;
    width: 100%;
    text-align: center;
    background-color: #252525;
    background-image: url("../images/blacktitlebg.jpg");
    background-repeat: repeat-x;
}

.contacttitle {
    font-weight: bold;
    color: #fff;
    width: 100%;
    text-align: center;
    background-color: #262648;
    background-image: url("../images/bluetitlebg.jpg");
    background-repeat: repeat-x;
}

img.expand {
    position: absolute;
    left: 5px;
    top: 5px;
    border: 0px;
}

.metacontent {
    background-color: #2f2f2f;
    border-top: 1px solid #eee;
    text-align: center;
    padding: 30px 30px 30px 30px;
}

.smallbuttonsbox {
    position: absolute;
    background-image: url("../images/smallbuttonsgif.gif");
    top: 5px;
    right: 20px;
    width: 152px;
    height: 44px;
    text-align: center;
    padding-top: 5px;
}

```

```

padding-left: 7px;
}

.listitem_white {
position: relative;
width: 100%;
height: 40px;
background-color: #fff;
border-bottom: 2px #ccc;
padding: 5px 5px 5px 5px;
}

.listitem_grey {
position: relative;
width: 100%;
height: 40px;
background-color: #f0f0f0;
border-bottom: 2px #ccc;
padding: 5px 5px 5px 5px;
}

.descriptioncontent {
background-color: #f0f0f0;
border-top: 1px solid #eee;
text-align: left;
padding: 10px 10px 10px 10px;
}

img.button {
margin: 15px 15px 15px 15px;
}

img.buttonsmall {
margin-right: 5px;
height: 35px;
width: 35px;
}

.contacttextbg {
background-image:url("../images/leftbar.png");
background-repeat: repeat-y;
position:relative;
background-color: #ffffff;
text-align: left;
padding: 10px 10px 10px 50px;
margin-top: 15px;
margin-bottom: 15px;
}

.errormegabox {
position:relative;
top:30%;
text-align:center;
}

img.contacticons {

```

```

    position: absolute;
    left: -50px;
    top: 0px;
}

```

IMAGES/BUTTONS/DATUM/DATUM.DJPG

```

<?php
header("Content-type: image/jpeg");

/* Get values as a string from the url... */
$datum = $_GET['datum'];

/* Set basic parameters */
$image = ImageCreateFromPNG("./blank.png");
$font = '../huskysta.ttf';
$textcolor = imagecolorallocate($image, 0, 0, 0);

/*
The first line of text. First, it sets the font to a default
of 36 points height. Then it sees whether the resulting text is too
wide for our image of 200x200 pixels. If it is, it decreases the font
size and keeps doing so until it all fits on one line. It then uses
that same size function to determine how to center the text. Finally,
it displays the text.
*/

$fontsize = 36;
$size = imagettfbbox($fontsize, 0, $font, $datum);
while ($size[2] > 180) {
    $fontsize--;
    $size = imagettfbbox($fontsize, 0, $font, $datum);
}

/*
This is what centers the font. It finds the size that will be consumed
by the fontbox, takes half that size, and subtracts it from the midpoint of
the whitespace in the blank button. The end result is where the left-hand side
of the textbox should be with respect to blank image in order to have the text
centered.
*/

$sizedatum = (95 - (($size[2]) / 2));
imagefttext($image, $fontsize, 0, $sizedatum, 100, $textcolor, $font, $datum);

/* Displaying the image */
ImageJPEG($image);
ImageDestroy($image);
?>

```

IMAGES/BUTTONS/ELLIPSOID/ELLIPSOID.DJPG

```
<?php
header("Content-type: image/jpeg");

/* Get values as a string from the url... */
$ellipsoid = $_GET['ellips'];

/*
This turns long strings into abbreviations. So "Geodetic Reference System"
becomes "GRS."
*/
$ellipsoid = str_replace("Geodetic Reference System", "GRS", $ellipsoid);
$ellipsoid = str_replace("Système de référence géodésique de", "SRG",
$ellipsoid);

/* Set basic parameters */
$image = ImageCreateFromPNG("../blank.png");
$font = '../huskysta.ttf';
$textcolor = imagecolorallocate($image, 0, 0, 0);

/*
The first line of text. First, it sets the font to a default
of 36 points height. Then it sees whether the resulting text is too
wide for our image of 200x200 pixels. If it is, it decreases the font
size and keeps doing so until it all fits on one line. It then uses
that same size function to determine how to center the text. Finally,
it displays the text.
*/

