Summer 8-2012

Designing and Developing an Alternative Implementation of the Digital Bathymetric Database, Variable Resolution (DBDB-V)

Donald Lester Brandon Jr.
University of Southern Mississippi

The University of Southern Mississippi

DESIGNING AND DEVELOPING AN ALTERNATIVE IMPLEMENTATION OF THE DIGITAL BATHYMETRIC DATABASE, VARIABLE RESOLUTION (DBDB-V)

by

Donald Lester Brandon, Jr.

A Thesis
Submitted to the Graduate School
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Approved:

[redacted]

Dean of the Graduate School

August 2012

ABSTRACT

DESIGNING AND DEVELOPING AN ALTERNATIVE IMPLEMENTATION OF THE DIGITAL BATHYMETRIC DATABASE, VARIABLE RESOLUTION (DBDB-V)

by Donald Lester Brandon, Jr.

August 2012

This documentation describes a project to design and develop an alternative implementation of the Digital Bathymetric Database Variable Resolution (*DBDB-V*) that will allow efficient ingestion into Geospatial Information System (*GIS*). DBDB-V is a well-known storage facility for wide-reaching *bathymetry.* It is created, maintained, and extensively used by the U.S. Navy, and it is seen in many applications within the public sector. Environmental Systems Research Institute (ESRI) develops a popular suite of GIS applications called *ArcGIS Desktop* that is used worldwide and offered to U.S. Navy programs though the Commercial Joint Mapping Toolkit (*CJMTK*). This thesis will examine the development of an ESRI-compatible spatial geodatabase (GDB) that will hold the DBDB-V data and provide native data ingestion into ESRI products thereby improving the efficiency of how bathymetry is used within the GIS. It will also investigate tools needed to build and update the geodatabase as well as to provide access to the data stored within it. The thesis will also consider test cases to validate the new geodatabase and its tools.

ACKNOWLEDGMENTS

The author of this thesis would like to thank his thesis director, Dr. George Raber, and the other committee members, Dr. Clifton Dixon, Dr. Gregory Carter, and Dr. Brian Bourgeois for their professionalism, expertise, and hindsight offered during the duration of this research effort. Also, he would like to thank Dr. Brian Bourgeois not only for his support of the thesis material but for providing the extra push within the naval community in an attempt to build up interest to back the research. Furthermore, he would again like to thank Dr. George Raber for his endless patience during the progression of this research and the coursework preceding it. Finally, the author would like to thank the administrations at both the University of Southern Mississippi and the Naval Research Laboratory for managing the insurmountable amount of labor-intensive paperwork that kept me confused on a daily basis. Without their support it would have been all the more difficult to achieve this goal.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF ILLUSTRATIONS

Figure

LIST OF CODE EXAMPLES

Code Snippet

CHAPTER I

INTRODUCTION

Statement of Research

The purpose of this thesis is to introduce a new approach to providing Digital Bathymetric Database Variable Resolution (DBDB-V) bathymetry data to Environmental Systems Research Institute (ESRI) based systems. The current implementation of DBDB-V maintains the data using a file format that in earlier versions of ESRI was not readily compatible. The approach mentioned in this research results in a native ESRI database which will allow a smoother interface between the data and the GIS. The data will be quicker to access, manipulate and manage than was previously possible. The effort is intended to support programs in both the private and public sectors that require bathymetric variables for a variety of scenarios. Hydrographic survey planning for navigational aid or mineral research and oceanic modeling are examples of potential research programs. It is relevant to the field of geography because it deals with the spatial mapping of this environment and its variables.

Background Information

DBDB-V

The DBDB-V is a digital database that stores ocean depths and their uncertainties at various grid resolutions as well as in different levels of classification (NIMA 15). Four classification levels exist (level 0-level 3) and each is a separate database with only the first level being publically available. The other three are used primarily in situations deemed classified or secret by U.S. government officials. The DBDB-V is developed and maintained by NAVOCEANO, an organization comprised of military and civilians employees whose primary focus is to provide oceanographic data to the U.S. military branches. NAVOCEANO provides updates to the data on at least an annual basis and is currently releasing version 6.2 of the database to the public. The version being used for this research is version 6.0 which was released in the spring of 2010.

The DBDB-V holds the bathymetric data in spherical coordinates systems and allows extractions to be performed using those systems or planar coordinate systems. The spherical coordinates (geographic or polar stereographic) are measured in arc-minutes and include 2.0, 1.0, 0.5, 0.1, and 0.05 arc-minute measurements while the planar coordinates (Universal Transverse Mercator or Universal Polar Stereographic) are measured in metric resolutions and include 50.0, 20.0, 10.0, 5.0, and 2.0 meter measurements.

The coverage provided by the DBDB-V depends on which resolution of data is selected. The 2 arc-minute resolution grid is the most complete, offering virtual worldwide coverage. The 1 arc-minute grid provides coverage of the Strait of Gibraltar, the Mediterranean Sea, the Red Sea, the Caspian Sea, the Aegean Sea, the Ionian Sea, the Sea of Marmara, the Baltic Sea, the South China Sea, and most of the Pacific Coastline of the United States. The 0.5 arc-minute grid provides scattered coverage along the Pacific coastline and the Atlantic coastline of the United States, the eastern coastline of Greenland, the waters around Iceland, and parts of the North Sea. The 0.1 arc-minute grid covers only a small area around the coastline of the Carolinas and another small area east of Madagascar. Finally, the 0.05 arc-minute grid offers coverage along the coastline of the continental United States, Hawaii, and Puerto Rico. Figure 1 shows the 0.05 arc-minute resolution data coverage on a map.

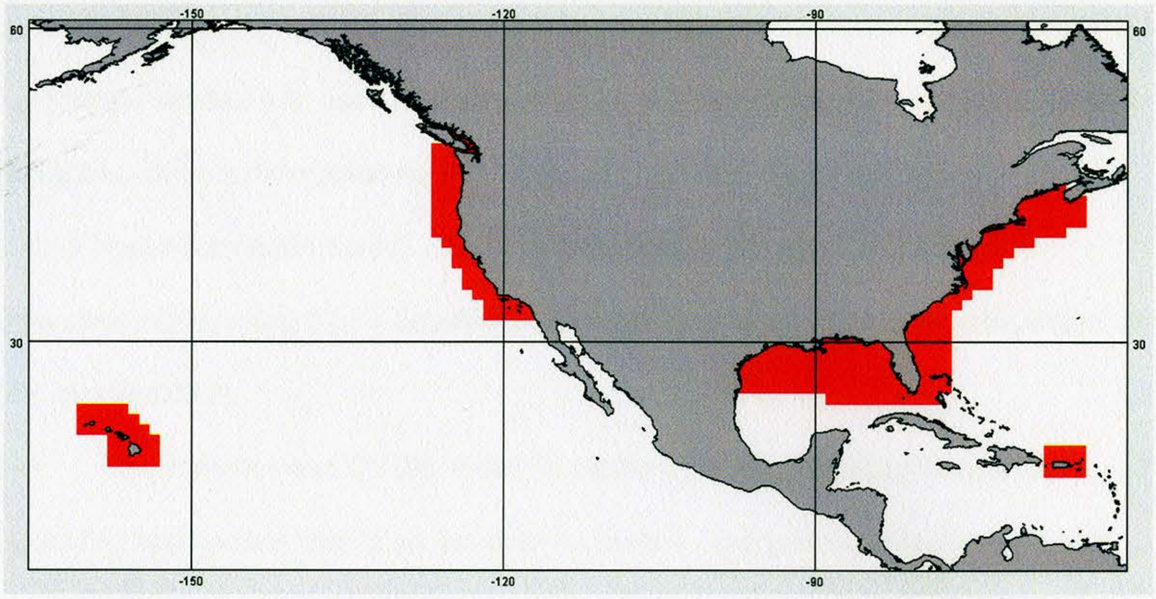


Figure 1. 0.05 arc-minute resolution coverage from DBDB-V, version 6. The image shows the coverage as orange rectangles. The image was generated using the tools provided by NAVO that accompanied the bathymetric dataset.

The DBDB-V is stored using the Hierarchal Data Format, version 5, also referred to as HDF5. HDF5 is a general purpose yet highly efficient storage and management system for scientific data that is capable of handling large quantities of complex data (The HDF Group). It is maintained by the HDF Group, a group of staff and students from the University of Illinois. It is well suited for handling the large DBDB-V dataset however; it does not integrate well with all GIS interfaces. In the case of our GIS application, the tool used to interact with HDF data loads the data into the tool very slow from the large DBDB-V dataset. Then, once in the tool, the data can only be loaded from the extents of the selected tiles. Finally, the rasters that are loaded from the tool have no spatial references defined so they are not georeferenced correctly.

The DBDB-V disc provides a collection of tools that allow access to the bathymetry. These tools are both command-line based, written in ANSII C, and GUI-based, a Java interface to one of the tools, and provide the ability to extract data, generate subset databases, and edit the data in the primary database. The extraction tools interpolate the data into raster datasets using one of three interpolation methods; Nearest Neighbor, Bilinear Interpolation, or a Minimum Curvature Spline feathering algorithm(Lockheed Martin 32). These tools are built upon a public Application Programming Interface (API) that allows you to programmatically access and manipulate the data structure.

The importance of DBDB-V data in mission planning applications and ocean modeling applications should not be under estimated. The gridded datasets are used to model the ocean bottom in applications that generate survey line estimations, and the higher resolution datasets may be used in applications that involve defining topology characteristics such as seamounts or canyons. If mission planning systems are GIS-based, DBDB-V needs to be in a format that can be easily integrated into the application.

ESRI ArcObjects

The GIS application used in this research consists of custom built tools created using ESRI's ArcObjects. ArcObjects are Component Object Model (COM) components that are implemented through a large collection of interfaces. They are an extensive set of over 3,400 classes stored in more than 70 libraries and contain over 21,000 methods and properties that can be used to build elaborate mapping applications (Burke, 9). They are the building blocks of ESRI's sophisticated ArcGIS Desktop suite of applications that includes ArcMap, ArcCatalog, ArcScene, and ArcGlobe.

ArcObjects are responsible not only for the look and feel of an ESRI GIS application, but also for the underlying functionality that controls how the objects are structured, used, and managed. They allow users to perform various levels of analysis on geographical data. The analyses can be spatial, temporal, dimensional, and even based on different types of networks. ArcObjects have classes that will allow a large number of raster and vector data types to be incorporated into a GIS. Examples of raster data types may include, but are not limited to, Tagged Image File Format (.tif), ERDAS Imagine, (.img), and ESRI ASCII Raster (.asc). Vector data types may include shapefiles and coverages. Figure 2 shows an example of ArcMap displaying some of the aforementioned data types.

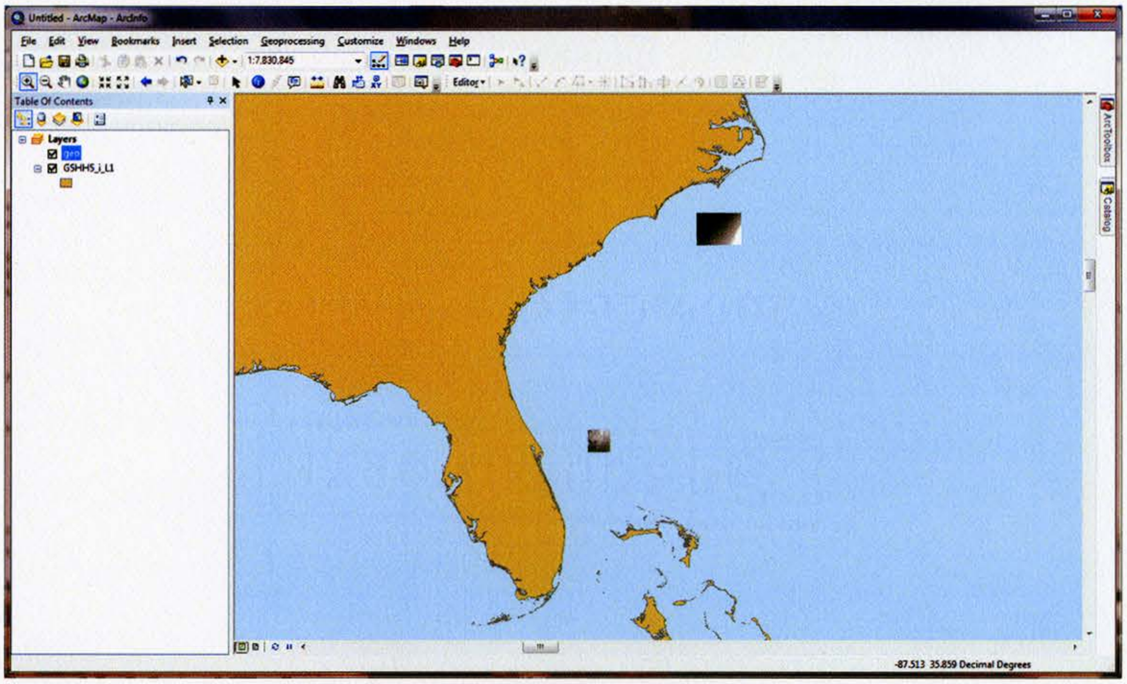

Untitled - ArcMap - ArcInfo
File Edit View Bookmarks Insert Selection Geoprocessing Customize Windows Help
1:7,830,845
Editor
Table Of Contents
Layers
geo
GSHHS_i_L1
ArcToolbox
Catalog
-87.513 35.859 Decimal Degrees

Figure 2. ArcMap Displaying Vector and Raster Datasets. The shapefiles represent the land objects and the raster data represents the bathymetry (the black and white image files).

In addition to the ArcGIS Desktop, ESRI offers another way for users to access the functionality of ArcObjects. The ArcEngine Software Development Kit (SDK) provides the capability to create custom applications that can focus on very specific objectives allowing the application to be streamlined to the needs of the user. This is very useful because full featured GIS applications such as ArcMap are very complex and difficult to learn. They may include many functions that are unnecessary to the purpose of the user and they may conceal the ones that are actually desired. This makes navigating a full featured GIS cumbersome. Creating custom applications can take advantage of the full mapping features of the ArcObjects as well as console applications that only take advantage of the geospatial processing power of the libraries. Using the ArcEngine SDK, it is possible to develop GIS application for a number of different operating systems other than Windows including various flavors of Linux and Solaris.

CHAPTER II

LITERATURE REVIEW

It is difficult to find literature that directly relates to the subject of this thesis work. While many papers identify the use of DBDB-V data in various projects, none were found that specifically discuss the creation of an alternative storage mechanism that is GIS compatible. One paper from the Oceans 1989 Proceedings titled *Proposed Internal Database Structure for Digital Bathymetric Database Production,* authored by Braud, Breckenridge, Current and Landrum, discusses the considerations for storing DBDB data and also the limitations of storage and processing power for the computers of that era. This paper considered the criteria for creating a database for the DBDB data but when considering how far technology has come since the late 1980's, the information is no longer relevant.

A paper titled *Marine Geospatial Ecology Tools: An Integrated Framework for Geoprocessing with ArcGIS, Python, R, MATLAB, and C++* and authored by Roberts, Best, Dunn, Treml and Haplin (2010) discusses a set of tools that were developed for the same reasons that this thesis exists; to overcome the complexities of dealing with HDF formatted data as they relate to interacting with a GIS. However it does not demarcate if this data is version four or version five of the HDF. This would be relevant when regarding these tools for DBDB-V since newer versions of the bathymetry use the HDF5 format. It also does not mention that the tools would work for bathymetry specifically since they were initially designed for coastal ecology and management.

Another paper of interest was one titled *Examples of Carter Corrected DBDB-V Applied to Acoustic Propagation Modeling* authored by Fabre and Fabre. This was an internal report of the Naval Research Laboratory that was released in March of 2008.

This report is interesting because it introduces some of the potential shortcomings of the accuracy of the DBDB-V. It states that DBDB-V uses nominal depths rather than true depths and discusses a methodology to correct the inaccurate depths. It also makes mention of future releases of the DBDB-V, one being the current version used for this research containing true depth as an extraction option.

VGRID: A Generic, Dynamic HDF5 Storage Model for Georeferenced Grid Data is a paper authored in 2002 by Steed, Braud, and Koehler that discusses a "new" technique for the storage and retrieval of bathymetric data. VGRID was considered to be a potentially improved storage facility for the DBDB-V although it was actually designed with another source of bathymetry in mind. One important point drawn from this paper is the similarity that both the VGRID and the DBDB-V are structured from the HDF5 storage model. In fact, the VGRID has some features of the DBDB-V pulled into its architecture such as the same available coordinate systems. The paper is a wealth of information of the organization and functionality of HDF5 and is invaluable to understanding how the DBDB-V is structured.

Another paper, *OAML Feathering Algorithm Overview,* authored in 2003 by Steed and William Rankin, discusses the OAML feathering algorithm. This algorithm was the feathering method that was first included in the DBDB-V to allow data of different resolutions to be combined into a single raster with a limited amount of discontinuity along the edges. The paper gives an in-depth view into the methodology of one of the more important interpolation algorithms within the DBDB-V which will inevitably need to be implemented in this research at some point.

By far, the most useful document regarding the DBDB-V is the specification itself, *Database Design Description for the Digital Bathymetric Database - Variable Resolution (DBDB-V) Version 6.0* (Lockheed Martin 2010). The specification gives an in-depth description on topics such as spatial resolutions, coordinates systems, data hierarchy, and input/output formats. It also discusses the underlying API and the tools built on top of it that allow data management, matrix management, and extraction management.

CHAPTER III

METHODOLGY

Plan of Work

Several elements had to be considered when developing a plan of work for this research project. These included the primary audience of the research, the target GIS interface, the SDK used to access the GIS libraries, the storage facility for the new geospatial data objects, and the host operating system for the development environment. Each element was scrutinized thoroughly to ensure that the research would maximize potential usability while providing an efficient interface to a sophisticated collection of bathymetric data.

Table 1

Development Considerations

Development Considerations				
Audience	GIS Interface	Development Language	Storage Facilities	Host OS
DOD	ESRI	.NET	ESRI Personal Geodatabase	Windows
Academia	Quantum GIS	Python	ESRI File Geodatabase	Linux
Private Sector		Java	PostGreSQL Database	Solaris
		C++		

When considering the prospective users of the new geodatabase (GDB), care was taken to include all potential user groups. The U.S. Navy produces the original DBDB-V data so it is logical that they are the primary user of the data. The data is used throughout the Navy for a wide variety of missions and by a large number of groups within. NAVOCEANO is considered the key targeted user of the new data and has offered useful suggestions as to how the data should look and feel and how the tools that are going to be developed should interact with it. However, with the public availability of the DBDB-V and a rapidly expanding knowledge of its existence, other communities have been quick to begin utilizing its vast amount of data in a variety of other projects. Academia is one such example where colleges around the globe are beginning to incorporate this data into their research projects. The University of Southern Mississippi's Department of Marine Science has expressed an interest in using this type of dataset to project a better image of what the convergence of littoral zones and shorelines is like along the northern Gulf of Mexico. While the input from the naval community should be considered foremost, it should not be considered exclusive.

ESRI provides a powerful interface to GIS data but with that power comes a seemingly endless number of ways to program a method to perform a single task. More times than not, a programmer's choice of how to accomplish a task is not always the most optimal manner available although this choice is usually not realized without iteration and prototyping. Furthermore, while there are so many ways to program these methods there are also a number of ways for this code to be integrated into the ESRI interface. This includes providing the methods as ArcToolbox tools, ArcGIS Desktop embedded components, or as standalone executables. These options are explored in the following paragraphs.

The most common method of providing ArcToolbox tools to the ArcGIS Desktop interface for programmers is by writing code in the Python programming language.

Python is an interpreted programming language that has grown enormously in popularity over the years. Until recent years ESRI has always leveraged Visual Basics for Applications (VBA) as the scripting language of choice for developing these tools. However, Python's growing following has led to it becoming the evident replacement to VBA in future releases of ArcGIS Desktop and the primary scripting language used in ArcGIS 10. Python provides access to ESRI's geoprocessor tools and while this allows Python to generate very powerful geoprocessing scripts, it limits what can be done with this method since the geoprocessor only supports a portion of what ArcObjects can provide.

Another option for designing the interface to the GIS would be to implement embedded components into the ArcGIS Desktop. Embedded components can take the form of menu items or toolbar items within the Desktop application. They can be coded to utilize the GIS geoprocessor but can also be coded with pure ArcObjects. Coding in ArcObjects has two major advantages over using the geoprocessor. First, using ArcObjects provides much more functionality that is available with the geoprocessor which allows sophisticated tool and application development. Second, code written in ArcObjects is executed much faster than code using the geoprocessor, a fact that is noticed during code execution, particularly during application startup. Using embedded components affords users the full range of the tools available in the Desktop under the stipulations of the installed license and while this may be beneficial in many situations, it may also in some situations be considered overbearing or undesirable because the additional tools can increase the complexity.

Standalone applications that are written using ArcObjects can have the advantage over embedded components by streamlining the developed application. This allows a smaller application with user interfaces that are easier to maneuver and functionality specific to defined tasks. This is advantageous to users who are not familiar with all of the intricacies of the Desktop and want to avoid the learning curve associated with complex software. Also, users may desire an application that is specific to their needs. That is not to say that an application cannot be a full featured GIS mapping environment with virtually all the components of the Desktop suite available. It does mean, however, that it could simply be a console application running in the background churning out processed data to a user. As with embedded components, the most optimal manner of development is using pure ArcObjects. However it is possible to integrate the ESRI geoprocessor into the code as well. Overall, programming standalone applications using ArcObjects provides the most flexibility when a specific objective is the goal. For the purpose of this research, this was the selected method of development with the realization that a conversion of the code to embedded components may eventually be desired.

The results of this project include very large datasets and how that data is stored affects the efficiency of the tools used for its generation and display. Numerous storage options are available for storing large quantities of geographical data but because ESRI is the foundation for this work, it seemed appropriate to only consider those storage options native to the ESRI environment. There are two practical options to consider: the personal geodatabase and the file geodatabase.

The personal geodatabase is a Microsoft Access database that allows data to be stored both spatially and non-spatially. The data can be queried or managed in ways typical with the Access database, allowing concise searches to be performed. There is a 2 gigabyte size limit of the personal geodatabase due to the properties of the Access database so with very large projects that can potentially consume a large amount of data its practicality can come into question. Also, it only allows one user at a time to access and edit its data. The personal geodatabase can store many useful forms of geospatial data including feature classes, mosaic rasters, raster catalogs, and non-spatial tables. These characteristics make the personal geodatabase a viable option for this research project.

The file geodatabase is actually a system of binary files stored in a collection of folders the allow ESRI software to store, manage, and manipulate spatial and non-spatial data. Compared to the size limitations of the personal geodatabase the file geodatabase seems almost boundless. It gives the user a default 1 Terabyte (TB) size limit per dataset that can be configured to 256 TB. Like the personal geodatabase the file geodatabase can ingest a large number of geospatial datasets as well as non-spatial tables. Unlike the personal geodatabase, it allows multiple users to access the data although only one user can edit the data at one time.

Two of the questions that always seem to arise in the scientific community are which platform the development of the software should take place on and which operating system would be most beneficial to the user. In many cases the decision to use a specific platform for development may directly affect the platform required by the user and vice-versa. They are not always difficult decisions to make but should always be considered, and this research effort was no different.

Two essential factors involved in the selection of a development platform include the desired development language and the software libraries required to construct the applications. There are many development languages that are capable of producing cross platform applications, or those applications that can be run on multiple operating systems. Such languages include C++, Java, Python, Perl, and Tcl/Tk. Other languages, such as those that are based on the Microsoft .Net environment, are targeted only at the Windows environment. These include Visual Basic, C#, F#, and IronPython.

Another important consideration is which software libraries are either required or desired to develop the applications. There are many different APIs that are available to develop GIS applications and they differ in many ways. For example, ArcEngine is the interface to ArcObjects, ERSI's extensive library for developing all of their GIS applications. It provides components that allow a developer to create mapping applications of various complexities and also allows efficient processing of many types of geospatial data. However, in order to access the capabilities available through ArcEngine a developer will pay a substantial amount of money. In contrast, an API is available at no cost that will allow a developer to access the objects that are responsible for the structure of Quantum GIS (QGIS). QGIS is a full-featured GIS application that provides a user many of the same features that are available in ArcGIS Desktop. The API is developed using the C++ QT library which makes it capable of being used for cross platform development.

All of the previously mentioned factors were considered for this research project. The decision as to which of these factors would be incorporated into the project was greatly influenced by who was projected to be the primary user of this software.

NAVOCEANO is obviously that user since they are the most substantial user of the original DBDB-V data. While NAVOCEANO does not depend solely on a single operating system, Windows platforms have consistently been used throughout the organization for many projects. That, factored in with the extensive use of ESRI systems within NAVOCEANO, made it ideal for this research to focus effort towards a Windows-based ESRI GIS application for not only the manipulation and display of the DBDB-V data but also the storage of the converted data. Because of the flexibility of the file geodatabase and the fact that it allows such large datasets, it was the logical choice for the storage solution. The table below shows the considerations that were ultimately selected based on the criteria determined in the previous paragraphs.

Table 2

Selected Development Considerations

Development Considerations				
Audience	GIS Interface	Development Language	Storage Facilities	Host OS
DOD	ESRI	.NET	ESRI Personal Geodatabase	Windows
Academia	Quantum GIS	Python	ESRI File Geodatabase	Linux
Private Sector		Java	PostGreSQL Database	Solaris
		C++		

Several tasks were defined that influenced how this project would be approached from a development standpoint. First, a valid geodatabase schema had to be developed that would ensure that the data would be organized in such a manner that a smooth transition could exist from the existing DBDBDV. Second, a geodatabase would need to be created that supported that geodatabase schema and allows the existing DBDB-V data to be stored within it. Finally, a tool would need to be developed that would allow the data to be accessed and manipulated to derive subsets that could be saved or used for other purposes. These tasks are discussed in greater detail in the next section.

Task Description

Designing the geodatabase schema

The geodatabase schema provides the definition of the geodatabase and describes how the entities within the geodatabase are structured. In a geodatabase, the entities can be spatial (vector entities vs. raster entities) or non-spatial (tables).

Vector entities are those data that are most commonly used to represent geographical features as points, lines, and polygons. The most common example of point data would be places of interest (POI) which represent objects such as gas stations, post offices, and schools. Lines features are used to represent objects such as administrative boundaries, rivers, and roads while polygons are often used to signify national parks, large bodies of land and water, or city parcels. Feature classes and shapefiles are common formats for vector data when working with an ESRI GIS. The raster entities are gridded datasets whose resolution is based on pixel sizes. Rasters generally represent image files but can also correspond to geographical features usually characterized by vector datasets. There are many file formats for rasters. Some of the more popular ones for an ESRI GIS are ASCII gridded rasters, ERDAS Imagine rasters, and GeoTIFF. Standalone tables provide a means to incorporate various non-spatial data into the geodatabase.

The schema also defines the interactions between the entities such as the relationships and cardinality. Relationships define how the different objects in the geodatabase are related to each other. Relationships can be created between spatial elements, between non-spatial entities, and between combinations of the two. They can be based upon spatial characteristics (spatial relationships) or the attributes of the entities (attribute relationships) and depend upon the objects being related. Cardinality determines the number of times objects between the two entities can be related. Relationships can have a one-to-one cardinality, a many-to-many cardinality, or a one-to-many cardinality. The one-to-one specifies that only one object of the origin element can relate to one object of the destination element. The one-to-many (or many-to-one) specifies that an object of one entity can relate to many objects of the other entities. Finally, the many-to-many cardinality stipulates that many objects of the origin entities may relate to many objects of the destination entities. There are other interactions that can be defined by a database schema, but for the scope of this project, they are not relevant.

The schema for this project is straight-forward. It contains three raster catalogs, four standalone tables, with no relationship classes defined. Originally, the concept included a series of relationships between the raster catalogs and the tables; however the idea was abandoned for simplicity in terms of functionality. Instead, the schema relies on a series of joins that connects the raster catalogs to the tables to provide information on coordinate systems, resolution, tile boundaries, and DBDB-V information. These joins will accomplish the same results as creating a collection of relationships with little or no cost while processing. Figure 3 is a representation of the project schema and shows the joins as color-coded lines.