$fontsize = 36;
$size = imagettfbbox($fontsize, 0, $font, $ellipsoid);
while ($size[2] > 180) {
    $fontsize--;
    $size = imagettfbbox($fontsize, 0, $font, $ellipsoid);
}

/*
This is what centers the font. It finds the size that will be consumed
by the fontbox, takes half that size, and subtracts it from the midpoint of
the whitespace in the blank button. The end result is where the left-hand side
of the textbox should be with respect to blank image in order to have the text
centered.
*/

$sizeellipsoid = (95 - (($size[2]) / 2));
imagefttext($image, $fontsize, 0, $sizeellipsoid, 100, $textcolor, $font,
$ellipsoid);

/* Displaying the image */
ImageJPEG($image);
ImageDestroy($image);
?>
```

IMAGES/BUTTONS/RESOLUTION/RESOLUTION.DJPG

```
<?php
header("Content-type: image/jpeg");

/* Get values as a string from the url... */
$resunits = $_GET['units'];
$restype = $_GET['type'];

/*
This is the script that tries to match the resolution
to a picture representing high, medium, or low resolution.
Of course, resolution is a very relative notion, so this
isn't a very reliable means of expressing resolution.

First the script looks at the unit type (ie. decimal
degrees, meters, etc.) It then assigns a hi/med/low
resolution image based on the amount of units for that
unit type.
*/
if (preg_match("/decimal/i", $restype) || preg_match("/décimaux/i", $restype)) {
    if ($restype < 0.01) {
        $image = ImageCreateFromJPEG("./hires.jpg");
    } elseif ($restype < 1) {
        $image = ImageCreateFromJPEG("./medres.jpg");
    } else {
        $image = ImageCreateFromJPEG("./lowres.jpg");
    }
} elseif (preg_match("/kilom/i", $restype)) {
    if ($restype < 1) {
        $image = ImageCreateFromJPEG("./hires.jpg");
    } elseif ($restype < 10) {
        $image = ImageCreateFromJPEG("./medres.jpg");
    } else {
        $image = ImageCreateFromJPEG("./lowres.jpg");
    }
} elseif (preg_match("/meter/i", $restype) || preg_match("/mètre/i", $restype))
{
    if ($restype < 10) {
        $image = ImageCreateFromJPEG("./hires.jpg");
    } elseif ($restype < 200) {
        $image = ImageCreateFromJPEG("./medres.jpg");
    } else {
        $image = ImageCreateFromJPEG("./lowres.jpg");
    }
} else {
$image = ImageCreateFromJPEG("./unknown.jpg");
}

/* Displaying the image */
ImageJPEG($image);
ImageDestroy($image);
?>
```

IMAGES/BUTTONS/TIME/TIME.DJPG

```
<?php
header("Content-type: image/jpeg");

/* Get values as a string from the url... */
$begdate = $_GET['begdate'];
$separator = "-";
$enddate = $_GET['enddate'];

/* Set basic parameters */
$image = ImageCreateFromJPEG("./blank.jpg");
$font = '../huskysta.ttf';
$textcolor = imagecolorallocate($image, 0, 0, 0);

/*
The first line of text. First, it sets the font to a default
of 36 points height. Then it sees whether the resulting text is too
wide for our image of 200x200 pixels. If it is, it decreases the font
size and keeps doing so until it all fits on one line. It then uses
that same size function to determine how to center the text. Finally,
it displays the text.
*/

$fontsize = 36;
$size = imagettfbbox($fontsize, 0, $font, $begdate);
while ($size[2] > 180) {
    $fontsize--;
    $size = imagettfbbox($fontsize, 0, $font, $begdate);
}

/*
$sizebeg is what centers the font. It finds the size that will be consumed
by the fontbox, takes half that size, and subtracts it from the midpoint of
the whitespace in the blank button. The end result is where the left-hand side
of the textbox should be with respect to blank image in order to have the text
centered.
*/

$sizebeg = (95 - (($size[2]) / 2));
imagefttext($image, $fontsize, 0, $sizebeg, 75, $textcolor, $font, $begdate);

/* Second line of text. Note that the fontsize is purposefully reset. */

$fontsize = 36;
$size = imagettfbbox($fontsize, 0, $font, $separator);
$sizesep = (95 - (($size[2]) / 2));
imagefttext($image, $fontsize, 0, $sizesep, 105, $textcolor, $font, $separator);