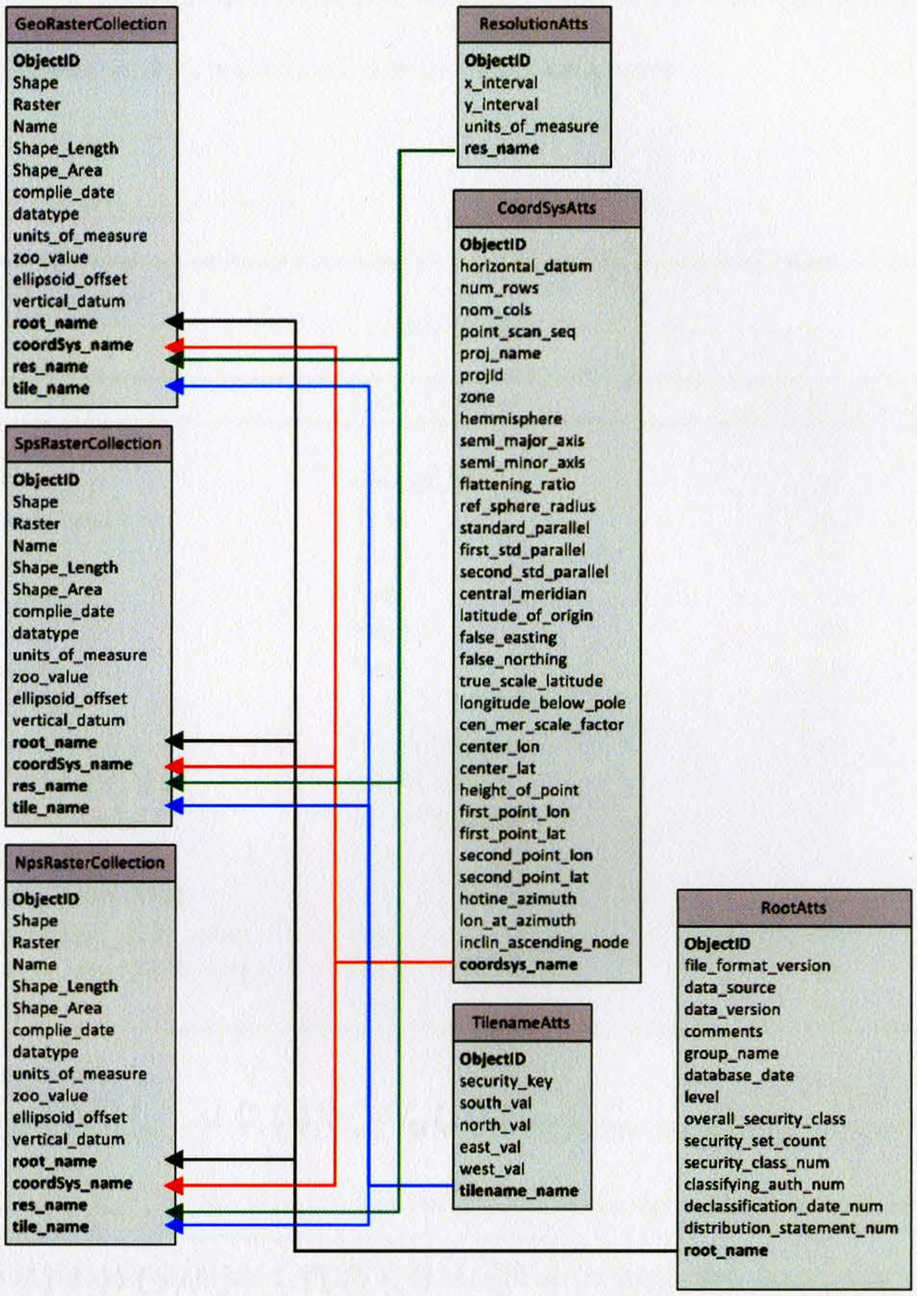


Figure 3. File Geodatabase Schema. The collections on the left represent the rater catalogs that hold the raster information. The collections on the right represent the non-spatial tables holding important attributes. The various colored arrows represent the joins that link the attributes of the tables to the raster collections.

The entities of the database schema relate directly to the information that is available to a user through the original DBDB-V application. The four tables represent the non-spatial attributes that are stored within the DBDB-V and include attributes for root, coordinate system, resolutions, and tile name categories.

Table 3

Root Table Attributes

RootAtts

Attribute Name	Data Type	Join Attribute
ObjectID	Object ID	no
file_format_version	Text	no
data_source	Text	no
data_version	Text	no
comments	Text	no
group_name	Text	no
database_date	Date	no
level	Short Integer	no
overall_security_class	Short Integer	no
security_set_count	Short Integer	no
security_class_num	Text	no
classifying_auth_num	Text	no
declassification_date_num	Text	no
distribution_statement_num	Text	no
root_name	Text	yes

The root table, show in Table 3, provides attributes that defines version information for the DBDB-V data. The attributes outline the version, source, level, and distribution parameters of the DBDB-V data. These are important parameters when working with DBDB-V because not all versions contain the same information and not all levels of DBDB-V are available or approved for all applications. The data_version attribute provides the version of the DBDB-V collection while the level attribute gives the DBDB-V classification level. Level 0 DBDB-V is the only classification that is publicly available. The distribution_statement_100 attribute gives a descriptive message of how the DBDB-V can be used.

Table 4

Coordinate System Attributes

CoordSysAtts

Attribute Name	Data Type	Join Attribute
ObjectID	Object ID	no
Horizontal_datum	Text	no
Rows	Long Integer	no
Columns	Long Integer	no
Point_scan_seq	Long Integer	no
Projection_name	Text	no
Projection_id	Short Integer	no
Zone	Long Integer	no
Hemisphere	Short Integer	no
Lon_Z	Double	no
Lat_Z	Double	no
SemiMajorAxis	Double	no
SemiMinorAxis	Double	no
FlatteningRatio	Double	no
RefSpherradius	Double	no
StandardParallel	Double	no
FisrtStandardParallel	Double	no
SecondStandardParallel	Double	no
CentralMeridian	Double	no
OriginLatitude	Double	no
FalseEasting	Double	no
FalseNorthing	Double	no
TrueScaleLatitude	Double	no
LongitudeBelowPole	Double	no

Table 4 (continued).

CentralMeridianScaleFactor	Double	no
CenterLongitude	Double	no
Centerlatitude	Double	no
HeightToPoint	Double	no
FirstPointLongitude	Double	no
FirstPointLatitude	Double	no
SecondPointLongitude	Double	no
SecondPointLatitude	Double	no
HotineAzimuth	Double	no
LongitudeAtAzimuth	Double	no
InclinationAscendingNode	Double	no
LongitudeAscendingNode	Double	no
RevolutionPeriod	Double	no
LandsatRatio	Double	no
LandsatPathFlag	Double	no
SatelliteNumber	Double	no
PathNumber	Double	no
OblateEqualAreaMParameter	Double	no
OblateEqualAreaNParameter	Double	no
OblateEualAreaRotationAngle	Double	no
coordSysName	Text	yes

The coordinate system table, Table 4, provides attributes that define the different coordinate systems that are available within the DBDBV. There are forty-three attributes within this table that correspond to various parameters used to calculate positions in one of three coordinate systems; geographic, north polar stereographic, and south polar stereographic. Variables such as the semi major axis, semi minor axis, hemisphere, flattening ratio, and false eastings and northings are all defined in this Table. This Table also includes values for the projection name and the horizontal datum.

Table 5

Resolution Attribute Table

ResolutionAtts

Attribute Name	Data Type	Join Attribute
ObjectID	Object ID	no
x_interval	Double	no
y_interval	Double	no
units_of_measure	Text	no
res_name	Text	yes

The resolution table, shown in Table 5, provides attributes that define the resolution of a given tile of DBDBV data. The measure of resolution is stored in two attributes; the x_interval and y_interval. The unit_of_measure attribute determines what those measurements correspond to, i.e. meters, miles or as in most cases of DBDB-V, arc-minutes. An arc-minute of resolution is equal to roughly 1 nautical mile at the equator, or 1,852 meters (1/60 degree).

Table 6

Tile Name Attribute Table

TileNameAtts

Attribute Name	Data Type	Join Attribute
ObjectID	Object ID	no
security_key	Long Integer	no
south	Double	no
north	Double	no

Table 6 (continued).

east	Double	no
west	Double	no
tile_name	Text	yes

The primary purpose of the TileNameAtts table, Table 6 is to provide attributes that describe the boundaries of a tile of DBDB-V data. The Table stores values for the north, south, east, and west boundaries of the tile and the name that corresponds to that tile. A record will be created for every tile of every resolution of every coordinate system of the DBDB-V dataset.

The geospatial elements of the DBDB-V are represented in the schema by three raster catalogs. A raster catalog is a table that stores a raster dataset in each of its records including the raster and geographic information. One convenient characteristic of a raster catalog is its ability to store attributes in addition to the raster datasets and their properties. This allowed a number of important properties to be included in the schema without the necessity for another table. The attributes included the compilation date of the dataset, the units that were used for measuring the dataset, and the vertical datum (i.e. mean sea level) of the dataset. These attributes are in addition to the properties of the raster dataset. The complete listing of the raster catalog attributes, sixteen in all, are listed in Table 7. The attribute table is similar between all three of the raster catalogs in the geodatabase.

Table 7

Geographic Raster Collection Attributes

GeoRasterCollection

Attribute Name	Data Type	Join Attribute
ObjectID	Object ID	no
Shape	Geometry	no
Raster	Raster	no
Name	Text	no
Shape_Length	Double	no
Shape_Area	Double	no
compile_date	Text	no
Datatype	Text	no
units_of_measure	Text	no
zoo_value	Double	no
ellipsoid_offset	Double	no
vertical_datum	Text	no
root_name	Text	yes
coordSys_name	Text	yes
res_name	Text	yes
tile_name	Text	yes

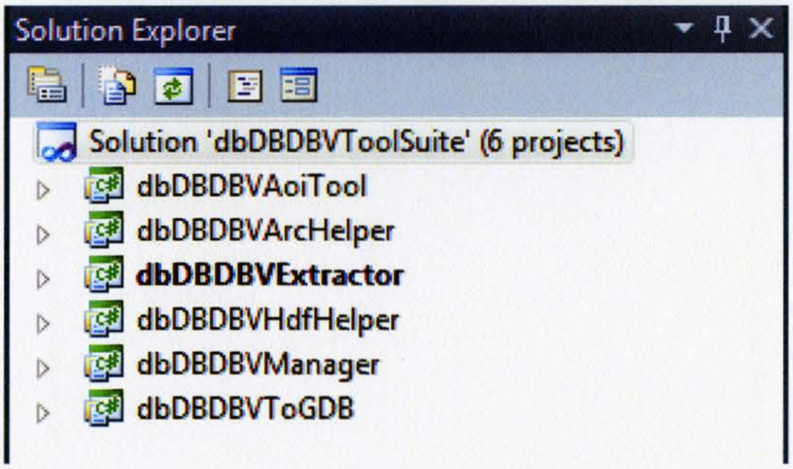
Creating the Geodatabase

Using the schema described above, a geodatabase was developed that would allow the complete storage of the DBDB-V dataset and all of its attributes. The storage facility selected to house all of this information was the file geodatabase. The file geodatabase was selected because it offers an enormous advantage over ESRI's personal geodatabase which utilizes a Microsoft Access approach, limiting it to 2GB size limits. With the file geodatabase there is a 1TB size limit per dataset but is configurable to

256TB. Also, feature storage of the file geodatabase requires one-third of the capacity of the personal geodatabase which can result in significant performance improvements.

The process that generates the file geodatabase based on the schema defined above resides within a Microsoft Windows Graphical User Interface (GUI) application. The application is being developed using Visual C# 2010 Express Edition, a free version of the popular Integrated Development Environment (IDE) released by Microsoft. The IDE provides a programming environment for managing, editing, and debugging the numerous files involved in creating the application.

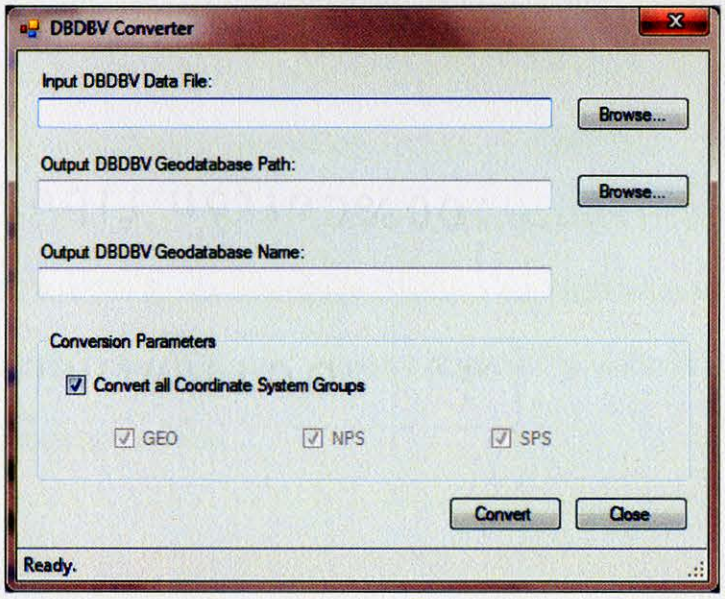
In the .NET programming realm, all of the projects that make up an application are managed within a solution. The solution for this research project consisted of six projects, three of which were responsible for the GUIs, two of which were helper classes, and one of which generated a tool used within one of the GUIs. Figure 4 provides a listing of these projects as seen within the IDE, two which were directly affiliated with the task of creating a geodatabase based on the schema provided.


Solution Explorer

Solution 'dbDBDBVToolSuite' (6 projects)
- dbDBDBVAoiTool
- dbDBDBVArcHelper
- **dbDBDBVExtractor**
- dbDBDBVHdfHelper
- dbDBDBVManager
- dbDBDBVToGDB

Figure 4. Solution Explorer Showing Projects. The projects provide a structure for the source code that makes up the various GUIs and helper methods.

The purpose of the dbDBDBVToGDB project is to provide a user interface to the code that creates the file geodatabase and then loads that geodatabase with data. It presents a simple, easy to use GUI that offers a means to obtain user input that formulates how a new geodatabase will be structured. Parameters on the GUI (Figure 5) determine where the geodatabase will be stored, what name it will have, and what contents it will hold. The input value for the input DBDB-V data file provides the path to the input DBDB-V data. It will be explained further in the next section. The input value for the output GDB path provides the path in which the geodatabase will reside. It does not include the name of the geodatabase which is supplied as input to the next field. This value must be hand-typed. The geodatabase name cannot be achieved through browsing. The remaining dialog objects, the conversion parameters and buttons, will be discussed later. The dialog has a status bar and a progress bar at the bottom which are both used to monitor the data processing.


DBDBV Converter

Input DBDBV Data File:

Output DBDBV Geodatabase Path:

Output DBDBV Geodatabase Name:

Conversion Parameters

[✓] Convert all Coordinate System Groups

[✓] GEO [✓] NPS [✓] SPS

Ready.

Figure 5. DBDBVConverter GUI. The DBDBVConverter GUI application provides the functionality to create the file geodatabase.

The dbDBDBVToGDB project is comprised of three classes; Program.cs, Form1.Designer.cs, and DBDBVConverter.cs. Program.cs is a Windows generated class that is common in GUI applications and has a sole responsibility of configuring the application and running it. Form1.designer.cs is also common to GUI applications generated by Windows but is derived from the layout of the application within the graphical designer. This is the class that gives the application its *look and feel* described above.