/* Third and final line of text. */
$fontsize = 36;
$size = imagettfbbox($fontsize, 0, $font, $enddate);
while ($size[2] > 180) {
    $fontsize--;
    $size = imagettfbbox($fontsize, 0, $font, $enddate);
}
```



```

$size = imagettfbbox($fontsize, 0, $font, $enddate);
$sizeend = (95 - (($size[2]) / 2));
imagefttext($image, $fontsize, 0, $sizeend, 135, $textcolor, $font, $enddate);

/* Displaying the image */
ImageJPEG($image);
ImageDestroy($image);
?>

```

PARSERS/AUTO.PHP

```

<?php
/*
Parser official title: (always on line 4, HTML formatted)
Automatic

Author: (always on line 7, HTML tags supported)
Julien McArdle
*/

include("../parsers/fgdc.php");
?>

```

PARSERS/FGDC.PHP

```

<?php
/*
Parser official title: (always on line 4, HTML formatted)
FGDC (<em>Optimized for GeoBase</em>)

Author: (always on line 7, HTML tags supported)
Julien McArdle

Information you should know:
This parser was optimized to interpret FGDC formatted metadata files from
GeoBase. The selected tags here that are extracted reflect the tags being used
by GeoBase. Likewise, the way some tags are interpreted reflect their usage from
GeoBase. Take $this->horizdn for instance, which is the tag that contains the
Datum. Here we extract the first word in that tag, as GeoBase uses abbreviations
followed by a potentially long string... ie. "NAD83 Blah Blah Blah"
becomes "NAD83." However, if another producer of metadata uses that tag in a
different manner, some info might be prematurely cut off by the fact that we do
only take the first word.
*/

include("../scripts/findbutton.php");
?>

<?php
/*
This is the XML parser. Not really the most elegant thing, but it is fully
functional. It extracts the appropriate tags ie. cntaddress) out of the main

```

root tags (ie. IDINFO.) If need be, it'll do a bit of processing to the text to make it more suitable for parsing (ie. trim white space with the trim() function.) What happens then is dependent upon the nature of the button. "Buttons" are what I call the 200x200 pixel images that make up the visual component of the resulting report.

In some cases, the script will call the findbutton function and try to find a matching image to the string of text within the XML tag. In other cases, it will call a dynamic JPEG (.djpg) file and do per-image processing. For instance, the date range button works this way. The code for the processing of those images is contained within the .djpg files themselves.

As for the need of the \$divIDs: unique IDs are assigned to the DIVs that make up the end report. This is what enables specific DIVs from being closed/open with the expand/collapse buttons. We have to generate random DIVs because we cannot assign static values, in case the root tag repeats itself and generates two boxes in the same category. We also cannot rely on using values within the page, in case of repetition again. The likelihood of collision here is 36^8 divided by the number of unique elements present. If there were collisions, then the collapse/expand function would close/expand an extra item it wasn't intended to.

Format of variables being pulled by the underlying XML parser PHP class:

```
$this->tag
```

...Where "tag" is the XML tag we want from within the greater root tag.

The code for this parser is originally based on an RSS parsing script from SitePoint. You can check out the original work here:

<http://www.sitepoint.com/article/php-xml-parsing-rss-1-0>

```
*/
```

```
$metadata = $metadatafile;
```

```
class XMLParser {
```

```
    var $insideitem = false;
```

```
    function startElement($parser, $tagName, $attrs) {
```

```
        if ($this->insideitem) {
```

```
            $this->tag = $tagName;
```

```
        } elseif ($tagName == "IDINFO") {
```

```
            $this->insideitem = true;
```

```
        } elseif ($tagName == "SPREF") {
```

```
            $this->insideitem = true;
```

```
        } elseif ($tagName == "METAINFO") {
```

```
            $this->insideitem = true;
```

```
        }
```

```
    }
```

```
    function endElement($parser, $tagName) {
```

```
        if ($tagName == "IDINFO") {
```

```
            $divID = idgen(8);
```

```
            $divID2 = idgen(8);
```

```
            $divID3 = idgen(8);
```

```
            echo "<div class=\"metablock\">\n";
```

```
            echo "<div class=\"metatitle\">\n";
```

```

        echo "<a
href=\"javascript:HideShow('$divID');HideShow('$divID2');HideShow('$divID3');\">
\n";
        echo "<div id=\"\$divID2\"><img src=\"./images/collapse.png\"
class=\"expand\"></div>\n";
        echo "<div id=\"\$divID3\" style=\"display: none\"><img src=\"./
images/expand.png\" class=\"expand\"></div>\n";
        echo "</a>\n";
        $title = trim($this->title);
        echo "$title, $this->sername</div>\n\n";
        echo "<div id=\"\$divID\" class=\"metacontent\">\n";