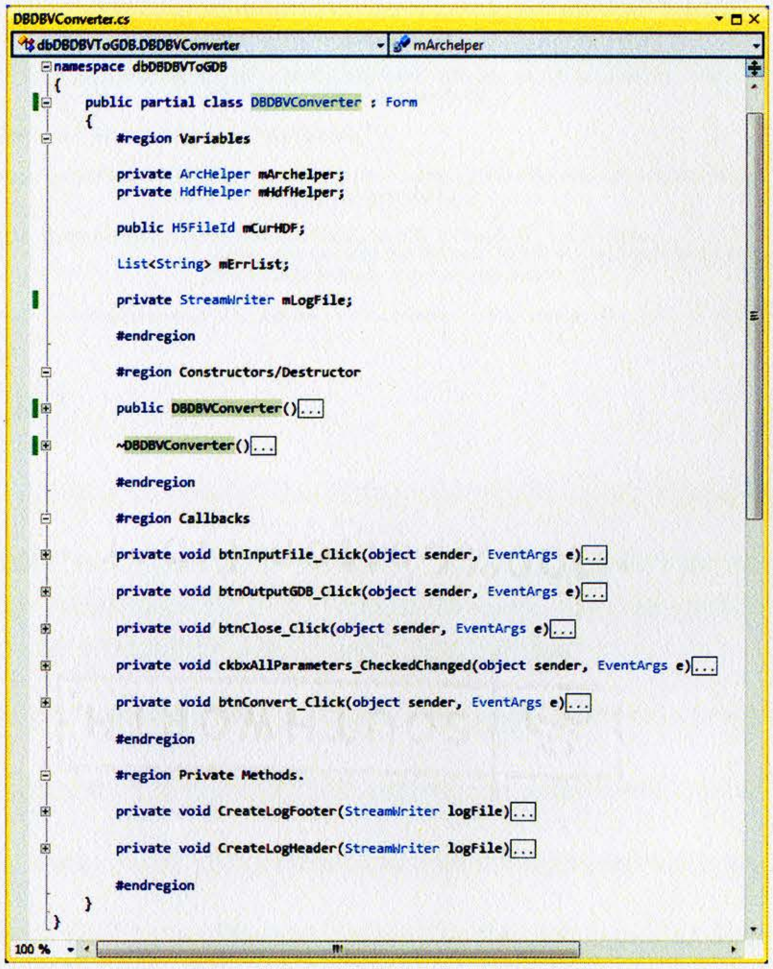
DBDBVConverter.cs is the class that provides the functionality to the application. It is a collection of event callbacks (methods that are linked to user interaction with the GUI) and private methods (methods only visible to the class that defines them) that defines the logic of the program. Methods can also be public methods (methods visible to properly configured calling classes) as will be seen later, however this class contains none. The constructor is also a vital piece to the DBDBVConverter.cs class as well as any other class that contains ESRI components. The constructor is responsible for any initialization that is necessary before the class is instantiated and made available to the user. It is not always required or utilized but because an ESRI application must be bound to a license it is required here. The code below is the constructor for this application. The method ESRI.ArcGIS.RuntimeManager.Bind sets the application to run under a valid ArcEngine or ArcGIS Desktop license. Figure 6 displays the methods and variables used in the DBDBVConverter.cs class.

Code Snippet 1

The DBDBVConverter constructor

```
public DBDBVConverter()
{
  mArchelper = new ArcHelper();
```

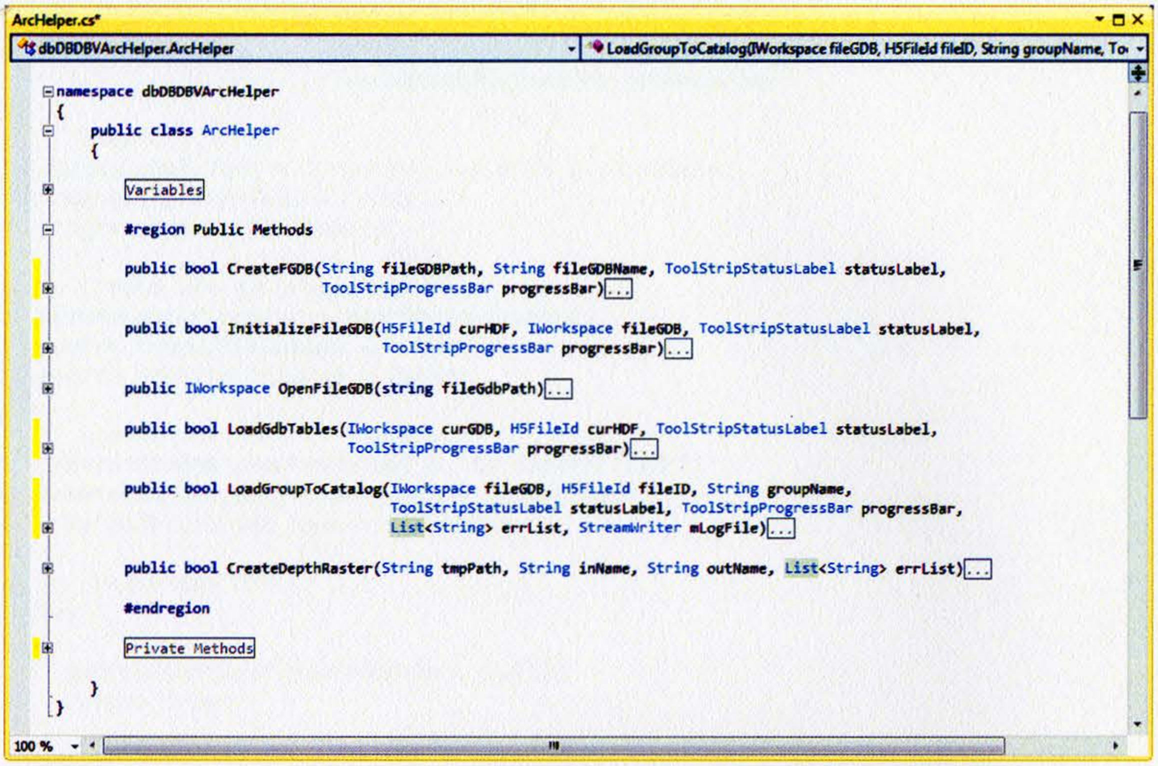
```
    mHdfHelper = new HdfHelper();

    mErrList = new List<String>();

    mLogFile = new StreamWriter(@"C:\Tmp\ConversionLog.log", true);
    CreateLogHeader(mLogFile);
    CreateLogFooter(mLogFile);

    ESRI.ArcGIS.RuntimeManager.Bind(ESRI.ArcGIS.ProductCode.EngineOrDesktop);
    InitializeComponent();
}
```



```
DBDBVConverter.cs
dbDBDBVToGDB.DBDBVConverter        mArchelper
namespace dbDBDBVToGDB
{
    public partial class DBDBVConverter : Form
    {
        #region Variables

        private ArcHelper mArchelper;
        private HdfHelper mHdfHelper;

        public H5FileId mCurHDF;

        List<String> mErrList;

        private StreamWriter mLogFile;

        #endregion

        #region Constructors/Destructor

        public DBDBVConverter()...

        ~DBDBVConverter()...

        #endregion

        #region Callbacks

        private void btnInputFile_Click(object sender, EventArgs e)...

        private void btnOutputGDB_Click(object sender, EventArgs e)...

        private void btnClose_Click(object sender, EventArgs e)...

        private void ckbxAllParameters_CheckedChanged(object sender, EventArgs e)...

        private void btnConvert_Click(object sender, EventArgs e)...

        #endregion

        #region Private Methods.

        private void CreateLogFooter(StreamWriter logFile)...

        private void CreateLogHeader(StreamWriter logFile)...

        #endregion
    }
}
100 %
```

Figure 6. DBDBVConverter Methods. This image shows the methods that are defined in the DBDBVConverter class. The class contains methods that define the class, allow user interaction with the GUI, and provide functionality to the GUI behind the scenes.

Also seen in the constructor is the instantiation of the class ArcHelper. ArcHelper is a custom built class that includes the methods required to create, initialize, and load the geodatabase. Complete public methods and variables defined in this class are seen in Figure 7 while the private methods are seen in Figure 8.


ArcHelper.cs*
dbDBDBVArcHelper.ArcHelper
LoadGroupToCatalog(IWorkspace fileGDB, H5FileId fileID, String groupName, To
100 %

```
namespace dbDBDBVArcHelper
{
    public class ArcHelper
    {

        Variables

        #region Public Methods

        public bool CreateFGDB(String fileGDBPath, String fileGDBName, ToolStripStatusLabel statusLabel,
                               ToolStripProgressBar progressBar)...

        public bool InitializeFileGDB(H5FileId curHDF, IWorkspace fileGDB, ToolStripStatusLabel statusLabel,
                                      ToolStripProgressBar progressBar)...

        public IWorkspace OpenFileGDB(string fileGdbPath)...

        public bool LoadGdbTables(IWorkspace curGDB, H5FileId curHDF, ToolStripStatusLabel statusLabel,
                                  ToolStripProgressBar progressBar)...

        public bool LoadGroupToCatalog(IWorkspace fileGDB, H5FileId fileID, String groupName,
                                       ToolStripStatusLabel statusLabel, ToolStripProgressBar progressBar,
                                       List<String> errList, StreamWriter mLogFile)...

        public bool CreateDepthRaster(String tmpPath, String inName, String outName, List<String> errList)...

        #endregion

        Private Methods

    }
}
```

Figure 7. DBDBVArcHelper Public Methods. This image shows the methods that are considered public within the DBDBVArchelper.cs class. They are responsible for interfacing with the geodatabase and its tables and catalogs.

The first two methods listed in Figure 7 are responsible for initiating the structure of the file geodatabase. The CreateFGDB method is a simple method that uses the ESRI geoprocessor to create a new file geodatabase with the name provided from the DBDBVConverter GUI in the directory defined by the path provided from the GUI. The method is shown in the code below and it shows how the geoprocessor is setup and executed to produce a new file geodatabase. The process is used in several different methods within this research process but is avoided when possible due to the additional overhead involved.

Code Snippet 2

The CreateFGDB method.

```
public bool CreateFGDB(String fileGDBPath, String fileGDBName,
                       ToolStripStatusLabel statusLabel,
                       ToolStripProgressBar progressBar)
{
  // Update status.
  statusLabel.Text = "Creating the file geodatabase...";
  progressBar.Visible = true;
  progressBar.PerformStep();

  // Create the geoprocessor.
  Geoprocessor geoPro = new Geoprocessor();
  geoPro.OverwriteOutput = true;
  geoPro.AddOutputsToMap = false;

  // Create the tool to execute.
  CreateFileGDB createGdbTool = new CreateFileGDB();
  createGdbTool.out_folder_path = fileGDBPath;
  createGdbTool.out_name = fileGDBName;

  // Create the GDB.
  try
  {
    geoPro.Execute(createGdbTool, null);
    return true;
  }
  catch
  {
    object severity = 2;
    MessageBox.Show(geoPro.GetMessages(ref severity), "Error - Creating
                    Geodatabase '" + fileGDBName + "'.",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    return false;
  }
}
```

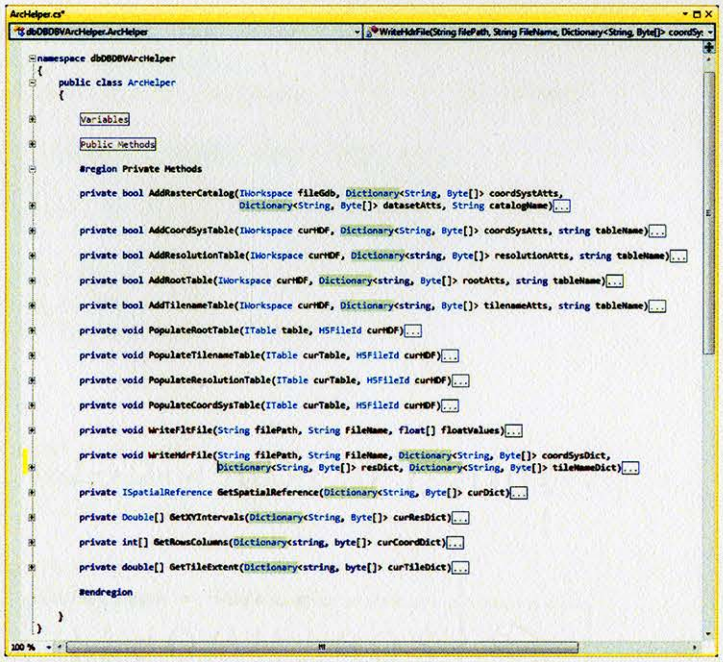
The second method, InitializeFileGDB, is somewhat more sophisticated. First, it utilizes an instance of an object of the class HdfHelper that is defined in the DBDBVHdfHelper project. This class provides an interface to the HDF API that interacts with the DBDB-V to pull attributes and data values from it. The code below shows an example of how the HdfHelper object is used within ArcHelper to get the attributes from the DBDB-V dataset.

Code Snippet 3

The method calls for creating collections of attributes.

```
// Create the dictionaries of attributes.
Dictionary<String, Byte[]> datasetAtts =
            HdfHelper.LoadDatasetDictionary(curHDF,
            "geo/0.05000/214_584/depth");

Dictionary<String, Byte[]> coordSysAtts =
            HdfHelper.LoadCoordSysDictionary(curHDF, "geo");

Dictionary<String, Byte[]> rootAtts = HdfHelper.LoadRootDictionary(curHDF);

Dictionary<String, Byte[]> tileNameAtts =
            HdfHelper.LoadTilesetDictionary(curHDF, "geo/0.05000/214_584");

Dictionary<String, Byte[]> resAtts =
            HdfHelper.LoadResDictionary(curHDF, "geo/0.05000");
```

The DBDB-V contains five levels of attributes as described in an earlier section; the root attributes, the coordinate system attributes, the resolution attributes, the tile name attributes, and the dataset attributes. The code above calls five methods from the HdfHelper object, shown in Figure 9, to build Dictionary objects to store the attributes.