        $titledataout = findbutton("data", $this->title);
        if ($titledataout == "nothing found") {
            $seriesdataout = findbutton("data", $this->sername);
            if ($seriesdataout == "nothing found") {
                echo "<img
src=\"./images/buttons/data/unknown.jpg\" class=\"button\">\n";
                } else {
                echo "$seriesdataout \n";
                }
            } else {
            echo "titledataout \n";
        }

        $geoformout = findbutton(geoform, $this->geoform);
        if ($geoformout == "nothing found") {
            echo "<img src=\"./images/buttons/geoform/unknown.jpg\"
class=\"button\">\n";
            } else {
            echo "$geoformout <br />\n";
        }

        $localplaceout = findbutton("place/local", $this->placekey);
        if ($localplaceout == "nothing found") {
            $provincialplaceout = findbutton("place/provincial",
$this->placekey);
            if ($provincialplaceout == "nothing found") {
                $nationalplaceout = findbutton("place/national",
$this->placekey);
                if ($nationalplaceout == "nothing found") {

                    echo "<img
src=\"./images/buttons/place/unknown.jpg\" class=\"button\">\n";
                    } else {
                    echo "$nationalplaceout \n";
                    }
                } else {
                echo "$provincialplaceout \n";
                }
            } else {
            echo "$localplaceout \n";
        }

        $begdate = trim($this->begdate);

```

```

        $enddate = trim($this->enddate);
        echo "<img src=\"./images/buttons/time/time.djjpg?begdate=
$begdate&enddate=$enddate\" class=\"button\"> <br />\n";

        echo "</div></div>\n\n";
        $this->title = $this->sername = $this->geoform = $this-
>placekey = "";
        $this->begdate = $this->enddate = $this->cntper = "";
        $this->cntorg = $this->address = $this->city = $this->state =
"";

        $this->country = $this->cntvoice = $this->cntemail = "";
        $this->insideitem = false;

    } elseif ($tagName == "SPREF") {
        $divID = idgen(8);
        $divID2 = idgen(8);
        $divID3 = idgen(8);
        echo "<div class=\"metablock\">\n";
        echo "<div class=\"metatitle\">\n";
        echo "<a
href=\"javascript:HideShow('$divID');HideShow('$divID2');HideShow('$divID3');\">
";
        echo "<div id=\"\$divID2\"><img src=\"./images/collapse.png\"
class=\"expand\"></div>\n";
        echo "<div id=\"\$divID3\" style=\"display: none\"><img src=\"./
images/expand.png\" class=\"expand\"></div>\n";
        echo "</a>\n";
        echo "Additional Technical Details</div>\n";
        echo "<div id=\"\$divID\" class=\"metacontent\">";

        $latres = trim($this->latres);
        echo "<img src=\"./images/buttons/resolution/resolution.djjpg?
units=$latres&type=$this->geogunit\" alt=\"Resolution\" class=\"button\">\n";

        /* This miniscript (short_horizdn) takes the typically very
long $this->horizdn string, and takes only
the first word. So "NAD83 blah blah blah" becomes "NAD83." This
could cause problems in the future
if you use compatible metadata files of a different standards
variant. */
        $pos = array_keys(str_word_count($this->horizdn, 2));
        $short_horizdn = substr($this->horizdn, 0, $pos[1]);
        echo "<img src=\"./images/buttons/datum/datum.djjpg?datum=
$short_horizdn\" class=\"button\">\n";
        echo "<img src=\"./images/buttons/ellipsoid/ellipsoid.djjpg?
ellips=$this->ellips\" class=\"button\">\n";

        echo "</div></div>\n\n";
        $this->latres = $this->longres = $this->geogunit = $this-
>horizdn = $this->ellips = "";
        $this->insideitem = false;

    } elseif ($tagName == "METAINFO") {
        $divID = idgen(8);
        $divID2 = idgen(8);
        $divID3 = idgen(8);

```

```

        echo "<div class=\"metablock\">\n";
        echo "<div class=\"contacttitle\">\n";
        echo "<a
href=\"javascript:HideShow('$divID');HideShow('$divID2');HideShow('$divID3');\">
\n";
        echo "<div id=\"$divID2\"><img
src=\"./images/collapse.png\" class=\"expand\"></div>\n";
        echo "<div id=\"$divID3\" style=\"display: none\"><img
src=\"./images/expand.png\" class=\"expand\"></div>\n";
        echo "</a> $this->cntorg</div>\n";
        echo "<div id=\"$divID\" class=\"metacontent\">\n";
        echo "<div class=\"contacttextbg\">";
        echo "<img src=\"./images/contactaddress.png\"
class=\"contacticons\" style=\"margin-top:22px;\">";
        echo "<strong>$this->cntper</strong> <br />\n";
        echo "$this->address <br />\n";
        $city = trim($this->city);
        $state = trim($this->state);
        $country = trim($this->country);
        echo "$city, $state, $country";
        echo "<img src=\"./images/whitebg_topright.png\"
style=\"position:absolute;top:0px;right:29px;\">";
        echo "</div>\n";
        echo "<div class=\"contacttextbg\"><img
src=\"./images/contacttelephone.png\" class=\"contacticons\">$this-
>cntvoice</div>\n";
        echo "<div class=\"contacttextbg\"><img
src=\"./images/contactemail.png\" class=\"contacticons\" style=\"margin-top:
3px;\">$this->cntemail </div>\n";
        echo "</div></div>\n\n";
        $this->cntorg = $this->address = $this->city = $this->state =
"";
        $this->cntper = $this->country = $this->cntvoice = $this-
>cntemail = "";
        $this->insideitem = false;
    }
}

function characterData($parser, $data) {
    if ($this->insideitem) {
        switch ($this->tag) {
            case "SERNAME":
                $this->sername .= $data;
                break;
            case "GEOFORM":
                $this->geoform .= $data;
                break;
            case "SRCSCALE":
                $this->srcscale .= $data;
                break;
            case "ONLINK":
                $this->onlink .= $data;
                break;
            case "LATRES":
                $this->latres .= $data;

```

```

        break;
        case "LONGRES":
            $this->longres .= $data;
            break;
        case "GEOGUNIT":
            $this->geogunit .= $data;
            break;
        case "HORIZDN":
            $this->horizdn .= $data;
            break;
        case "ELLIPS":
            $this->ellips .= $data;
            break;
        case "CNTORG":
            $this->cntorg .= $data;
            break;
        case "ADDRESS":
            $this->address .= $data;
            break;
        case "CITY":
            $this->city .= $data;
            break;
        case "STATE":
            $this->state .= $data;
            break;
        case "COUNTRY":
            $this->country .= $data;
            break;
        case "CNTVOICE":
            $this->cntvoice .= $data;
            break;
        case "CNTEMAIL":
            $this->cntemail .= $data;
            break;
        case "TITLE":
            $this->title .= $data;
            break;
        case "PLACEKEY":
            $this->placekey .= $data;
            break;
        case "BEGDATE":
            $this->begdate .= $data;
            break;
        case "ENDDATE":
            $this->enddate .= $data;
            break;
        case "CNTPER":
            $this->cntper .= $data;
            break;
    }
}
}

$xml_parser = xml_parser_create();
$metadata_parser = new XMLParser();

```

```

xml_set_object($xml_parser, &$metadata_parser);
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characterData");

$fp = fopen($metadata, 'r') or die("<div class=\"errormegabox\"><div
class=errorbox>\n<div class=errortitle>ERROR:</div>\n<div
class=errorcontent><img src=\"./images/error.gif\" style=\"float:left;margin-
right:10px;\">\n Could not read the file. You need to go through the Metadata
Parser Wizard again.\n</div></div></div>\n\n");

if(!$fp)
{
    echo "<div class=\"errormegabox\"><div class=errorbox>";
    echo "<div class=errortitle>ERROR:</div><div class=errorcontent><img
src=\"./images/error.gif\" style=\"float:left;margin-right:10px;\"> Couldn't
actually open the file you chose. Is it being used by something else right now?
</div>";
    echo "</div></div>";
    exit;
}

while ($data = fread($fp, 4096))
    xml_parse($xml_parser, $data, feof($fp))
        or die(sprintf("<div class=\"errormegabox\"><div class=errorbox><div
class=errortitle>ERROR:</div><div class=errorcontent><img
src=\"./images/error.gif\" style=\"float:left;margin-right:10px;\">XML error: %s
at line %d</div></div></div>",
            xml_error_string(xml_get_error_code($xml_parser)),
            xml_get_current_line_number($xml_parser));

fclose($fp);
xml_parser_free($xml_parser);
?>

```

PARSERS/FGDC_LISTVIEW.PHP

```

<?php
/*
Parser official title: (always on line 4, HTML formatted)
FGDC List View (<em>Experimental</em>)

Author: (always on line 7, HTML tags supported)
Julien McArdle
*/
?>

<?php
/*
Function findbutton lists all the images in a directory (minus their file
extention.) It then uses those images as a the search parameter in the string of
your choice. The function takes in two input variables: the subdirectory you
want to look into, and the string you want to use the search paramater on. The
subdirectory is appended to the base directory of this PHP script, and then into
the images/buttons/ directory.