ArcHelper.cs*
dbDBDBVArcHelper.ArcHelper
WriteHdrFile(String filePath, String FileName, Dictionary<String, Byte[]> coordSy:
100 %

```
namespace dbDBDBVArcHelper
{
    public class ArcHelper
    {

        Variables

        Public Methods

        #region Private Methods

        private bool AddRasterCatalog(IWorkspace fileGdb, Dictionary<String, Byte[]> coordSystAtts,
                                      Dictionary<String, Byte[]> datasetAtts, String catalogName)...

        private bool AddCoordSysTable(IWorkspace curHDF, Dictionary<String, Byte[]> coordSysAtts, string tableName)...

        private bool AddResolutionTable(IWorkspace curHDF, Dictionary<string, Byte[]> resolutionAtts, string tableName)...

        private bool AddRootTable(IWorkspace curHDF, Dictionary<string, Byte[]> rootAtts, string tableName)...

        private bool AddTilenameTable(IWorkspace curHDF, Dictionary<string, Byte[]> tilenameAtts, string tableName)...

        private void PopulateRootTable(ITable table, H5FileId curHDF)...

        private void PopulateTilenameTable(ITable curTable, H5FileId curHDF)...

        private void PopulateResolutionTable(ITable curTable, H5FileId curHDF)...

        private void PopulateCoordSysTable(ITable curTable, H5FileId curHDF)...

        private void WriteFltFile(String filePath, String FileName, float[] floatValues)...

        private void WriteHdrFile(String filePath, String FileName, Dictionary<String, Byte[]> coordSysDict,
                                  Dictionary<String, Byte[]> resDict, Dictionary<String, Byte[]> tileNameDict)...

        private ISpatialReference GetSpatialReference(Dictionary<String, Byte[]> curDict)...

        private Double[] GetXYIntervals(Dictionary<String, Byte[]> curResDict)...

        private int[] GetRowsColumns(Dictionary<string, byte[]> curCoordDict)...

        private double[] GetTileExtent(Dictionary<string, byte[]> curTileDict)...

        #endregion

    }
}
```

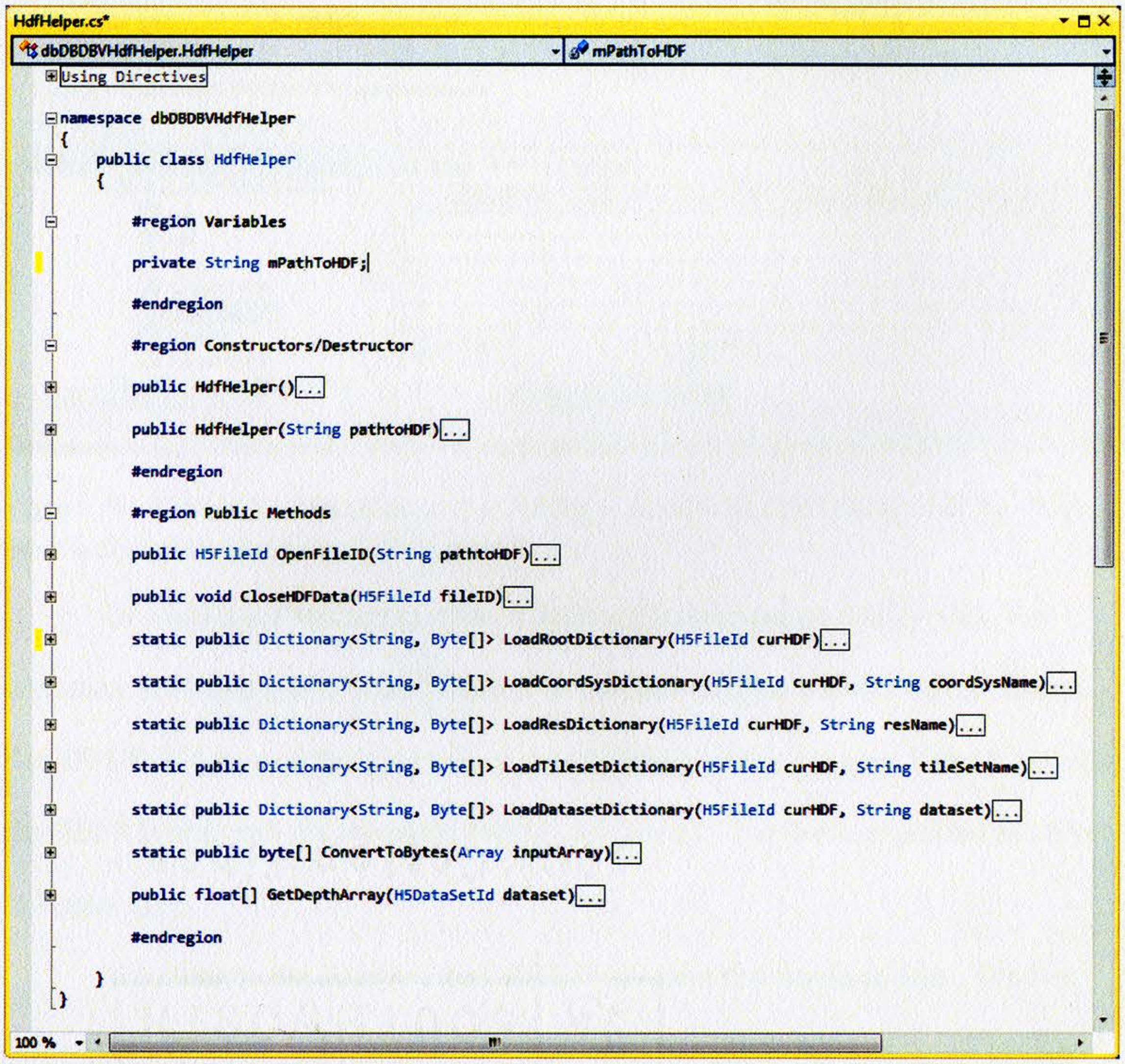
Figure 8. DBDBVArcHelper Private Methods. This image shows the methods that are considered private within the DBDBVArchelper.cs class.

The second thing the InitializeFileGDB method does is call private methods within its class that are responsible for adding the raster catalogs and non-spatial tables into the file geodatabase. These are the first five private methods in the ArcHelper class as shown in figure 8. The process behind these methods is to first create the raster catalog or table in the file geodatabase and then create new fields that correspond to the attributes of that level in the DBDB-V dataset. Programmatically, this is accomplished by reading the attributes from the Dictionary objects that were created in earlier code and using their values to generate fields within the new raster catalog or table. The code for creating the fields in the ResolutionAtts table is a very simple example of this and is shown below.

Code Snippet 4

The code to add fields to a table.

```
// Add fields to the table.
try
{
  //Create a geoprocessing tool to add the fields.
  AddField geoTool2 = new AddField();
  geoTool2.in_table = curHDF.PathName + "\\" + tableName;

  String[] strs = resolutionAtts.Keys.ToArray();

  foreach (String str in strs)
  {
    geoTool2.field_name = str;
    geoTool2.field_alias = str;
    geoTool2.field_is_nullable = "true";

    switch (str)
    {
      case "units_of_measure":
        geoTool2.field_type = "Text";
        geoTool2.field_length = 25;
        break;
      case "x_interval":
        geoTool2.field_type = "Double";
        break;
      case "y_interval":
        geoTool2.field_type = "Double";
        break;
      default:
            break;
        }

         // Execute the tool.
         geoPro.Execute(geoTool2, null);
       }
    }
    catch
    {
        object num = 2;
        MessageBox.Show(geoPro.GetMessages(ref num), "Error - Adding Fields to Table -
                        " + tableName, MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
```



HdfHelper.cs*

dbDBDBVHdfHelper.HdfHelper | mPathToHDF

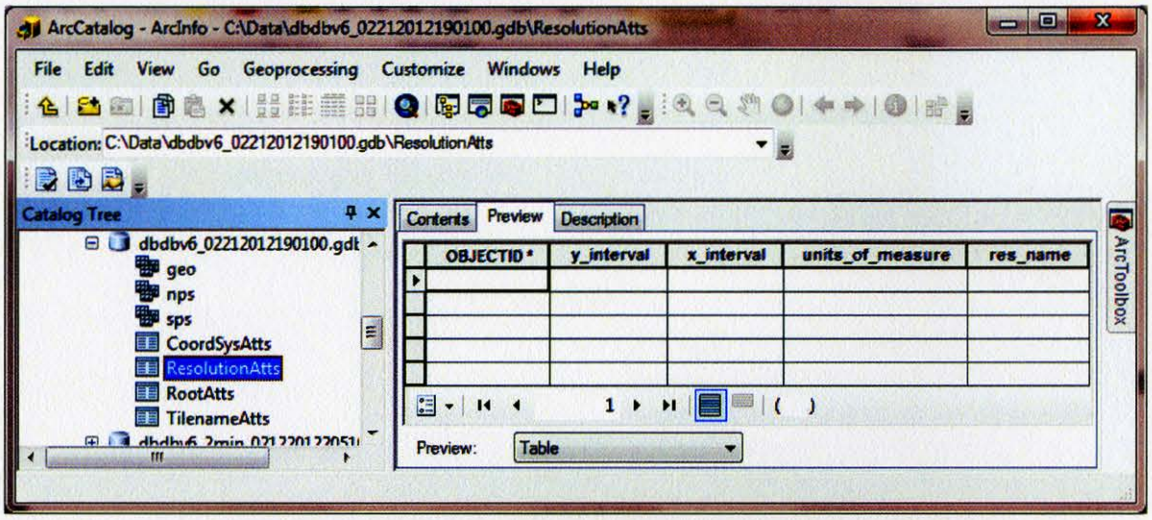
```
[+]Using Directives

namespace dbDBDBVHdfHelper
{
    public class HdfHelper
    {
        #region Variables

        private String mPathToHDF;

        #endregion

        #region Constructors/Destructor

        public HdfHelper()[...]

        public HdfHelper(String pathtoHDF)[...]

        #endregion

        #region Public Methods

        public H5FileId OpenFileID(String pathtoHDF)[...]

        public void CloseHDFData(H5FileId fileID)[...]

        static public Dictionary<String, Byte[]> LoadRootDictionary(H5FileId curHDF)[...]

        static public Dictionary<String, Byte[]> LoadCoordSysDictionary(H5FileId curHDF, String coordSysName)[...]

        static public Dictionary<String, Byte[]> LoadResDictionary(H5FileId curHDF, String resName)[...]

        static public Dictionary<String, Byte[]> LoadTilesetDictionary(H5FileId curHDF, String tileSetName)[...]

        static public Dictionary<String, Byte[]> LoadDatasetDictionary(H5FileId curHDF, String dataset)[...]

        static public byte[] ConvertToBytes(Array inputArray)[...]

        public float[] GetDepthArray(H5DataSetId dataset)[...]

        #endregion

    }
}
```

100 %

Figure 9. DBDBVHdfHelper Methods. These methods provide the functionality to interoperate with the DBDBD-V in its original format.

Loading data into the geodatabase

Once the file geodatabase has been created and it has been loaded with empty raster catalogs and non-spatial tables, viewed with ArcCatalog in figure 10, it needs to be loaded with attribute values and DBDB-V data. From a user standpoint, the processes of creating the geodatabase and filling it with data are seamless. However, from a programming perspective, they are two very different procedures.



Location: C:\Data\dbdbv6_02212012190100.gdb\ResolutionAtts

OBJECTID *	y_interval	x_interval	units_of_measure	res_name

Figure 10. ResolutionAtts table in ArcCatalog. A view in ArcCatalog that shows the table before any records have been added.

To understand how the process of loading the geodatabase had to work, it is important to understand how the DBDB-V is fundamentally stored in the HDF format. The DBDB-V data used for this research was DBDB-V version 6, level 0 and is saved in the HDF5 format with the file name dbdbv6_level0c.h5. The file is organized as shown in Figure 11.

According to the diagram, the DBDB-V contains five levels of data. The first four contain elements that have attributes that must be extracted and stored in the file geodatabase. DBDBV_v6.0_level0 is the root element of the DBDB-V dataset. It contains the attributes listed in Table 1 with the exception of the first and last attribute in the table. The next level in the diagram (geo, nps, and sps) represents the coordinate systems that exist in the DBDB-V dataset. Each of these coordinate systems holds 43 attributes, seen in Table 2, that must be extracted. Next are the resolutions that are labeled as the decimal values in the diagrams. There are three attributes for the resolutions, as listed in Table 3. The fourth level in the DBDB-V dataset represents the names of the tiles that exist for the given coordinate system at the given resolution. The tile name attributes define the boundaries of the ensuing dataset and are shown in Table 4.

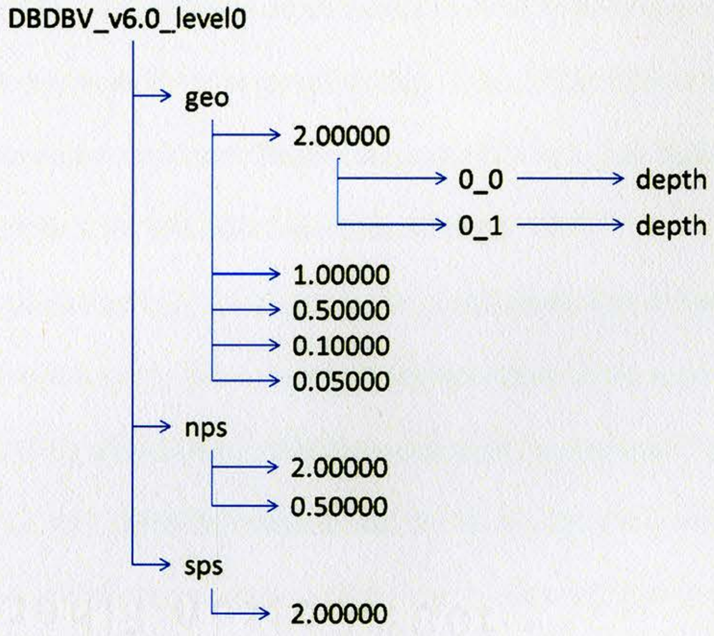


Figure 11. DBDB-V Data Structure. This diagram shows the structure of the data inside the DBDB-V dataset. The structure contains several layers of nested groups with the actual data residing in the "depth" dataset.

The final level introduced in the diagram is special in that it not only contains attributes that must be represented but also contains the data that will be used to build the rasters in the raster catalogs. The attributes are among those listed in Table 7, the Table that is associated with the raster catalogs. The dataset represents a 600 x 600 array of doubles that signifies depths at the geographical locations defined by the collection of attributes exposed through the DBDB-V levels navigated to get to that dataset. Therefore, it was very important that as this extraction was occurring, order was kept and the relationships between the various levels and the dataset were not compromised.

The extraction of the HDF datasets that held the DBDB-V information was an extremely challenging undertaking. The extraction of the HDF datasets required using a third party API called HDF5DotNet that wrapped the API produced by the HDF Group (The HDF Group) to interact with the DBDB-V dataset in its native format. The DBDB-V had to be polled for attributes and depth values in order to fill the geodatabase, and the API provided the functionality to accomplish this. The API is written in C++/CLI which enables it to communicate with any DotNet language (C++, Visual Basic, and C#). The version of the API used for this effort was HDF5DotNet 1.8.7.

The first step taken in populating the file geodatabase was to extract data from the DBDB-V dataset that corresponded to the non-spatial tables in the geodatabase. So for example, the RootAtts table in the geodatabase required the attributes from the root level of the DBDB-V so they had to be extracted and saved into the geodatabase table. The methods used to populate the non-spatial tables can be seen in Figure 8 and are recognized as having a method name starting with the word "Populate".