Once the script succesfully finds a match, it'll return the image pertaining to

```

the succesful search parameter. So for instance, let's take the directory is "geoform", and the string to look is "Vectorial Data." Well in the directory geoform are four images: vec.jpg, raster.jpg, tabular.jpg, and unknown.jpg. The script will find that "vec" is indeed found in the string. and will therefore output vec.jpg. If it can't fnd a match, it returns the text "not found", which the parser then uses accordingly. Sometimes the parser will display the image "unknown.jpg", other times it will proceed to look in another folder.

```

*/
function findbutton($subdir, $stringtosearch) {
    $cwd = getcwd();
    $dir = "$cwd/images/buttons/$subdir";
    $found = "no";

    if (is_dir($dir)) {
        if ($dh = opendir($dir)) {
            while (($file = readdir($dh)) !== false) {
                if ($file != "." && $file != ".." && $file !=
"Thumbs.db") {
                    $snakedfile = substr($file, 0, strpos($file, '.'));
                    if (preg_match("/$snakedfile/i",
$stringtosearch)) {
                        return "<img src=\"$dir/$file\"
class=\"buttonsmall\">";
                        $found = "yes";
                    }
                }
            }
            closedir($dh);
        }

        if ($found == "no") {
            return "nothing found";
        }
    }
}

?>

<?php
/*
The opening information.
*/
$divID = idgen(8);
$divID2 = idgen(8);
$divID3 = idgen(8);
echo "<div class=\"metablock\">\n";
echo "<div class=\"contacttitle\">";
echo "<a
href=\"javascript:HideShow('$divID');HideShow('$divID2');HideShow('$divID3');\">
\n";
echo "<div id=\"$divID2\"><img src=\"./images/collapse.png\" class=\"expand\"></
div>\n";
echo "<div id=\"$divID3\" style=\"display: none\"><img
src=\"./images/expand.png\" class=\"expand\"></div>\n";
echo "</a> Description of FGDC List View</div>\n";

```



```

echo "<div id=\"\$divID\" class=\"descriptioncontent\">\n";
echo "This parser was created for demonstration purposes only. It's intent is
\n";
echo "to demonstrate a more functional application of parsing metadata files.
\n";
echo "In this case, it displays and processes all the metadata files found in
the \n";
echo "same directory as the file you chose. If a picture doesn't load, right
click it, and select\n";
echo "<em>Show Picture</em>.\n";
echo "</div></div>\n\n";

$divID = idgen(8);
$divID2 = idgen(8);
$divID3 = idgen(8);
echo "<div class=\"metablock\">\n";
echo "<div class=\"metatitle\">\n";
echo "<a
href=\"javascript:HideShow('$divID');HideShow('$divID2');HideShow('$divID3');\">
\n";
echo "<div id=\"\$divID2\"><img src=\"./images/collapse.png\" class=\"expand\"></
div>\n";
echo "<div id=\"\$divID3\" style=\"display: none\"><img
src=\"./images/expand.png\" class=\"expand\"></div>\n";
echo "</a>Metadata Files</div>\n";

echo "<div id=\"\$divID\" style=\"text-align: left\">\n";

/*
The actual parser.
*/
$metadata = $metadatafile;

class XMLParser {

    var $insideitem = false;

    function startElement($parser, $tagName, $attrs) {
        if ($this->insideitem) {
            $this->tag = $tagName;
        } elseif ($tagName == "IDINFO") {
            $this->insideitem = true;
        }
    }

    function endElement($parser, $tagName) {
        if ($tagName == "IDINFO") {
            $title = trim($this->title);

            echo "<strong>$title</strong> <br /> $this-
>sername";

            echo "<div class=\"smallbuttonsbox\">";

```

```

        $titledataout = findbutton("data", $this->title);
        if ($titledataout == "nothing found") {
            $seriesdataout = findbutton("data", $this-
>sername);
            if ($seriesdataout == "nothing found") {
                echo "<img src=\"./images/buttons/data/
unknown.jpg\" class=\"buttonsmall\">\n";
            } else {
                echo "$seriesdataout \n";
            }
        } else {
            echo "titledataout \n";
        }
    }

    $geoformout = findbutton(geoform, $this->geoform);
    if ($geoformout == "nothing found") {
        echo "<img
src=\"./images/buttons/geoform/unknown.jpg\" class=\"buttonsmall\">\n";
    } else {
        echo "$geoformout\n";
    }

    $localplaceout = findbutton("place/local", $this-
>placekey);
    if ($localplaceout == "nothing found") {
        $provincialplaceout =
findbutton("place/provincial", $this->placekey);
        if ($provincialplaceout == "nothing found") {
            $nationalplaceout =
findbutton("place/national", $this->placekey);
            if ($nationalplaceout == "nothing
found") {
                echo "<img
src=\"./images/buttons/place/unknown.jpg\" class=\"buttonsmall\">\n";
            } else {
                echo "$nationalplaceout \n";
            }
        } else {
            echo "$provincialplaceout \n";
        }
    } else {
        echo "$localplaceout \n";
    }
}

echo "</div>";

$this->title = $this->sername = $this->geoform = $this-
>placekey = "";
$this->begdate = $this->enddate = $this->cntper = "";
$this->cntorg = $this->address = $this->city = $this->state =
"";
$this->country = $this->cntvoice = $this->cntemail = "";
$this->insideitem = false;
}
}

```

```

function characterData($parser, $data) {
    if ($this->insideitem) {
        switch ($this->tag) {
            case "SERNAME":
                $this->sername .= $data;
                break;
            case "GEOFORM":
                $this->geoform .= $data;
                break;
            case "GEOGUNIT":
                $this->geogunit .= $data;
                break;
            case "TITLE":
                $this->title .= $data;
                break;
            case "PLACEKEY":
                $this->placekey .= $data;
                break;
            case "BEGDATE":
                $this->begdate .= $data;
                break;
            case "ENDDATE":
                $this->enddate .= $data;
                break;
        }
    }
}

}

/*
This code finds out what directory the file the user selected was in. It then
checks that directory for other XML files. If it finds any, it parses them and
calls the code above.
*/
$basedirectory = substr($metadatafile, 0, strrpos($metadatafile, '\\\\'));
$listclass = listitem_grey;

if (is_dir($basedirectory)) {
if ($dh = opendir($basedirectory)) {
    while (($file = readdir($dh)) !== false) {
        $filename = strtolower($file) ;
        $exts = split("/\\.", $filename) ;
        $n = count($exts)-1;
        $exts = $exts[$n];
        $basedirectory = trim($basedirectory);
        $file = trim($file);
        $metadata = "$basedirectory\\\\$filename";
        if (ereg("xml", $metadata)) {

            if ($listclass == "listitem_white") {
                $listclass = "listitem_grey";
            } else {
                $listclass = "listitem_white";
            }
        }
    }
}
}
}

```

```

        echo "<div class=\"\$listclass\">\n";
        $xml_parser = xml_parser_create();
        $metadata_parser = new XMLParser();
        xml_set_object($xml_parser, &$metadata_parser);
        xml_set_element_handler($xml_parser, "startElement",
"endElement");
        xml_set_character_data_handler($xml_parser, "characterData");
        $fp = fopen($metadata, 'r') or die("<div
class=\"errormegabox\"><div class=errorbox>\n<div class=errortitle>ERROR:</div>\n
<div class=errorcontent><img src=\"./images/error.gif\"
style=\"float:left;margin-right:10px;\">\n Could not read the file. You need to
go through the Metadata Parser Wizard again.\n</div></div></div>\n\n");

        if(!$fp) {
            echo "<div class=\"errormegabox\"><div class=errorbox>";
            echo "<div class=errortitle>ERROR:</div><div
class=errorcontent><img src=\"./images/error.gif\" style=\"float:left;margin-
right:10px;\"> Couldn't actually open the file you chose. Is it being used by
something else right now?</div>";
            echo "</div></div>";
            exit;
        }

        while ($data = fread($fp, 4096))
            xml_parse($xml_parser, $data, feof($fp))
                or die(sprintf("<div class=\"errormegabox\"><div
class=errorbox><div class=errortitle>ERROR:</div><div class=errorcontent><img
src=\"./images/error.gif\" style=\"float:left;margin-right:10px;\">XML error: %s
at line %d</div></div></div>",

            xml_error_string(xml_get_error_code($xml_parser)),
                        xml_get_current_line_number($xml_parser));

        fclose($fp);
        xml_parser_free($xml_parser);
        echo "\n</div>\n\n";
    }
}
closedir($dh);
}
}
?>

```

SCRIPTS/FINDBUTTON.PHP