Populating the tables required a series of loops nested within one another to build the tables with every record possible from the DBDB-V. The ResolutionAtts table provides us with a simple example of how this works. First, there are five possible resolutions of data within DBDB-V; 0.05000 arc-minute, 0.10000 arc-minute, 0.50000 arc-minute, 1.00000 arc-minute, and 2.00000 arc-minute. There are three coordinate systems above that which contain some assemblage of these resolutions. They are the geographic (geo), north polar stereographic (nps), and south polar stereographic (sps) coordinate systems. Only one of these, the geographic coordinate system, contains all of the possible resolutions. Therefore, access is needed to only that group to get all of the information on the available resolutions. According to the diagram shown earlier the coordinate system group contains the resolution group, so a loop would be generated to get each resolution from the geographic coordinate system and the attributes from each resolution would be added as a record to the geodatabase table. The code used to populate the ResolutionAtts table is shown next.

Code Snippet 5

The PopulateResolutionTable Method.

```
private void PopulateResolutionTable(ITable curTable, H5FileId curHDF)
{
  // The geo group has all of the available resolutions.  We only need to use it
  // to populate the GDB table.
  String groupName = "geo";
  H5GroupId geoGroup = H5G.open(curHDF, groupName);

  long numGroups = H5G.getNumObjects(geoGroup);

  // Loop through the available resolutions.
  for (int i = 0; i < numGroups; i++)
  {
    String objName = H5G.getObjectNameByIndex(geoGroup, (ulong)i);

    // Create the dictionary.
    Dictionary<String, Byte[]> curDictionary =
                 HdfHelper.LoadResDictionary(curHDF, groupName + "/" +
                                                                        objName
                                                                        );

    String[] keys = curDictionary.Keys.ToArray();

    ICursor cursor = curTable.Insert(false);
    IRowBuffer row = curTable.CreateRowBuffer();

    try
    {
      foreach (String key in keys)
      {
        // Get the value for the key.
```

```
          Byte[] value = null;
          curDictionary.TryGetValue(key, out value);

          int fieldNum = cursor.FindField(key);

          IField curField = cursor.Fields.get_Field(fieldNum);
          IFieldEdit curFieldedit = (IFieldEdit)curField;

          esriFieldType curType = curField.Type;

          if (curType == esriFieldType.esriFieldTypeDouble)
          {
            row.set_Value(fieldNum, BitConverter.ToDouble(value, 0));
          }
          else if (curType == esriFieldType.esriFieldTypeString)
          {
            row.set_Value(fieldNum, System.Text.Encoding.ASCII.GetString(value));
          }
        }
      }
      catch
      {
        MessageBox.Show("Could not create a table row.", "Error - Creating Table
Rows.",
                                          MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
      }

      // Find the _name field.
      int fieldNum2 = cursor.FindField("res_name");

      row.set_Value(fieldNum2, (object)objName);

      cursor.InsertRow(row);
      cursor.Flush();
    }
}
```

The code relies heavily on standard ArcObjects interfaces to access the table's records and fields to add the attributes. However there are three significant lines of code that need further explanation. First is the line "H5GroupId geoGroup = H5G.open(curHDF, groupName);" that opens the group "geo" from the "curHDF" parameter, which is essentially the DBDB-V dataset. The next line, "long numGroups = H5G.getNumObjects(geoGroup);" gets the number of objects from the newly opened group. The "numGroups" variable is the number of objects that exist in that group and the objects represent the resolutions. The variable drives the loop, and inside the loop there is a call to the method LoadResDictionary, shown below.

Code Snippet 6

The call to the LoadResDictionary method.

```
Dictionary<String, Byte []> curDictionary =
              HdfHelper.LoadResDictionary(curHDF, groupName + "/" + objName);
```

This method, briefly mentioned earlier, is the driving force behind the extraction for the resolution attributes. It tunnels through the original DBDB-V dataset and gets the attribute names and values and stores them in a Dictionary object that contains the name as a string and the value as a Byte array. This Dictionary object is then used to add the values to the respective fields for the current resolution in the loop. The code below is from the LoadResDictionary method.

Code Snippet 7

The LoadResDictionary method.

```
static public Dictionary<String, Byte[]> LoadResDictionary(H5FileId curHDF, String
resName)
{
  Dictionary<String, Byte[]> atts = new Dictionary<string, Byte[]>();

  try
  {
    H5GroupId resGroup = H5G.open(curHDF, resName);
    H5ObjectWithAttributes resObj = (H5ObjectWithAttributes)resGroup;

    int numAtts = H5A.getNumberOfAttributes(resObj);

    H5ObjectInfo resInfo = H5O.getInfo(resObj);

    for (int i = 0; i < resInfo.nAttributes; i++)
    {
      H5AttributeId curAttId = H5A.openByIndex(resObj, ".", H5IndexType.CRT_ORDER,
                                          H5IterationOrder.INCREASING, (ulong)i);
      H5AttributeInfo curAttInfo = H5A.getInfo(curAttId);

      string curAttName = H5A.getName(curAttId);

      H5DataTypeId curAttType = H5A.getType(curAttId);
      H5T.H5TClass curAttClass = H5T.getClass(curAttType);
      H5DataTypeId curAttNativeType = H5T.getNativeType(curAttType,
H5T.Direction.ASCEND);
```

```
        int bufferSize = H5T.getSize(curAttNativeType);

        Array buffer = Array.CreateInstance(Type.GetType("System.Byte"), bufferSize);
        Byte[] bArray = ConvertToBytes(buffer);

        H5A.read(curAttId, curAttNativeType, new H5Array<byte>(bArray));

        atts.Add(curAttName, bArray);
    }

    return atts;
  }
  catch
  {
    return null;
  }
}
```

The process is continued until the loop is complete and the table contains a record for each resolution. The process is similar for the other tables albeit somewhat more complex because additional loops are required to handle not only multiple resolutions in multiple coordinate systems, but also multiple tile names in multiple resolutions and multiple depths in multiple tile names. In other words, there are additional nested loops to consider.

The final step required to populate the file geodatabase is to generate rasters from the DBDB-V depth arrays and store them into the raster catalogs along with their associated attributes. This functionality is outlined within the LoadGroupToCatalog method of the ArcHelper class. The method used to add the attributes is similar to the scenario discussed in the previous section and takes place after the raster data is extracted from the DBDB-V and converted to a raster that can be ingested into the geodatabase.

The process of creating a raster from the DBDB-V dataset is extensive and the steps involved can be more easily explained when following the pseudo code below in Figure 12. The interesting part of the pseudo code is where the process of creating the raster begins, or at the line stating, *Build a float array from the depth dataset*. The steps required to get to the depth data are similar to the ones discussed earlier. The rasters that are stored in the raster catalogs of the file geodatabase are GeoTIFF rasters. In order to construct those rasters from ArcObjects however, the data needs to be stored in something other than an array of values. ArcObjects has a class called FloatToRaster that can create a raster from a binary file of float values as long as there is a header file associated with it. The solution then, was to generate a float array from the values, get the depth values from the float array and store them into a binary stream, and then create a header file from the attributes to supplement the stream file. The code to generate the float array from the DBDB-V dataset depth values is shown below, and the code to generate the binary stream of float values is below that in code snippet 8.

```
Get all raster catalogs from the geodatabase
    While there are raster catalogs in the list
        Get the catalog corresponding to the passed in group name
        Get the coordinate system that matches the catalog
        Get all of the resolutions for that coordinate system
        While there are resolutions in the list
            Get the tile names associated with the current resolution
            Get the depth dataset associated with the tile
            Build a float array from the depth dataset
            Write out a binary flt file from the array
            Create dictionaries from attributes
            Write out an ASCII hdr file from the dictionaries in this path
            Create the depth raster (geotiff) from the flt and hdr files
            Flip the raster over to get the proper raster
            Add the raster to a record in the geodatabase
            Add the attributes to a record in the geodatabase
            Get the next tile in the resolution
        Get the next resolution in the coordinate system
    Get the next raster catalog
    End
```

Figure 12. Raster Generation Flowchart. This flowchart describes the process of generating rasters and adding the information to the geodatabases.

Code Snippet 8

The GetDepthArray method.

```
public float[] GetDepthArray(H5DataSetId dataset)
{
  H5DataTypeId datasetType = H5D.getType(dataset);
  H5T.H5TClass datasetClass = H5T.getClass(datasetType);
  H5DataTypeId datasetNativeType = H5T.getNativeType(datasetType,
H5T.Direction.ASCEND);

  long storageSZ = H5D.getStorageSize(dataset);

  float[] dArray = new float[600 * 600 * 4];

  H5D.read(dataset, datasetNativeType, new H5Array<float>(dArray));

  return dArray;
}

private void WriteFltFile(String filePath, String FileName, float[] floatValues)
{
  if (File.Exists(filePath + "\\" + FileName))
  {
    File.Delete(filePath + "\\" + FileName);
  }

  System.IO.FileStream fileStream = new System.IO.FileStream(filePath + "\\" +
                                                             FileName,
                                                             FileMode.OpenOrCreate
                                                             , FileAccess.Write);

  BinaryWriter binWriter = new BinaryWriter(fileStream);

  for (int l = 0; l < floatValues.Length; l++)
  {
    float val = floatValues[l];
    binWriter.Write(val);
  }

  binWriter.Close();
  fileStream.Close();
}
```

In order to create the header file to accompany the flt file, Dictionary objects had to be created for the coordinate system, the resolution, and the tile name that corresponded to the depth dataset. This is because the attributes from them were required to build the header file. The coordinate system attributes were used to define the spatial reference of the raster as well as the number of rows and columns in the raster. The resolution attributes were used to define the *cell size* of the raster and the tile name attributes were used to define the extents of the raster. An example of a header file is shown below.

```
NCOLS 600
NROWS 600
XLLCORNER 0
YLLCORNER -90
CELLSIZE 0.0333333333333333
NODATA_VALUE 32767
BYTEORDER LSBFIRST
```

Once the two files were generated, the raster was built using the ESRI geoprocessor and the FloatToRaster tool. The results of this, however, were not as they were expected and additional processing had to be performed to get the necessary raster. The problem with the output raster was the values in the cells were read from one corner along the y-axis but added at the opposite corner of the y-axis. To rectify this error another ESRI geoprocessing tool had to be invoked. The Flip tool in ESRI will take a raster as input and flip the north-south values while leaving the east/west cell values intact. Consider the examples in Figure 13. Once the raster was corrected it was added to the geodatabase, followed by its attributes in the same manner described earlier.

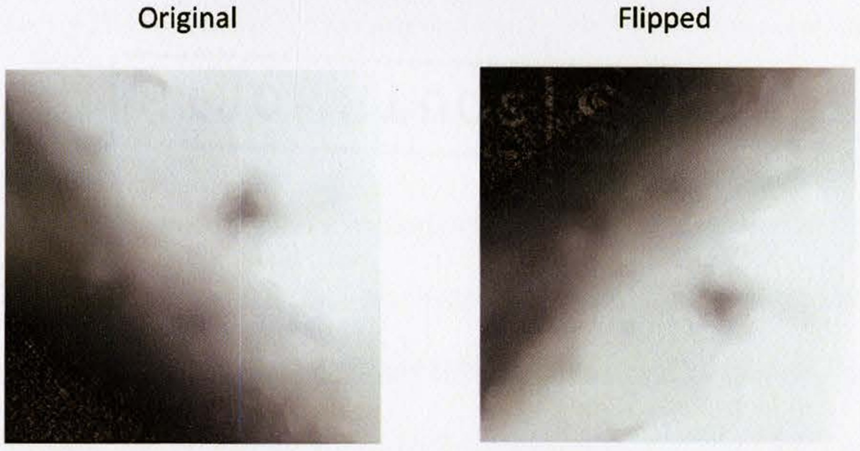


Figure 13. Raster Correction During CreateDepthRaster Method. The original raster generated from the DBDBDV depth dataset (left) and the corrected version (right).

Creating the data extraction tool

The data extraction tool (DBDBVExtractor) was designed to interact with a geodatabase created using the DBDBVConverter tool. The concept is to create a rectangular area, dubbed an area of interest, which will act as an extraction extent and then build a new raster from the data that falls within it. That raster can then be saved in a variety of formats for use in other applications or planning scenarios.

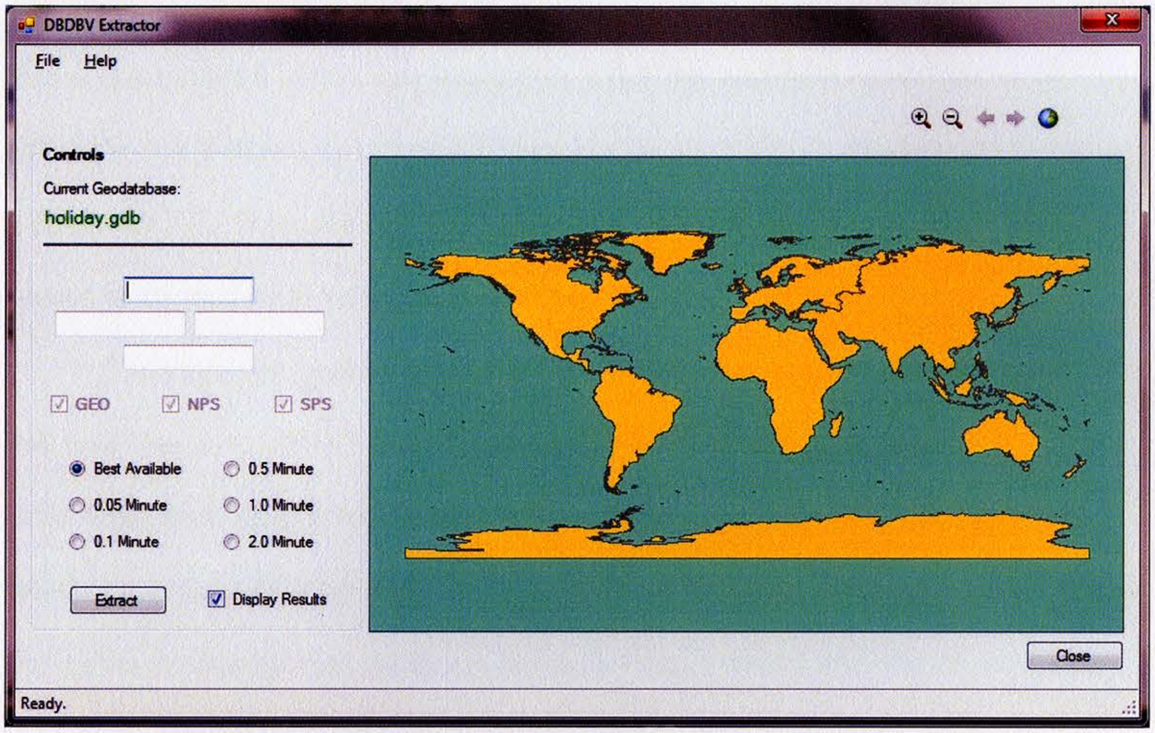


Figure 14. The DBDBVExtractor tool shown with a geodatabase loaded but no areas created.

The DBDBVExtractor tool is a standalone, custom GIS application built using ESRI's ArcObjects. It has a map area where geospatial data can be created and manipulated, and it provides many standard GIS functions such as panning, zooming, and switching between views. The GUI allows an extraction to be created from one of the three coordinate systems. The resolution can be selected from one of the five available resolutions. The "best available" option will provide a raster that has been feathered together from different resolutions as data availability permits, although this option has not been implemented yet.

One of the distinctive features of this tool is the ability to make an extraction based on an area of interest, that is, an extent boundary that defines the area where the data is to be extracted from. The dbAOITool is built as a plug-in to the DBDBDVExtractor application and is visible as a button on the GUI's toolbar. When the button is activated it allows user interaction in the map area to define a rectangular box. When the box is drawn its extents are placed in the application's extent boxes on the left side of the GUI. When the call is made to extract the data the values in these boxes are passed to the method to define the extraction's boundary.

The important method within the tool, where most of the work is performed, is the call back that is invoked when a mouse button is pressed. This method, dbAOITool.OnMouseDown, senses when a mouse button is pressed and determines whether it is the left button or right button. It is not concerned with any other buttons on the mouse in the event that the mouse has more than two. If the left button is pressed, held down, and dragged, a rectangle will be drawn to the screen. When the button is released, the rectangle is finished and the north, south, east, and west extents will be added to the GUI. If the right button is pressed, the rectangle will be cleared from the screen and the extents will be cleared from the GUI. The complete code for the dbAOITool.OnMouseDown is shown below.

Code Snippet 9

The OnMouseDown method.

```
public override void OnMouseDown(int Button, int Shift, int X, int Y)
{
  if (Button == 1)
  {
    // Set the color for the rectangle.
    IRgbColor rgbColor = new RgbColorClass();
    rgbColor.Red = 255;
    IColor rectColor = rgbColor;

    // Set up the symbol.
    ISimpleFillSymbol fillSymbol = new SimpleFillSymbolClass();
    fillSymbol.Outline.Color = rectColor;
    fillSymbol.Outline.Width = 10;
    fillSymbol.Style = esriSimpleFillStyle.esriSFSHollow;

    ISymbol symbol = (ISymbol)fillSymbol;

    // Set up a drawing environment.
    IScreenDisplay display = m_activeView.ScreenDisplay;
    display.StartDrawing(display.hDC, (Int16)esriScreenCache.esriNoScreenCache);

    try
    {
      // Set up the polygon to draw to screen.
      m_aoiGeometry = m_aoiRubberBand.TrackNew(display, symbol);

      display.SetSymbol(symbol);
      display.DrawRectangle(m_aoiGeometry.Envelope);
    }
    catch
    {
      // Catches an error if the user lifts the mouse button without moving it.
    }

    display.FinishDrawing();

    // Update the coordinates.
    extractorForm.UpdateCoordinates(m_aoiGeometry.Envelope.UpperLeft.Y,
                                    m_aoiGeometry.Envelope.LowerLeft.Y,
                                    m_aoiGeometry.Envelope.LowerLeft.X,
                                    m_aoiGeometry.Envelope.LowerRight.X);
  }

   if (Button == 2)
   {
    m_activeView.Refresh();
    extractorForm.ClearCoordinates();
   }
 }
```

The method dbDBDBVExtractor.ExtractByResolution is responsible for the extraction of the DBDB-V data from the raster catalog. The extraction of the data is performed in one of two ways depending on how the area of interest intersects the DBDB-V data to be extracted. First, it is possible the data to be extracted comes from only one raster in the raster catalog. This occurs if the extent rectangle falls completely within a single raster. This is determined by a spatial query that results in a selection set containing all of the rasters that intersect the extent rectangle as shown in Figure 15. If this number is one then the method to perform the extraction is simplified and is performed using the code shown below.

Code Snippet10

Extract the desired raster from a single source raster.

```
// Get the dataset from the current item.
IRasterCatalogItem curCatalogItem = (IRasterCatalogItem)featCursor.NextFeature();
IRasterDataset curRasterDataset = curCatalogItem.RasterDataset;

// Extract a geodataset and set the raster.
IExtractionOp extractOp = new RasterExtractionOpClass();
IGeoDataset outputGeoDataset =
       extractOp.Rectangle((IGeoDataset)curRasterDataset, clipEnvelope, true);

IRaster extractedRaster = (IRaster)outputGeoDataset;
```

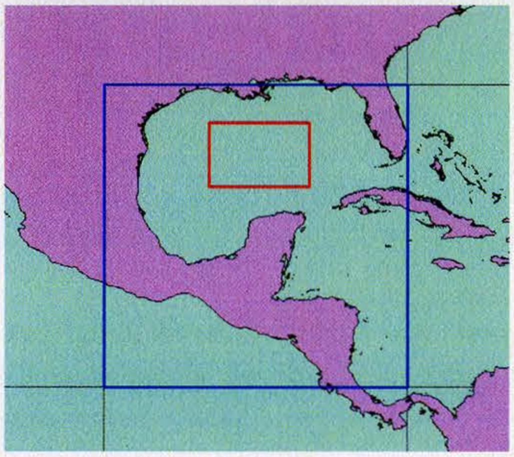


Figure 15. 2 Minute Resolution Selection. This image shows the 2 minute resolution raster (light blue) selected for the extent rectangle (red).

If the spatial query returns a selection set with a count greater than one then more than one raster intersects the extent rectangle, shown in Figure 16, a more sophisticated approach will need to be used to extract the data. This approach requires clipping each individual raster and preserving only the data that falls within the extent rectangle followed by performing a mosaic on the raster data that has been preserved. The clip procedure utilizes an object of type IRasterCollection from ArcObjects to store each of the clipped rasters. The unclipped rasters are accessed through the use of a feature cursor, clipped, and then pushed to the collection.

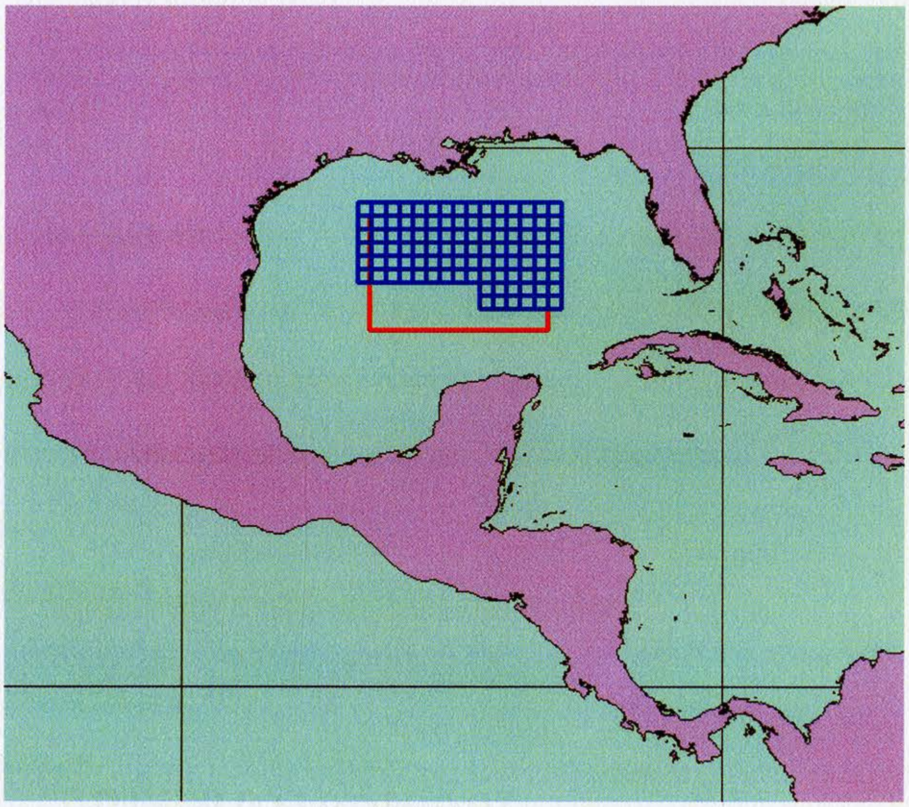


Figure 16. 0.05 Minute Resolution Selection. This image shows the 0.05 minute resolution rasters (light blue) selected for the extent rectangle (red).

Code Snippet 11

Clipping multiple rasters to an extent rectangle.

```
IRasterCollection m_rasColl = (IRasterCollection)new MosaicRasterClass();

// Build the raster collection.
IRasterCatalogItem rasCatItem = (IRasterCatalogItem)featCursor.NextFeature();

while (rasCatItem != null)
{
  IRasterDataset rasDataset = rasCatItem.RasterDataset;

  // Try to clip.
  try
  {
    IRaster tempRaster = rasDataset.CreateDefaultRaster();
    IRasterGeometryProc3 tempRasterGeomProc = new RasterGeometryProcClass();
    tempRasterGeomProc.Clip(clipEnvelope, tempRaster);
    m_rasColl.Append(tempRaster);
  }
  catch
  {
    return null;
  }
}
```

An IMosaicRaster object is used to seamlessly connect all of the rasters in the collection to produce a single raster. The collection created during the clip process is query interfaced to the IMosaicRaster class and is saved as a temporary GeoTIFF file. This provides the raster that is returned as the final extracted raster.

Code Snippet 12

Mosaicking multiple rasters to the desired extracted raster.

```
IMosaicRaster m_mosaic = (IMosaicRaster)m_rasColl;
m_mosaic.WhereClause = "";

ISaveAs m_saveAS = (ISaveAs)m_mosaic;
IRasterDataset rasDataset =
      (IRasterDataset)m_saveAS.SaveAs("tmp_" +
                                      selectedFeatureClass.FeatureClassID.ToStrin
                                      g(),
                                     tmpWS, "TIFF");

mExtractionDataset = rasDataset;

IRaster extractedRaster = rasDataset.CreateDefaultRaster();
```

After the raster is created, regardless of the method used to do so, statistics are calculated so the raster can be properly displayed in the DBDVExtractor map area or any other application that it is going to be viewed in. Calculating the statistics provides the application the minimum, maximum, and median values of the raster as well as its standard deviation. These values make it possible to assign various color ramps to the raster for display. Also, the raster extent is set to that of the extraction extent rectangle that was used to build it so that the map interaction is acceptable. Finally, the NoData value of the raster is set to -10 so that it matches the NoData value used in the original DBDB-V dataset. This value is used in the raster where there is no bathymetry data, for example, over land. This should not be confused with areas in which no data was available from the original data. These values usually are null.

CHAPTER IV

ANALYSIS OF RESULTS

Test Plan Description

In order to validate a file geodatabase that was created using the DBDBVConverter tool and to test the functionality of the DBDBVExtractor tool, a test plan was created that considers the creation of a new geodatabase and the extraction of several rasters from that geodatabase. The test plan calls for the creation of the database and then considers five different areas, each covering a specific resolution of DBDB-V data. The ability of the application to save the extractions and the use of those saved rasters in alternate applications are considered.

Test Plan Results

The first phase of the test plan involved generating a new file geodatabase and populating it with DBDB-V data. The tool used for this process was the DBDBVConverter application, the first of several tools in the DBDBVToolSuite created during this research project. There are three coordinate systems available in the DBDB-V dataset; the geographic coordinate system is used to store values between 80 S and 84 N, the north polar stereographic coordinate system is used to store values north of 84 N, and the south polar stereographic system is used to store the values south of 80 S. The tool is capable of generating a file geodatabase containing any combination of these coordinate systems or a single coordinate system. Regardless of which coordinate system option is selected, the geodatabase that is created will contain a raster catalog for each but only those selected will be populated.

A separate file geodatabase for each coordinate system was created for each test plan and the conversion from HDF to the geodatabase raster catalogs was performed individually. This approach was used because of an apparent processing error when considering values contained in the two polar regions. Creating the three separate geodatabases allowed not only a clean raster catalog of the geographic coordinate system values that could be used for testing but also generated two for the polar regions that could be used for troubleshooting later.

One of the questions proposed for this research was how the storage size of a new geodatabase solution for DBDB-V data would compare to the highly efficient HDF5 data structure currently in use. The DBDB-V used for this project was version 6, level 0 which was released October 2010. The DBDB-V dataset holds a total of 1,707 tiles (600 x 600 arrays) of depth data at various resolutions in the three different coordinate systems. It's on disk storage requirements were approximately 525MB for the data which was stored as a single .h5 file (.h5 being the extension for an HDF version 5 file).

In contrast, the data after the conversion was stored in three separate file geodatabases which if viewed without the help of an ESRI based application appears as typical operating system directories with a .gdb extension. The contents of this directory are a collection of binary files that represent everything inside the GDB including the raster catalogs that hold the DBDB-V depth values. Each of the databases only holds data from the coordinate system they represent. Therefore, the file geodatabase representing the geographical coordinate system contains data for 1,675 of the original 1,707 data tiles, the north polar stereographic geodatabase holds data from 23 of the tiles, and the south polar stereographic geodatabase holds data from the remaining 9 tiles. The total space required from the geodatabase representing the geographic coordinate system is 597MB. The geodatabase for the north polar stereographic data required approximately 9MB while the south polar stereographic geodatabase required approximately 5MB. This resulted in a total storage requirement of 611MB. While this is slightly larger than the HDF format currently used it is more than adequate for housing the new format.

While the format is suitable for storing the DBDB-V data, the process of converting the data from one format to the other is somewhat extensive. The cost in time to perform the conversions for the two polar coordinate systems is negligible. The small amount of data in either of them allows the conversions to be finished in a matter of minutes. However, the large number of tiles to be converted in the geographic coordinate system requires a great deal of processing to be performed that can take upwards of seven hours to complete. While this may seem cumbersome it should be realized that this processing is a single event which is followed by extremely quick extraction methods. These results from these extraction methods are discussed later. The table below shows the processing time required to create and populate each of the three geodatabases.

Table 8

DBDB-V Conversion Time Measurements

Coordinate System	Number of Resolutions	Number of Tiles	Time to Create Tables	Time to Populate Raster Catalogs	Total Time
Geographic	5	1,675	02:30:00	05:14:06	07:44:06
North Polar Stereographic	2	23 - 1	00:03:00	00:01:08	00:04:08
South Polar Stereographic	1	1	00:01:00	00:00:27	00:01:27
Totals		1,706	02:34:00	05:15:41	07:49:41

The file geodatabase for the geographic coordinate system listed above was used to test the capability to extract data from a file geodatabase in order to display it as a raster or to export it to other formats that could potentially be used in other applications. This geodatabase was selected because it contains data from all five of the resolutions available within DBDB-V. The test plan involved setting up one area of interest for each of the resolutions of data. The areas were carefully selected to be sure that both methods of extraction, single-tile and multi-tile, were represented. The DBDBVExtractor tool was used to test the ability of extracting and saving the DBDB-V data from the geodatabase. The tool generates the bounding box extent in values measured in decimal degrees and the values for each area are shown in the Table below.