```

<?php
/*
Function findbutton lists all the images in a directory (minus their file
extention.) It then uses those images as a the search parameter in the string of
your choice. The function takes in two input variables: the subdirectory you
want to look into, and the string you want to use the search paramater on. The
subdirectory is appended to the base directory of this PHP script, and then into
the images/buttons/ directory.

```

Once the script succesfully finds a match, it'll return the image pertaining to

the succesful search parameter. So for instance, let's take the directory is "geoform", and the string to look is "Vectorial Data." Well in the directory geoform are four images: vec.jpg, raster.jpg, tabular.jpg, and unknown.jpg. The script will find that "vec" is indeed found in the string. and will therefore output vec.jpg. If it can't find a match, it returns the text "not found", which the parser then uses accordingly. Sometimes the parser will display the image "unknown.jpg", other times it will proceed to look in another folder.

```

*/
function findbutton($subdir, $stringtosearch) {
    $cwd = getcwd();
    $dir = "$cwd/images/buttons/$subdir";
    $found = "no";

    if (is_dir($dir)) {
        if ($dh = opendir($dir)) {
            while (($file = readdir($dh)) !== false) {
                if ($file != "." && $file != ".." && $file !=
"Thumbs.db") {
                    $snakedfile = substr($file, 0, strpos($file, '.'));
                    if (preg_match("/$snakedfile/i",
$stringtosearch)) {
                        return "<img src=\"$dir/$file\"
class=\"button\">";
                        $found = "yes";
                    }
                }
            }
            closedir($dh);
        }

        if ($found == "no") {
            return "nothing found";
        }
    }
}
?>

```

SCRIPTS/IDGEN.PHP

```

<?php
/*
This is the ID Generator code. It'll create a string of random alphanumeric
character of a user-defined length. This code is used extensively in the FGDC
parser as a means to create unique ID values for the DIV tags. Without unique
values, there would be no way to target unique DIVs to expand/collapse upon
request.
*/

function assign_rand_value($num)
{
    // accepts 1 - 36
    switch($num)
    {
        case "1":
            $rand_value = "a";

```

```
break;
case "2":
    $rand_value = "b";
break;
case "3":
    $rand_value = "c";
break;
case "4":
    $rand_value = "d";
break;
case "5":
    $rand_value = "e";
break;
case "6":
    $rand_value = "f";
break;
case "7":
    $rand_value = "g";
break;
case "8":
    $rand_value = "h";
break;
case "9":
    $rand_value = "i";
break;
case "10":
    $rand_value = "j";
break;
case "11":
    $rand_value = "k";
break;
case "12":
    $rand_value = "l";
break;
case "13":
    $rand_value = "m";
break;
case "14":
    $rand_value = "n";
break;
case "15":
    $rand_value = "o";
break;
case "16":
    $rand_value = "p";
break;
case "17":
    $rand_value = "q";
break;
case "18":
    $rand_value = "r";
break;
case "19":
    $rand_value = "s";
break;
case "20":
```

```

    $rand_value = "t";
break;
case "21":
    $rand_value = "u";
break;
case "22":
    $rand_value = "v";
break;
case "23":
    $rand_value = "w";
break;
case "24":
    $rand_value = "x";
break;
case "25":
    $rand_value = "y";
break;
case "26":
    $rand_value = "z";
break;
case "27":
    $rand_value = "0";
break;
case "28":
    $rand_value = "1";
break;
case "29":
    $rand_value = "2";
break;
case "30":
    $rand_value = "3";
break;
case "31":
    $rand_value = "4";
break;
case "32":
    $rand_value = "5";
break;
case "33":
    $rand_value = "6";
break;
case "34":
    $rand_value = "7";
break;
case "35":
    $rand_value = "8";
break;
case "36":
    $rand_value = "9";
break;
}
return $rand_value;
}

function idgen($length)
{

```

```
if($length>0)
{
$rand_id="";
for($i=1; $i<=$length; $i++)
{
mt_srand((double)microtime() * 1000000);
$num = mt_rand(1,36);
$rand_id .= assign_rand_value($num);
}
}
return $rand_id;
}
```

?>