Table 9

Raster Boundary Values

Resolution	Geographical Location	North Boundary	South Boundary	East Boundary	West Boundary
2 minute	Gulf of Mexico	28.777811	25.0088374	-94.346623	-89.048970
1 minute	Sicily	38.833142	35.354269	11.981383	17.045076
0.5 minute	Irish Sea (Isle of Man)	54.479614	53.932813	-4.892036	-4.259649
0.1 minute	North Carolina Coast	34.818928	34.312543	-76.788847	-76.374787
0.05 minute	Hawaii	20.436332	20.149888	-156.380407	-156.093963
0.05 minute	Hawaii (multiple)	20.995255	18.904326	-157.169682	-154.754298

The 2 minute scenario was set up in the Gulf of Mexico, south of Louisiana and Texas and covering approximately 222,500 sq/km. It is a fairly large area but with a very course dataset the raster extraction process and the display of the results were very quick (within 3 seconds). In comparison to the extraction of the DBDB-V using the existing Java interface the process was equally as quick if not slightly quicker using the DBDBVExtractor tool. Furthermore, the raster that was created using the new tool was identical to the one created using the existing Java interface. The images in the figure below show the comparison.

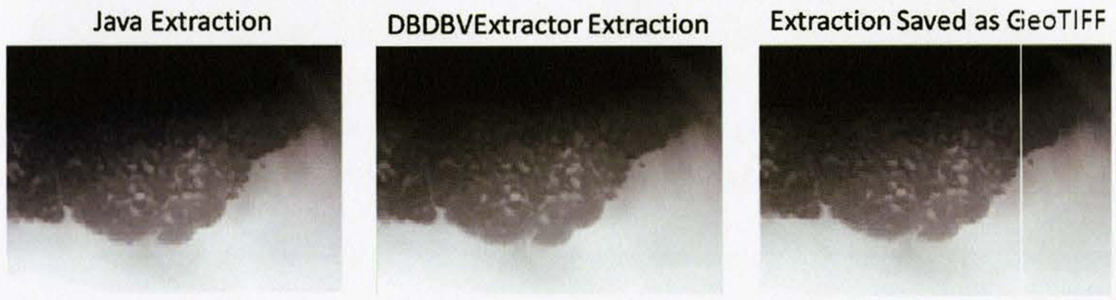


Figure 17. 2 Minute Raster Comparison. A comparison between the Java extraction (left), the DBDBVExtractor extraction (middle), and the result of saving the extraction to another image file (right).

The 1 minute scenario was set up to include the coastal waters around the island of Sicily in the Mediterranean Sea. The area is slightly smaller than the one in the first test case at approximately 173,700 sq/km. The depths range from 2m near the shores, indicated by the darker coverage in the images (figure 18) to 3,780m further from land, indicated by the whiter shades of coverage. As with the first test case, any differences between the Java extraction and the one done with the new tool are unnoticeable. The differences in the near shore are related to the feature classes making up the shorelines being different and not due to differences in the rasters.

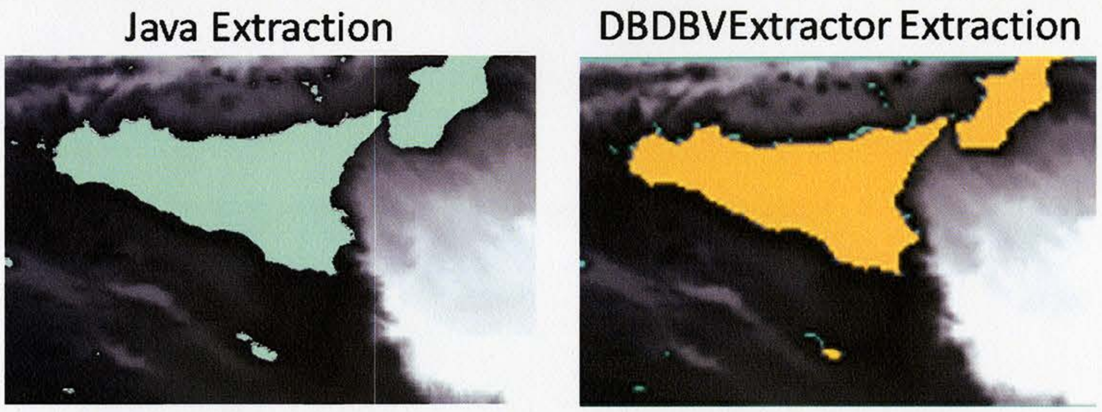
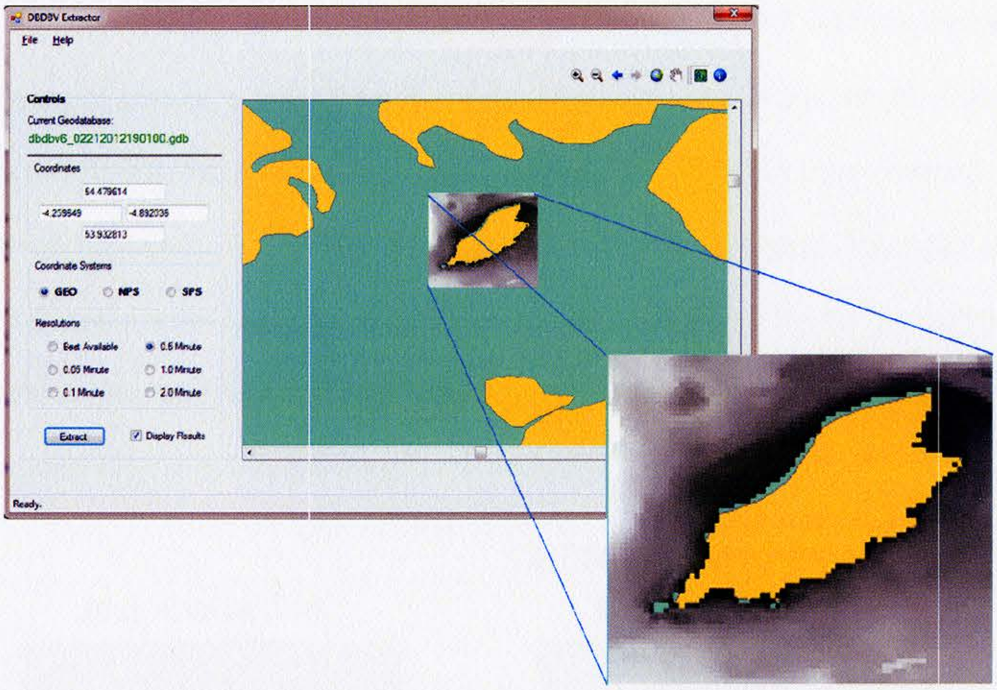


Figure 18. 1 Minute Raster Comparison. This image is a comparison between the Java extraction (left) and the DBDBVExtractor extraction (right).

The test case for the 0.5 minute resolution was created in the Irish Sea and was similar to test case two in that the extraction area surrounded an island, in this case the Isle of Man. The area covered approximately 2,400 sq/km with a cell size of roughly 900m. Figure 19 attempts to communicate the fact that, although much better than the two courser resolutions, 0.5 minute resolution still loses its visual effectiveness as the raster is zoomed. Again, this area was extracted from a single tile of DBDB-V data and the extraction process was exceptionally fast within the DBDBVExtractor tool. On the other hand there seemed to be an increase of time in the Java extraction process even though only one tile was used, along the manner of several seconds. The two extractions were comparable with no known differences noticed.

The area selected for the fourth test case, covering the 0.1 resolution data, was created to cover an area just offshore of the North Carolina coast and the Bogue Sound covering about 2,000 sq/km of mostly water. The 0.1 resolution offers an optimal cell size of approximately 200m. The results for this test case are inconclusive as the

DBDBVExtractor tool reported that no data was available in this area for extraction. An extraction using the Java extraction tool from the original DBDB-V data verified that this assessment was incorrect.



Current Geodatabase: dbdbv6_02212012190100.gdb

Coordinates: 54.479614 / -4.259649 / -4.892036 / 53.932813

Coordinate Systems: (● GEO) ○ NPS ○ SPS

Resolutions: ○ Best Available (● 0.5 Minute) ○ 0.05 Minute ○ 1.0 Minute ○ 0.1 Minute ○ 2.0 Minute

[✓ Display Results]

Figure 19. 0.5 Minute Resolution Extraction. This is a DBDBVExtractor extraction of the 0.5 minute resolution and a zoomed image of the raster. The graininess of the image is due to the cell size of 900m.

Two test cases were employed when considering the 0.05 resolution due to an issue that is present in the DBDBVExtractor when an extraction required data from more than one DBDB-V tile. The issue is related to the display of the extracted data only; the extraction itself results in a legitimate raster. This is demonstrated in the following paragraphs.

The first test case looks at a scenario where an extraction is performed in an area where only one tile is used. This allows a comparison to be made on this resolution similar to those seen in the previous test cases. The first case uses an area that was defined in the cut between the islands of Hawaii and Maui just east of the Ranier

Seamount. Because the tiles holding the DBDB-V get smaller with decreasing changes in the size of the resolution, the area defined here was considerably smaller than the other test cases. The area was around 990 sq/km and the grid had a cell size of approximately 90m. The depths within the area were between 490.6m and 3,175.4 with the deepest water being in a trench coming from the center of the cut between the islands. Figure 20 show a comparison between the Java extraction and the DDBBVExtractor extraction after it was exported to a GeoTIFF file. Looking closely at the bottom left corner, a slight difference is noticed in the images. However, this is due to the two extraction extent rectangles not being precisely matched and not the raster data being off. If the two images were to be overlaid, the images match nicely.

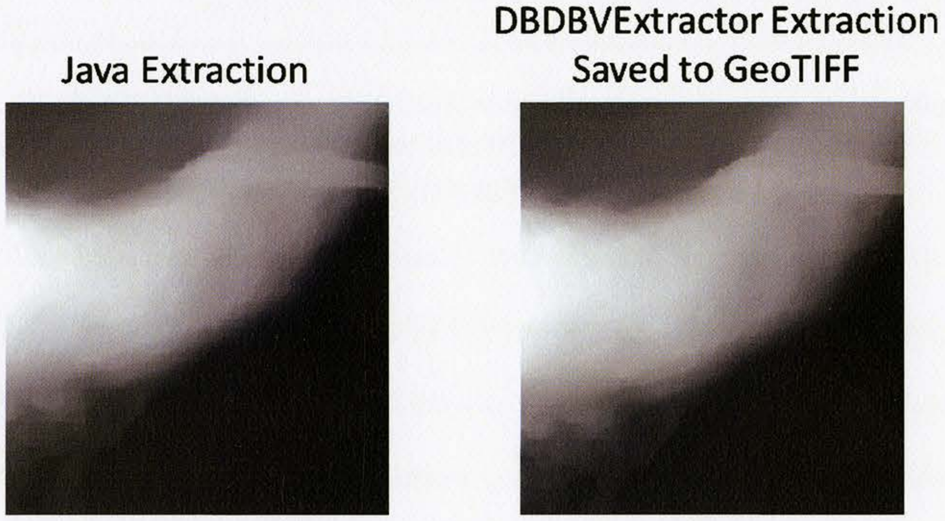


Figure 20. 0.05 Minute Resolution Comparison. This image is a comparison between the Java extraction (left) and the exported DBDBVExtractor extraction (right).

The second test case for the 0.05 minute resolution was created in the same area but used a much larger area than the previous one. The area was about 58,674 sq/km with a cell size approximately 90m. The size of this area required the extraction to be built from 30 DBDB-V data tiles (Figure 21).

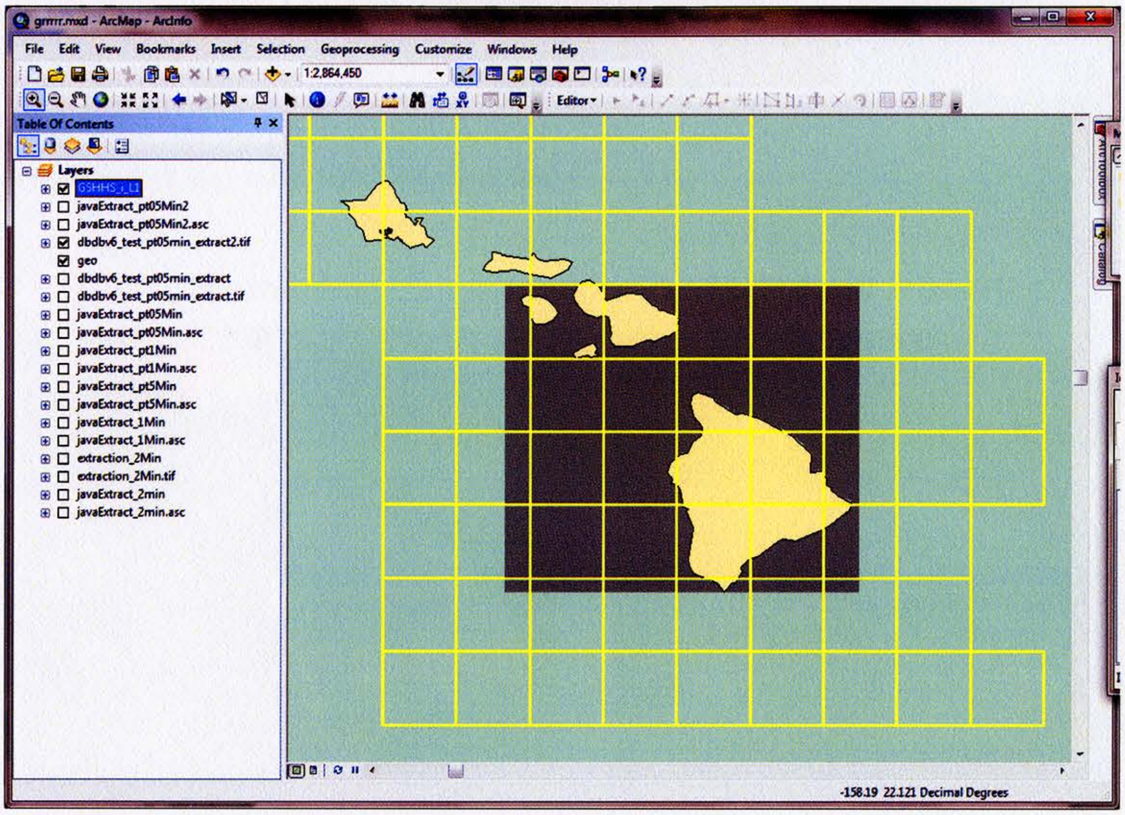


Figure 21. 0.05 Minute Resolution Mosaicking. The figure shows ArcMap displaying the raster created by the DBDBVextractor tool. The tiles used for the extraction are shown in yellow overlapping the grey rectangle.

Using the Java extractor, the process took about 45 seconds to create the raster compared to 15 seconds using the DBDBVExtractor tool. As mentioned earlier, when the DBDBVExtractor tool is executed for a multi-tile extraction the raster that gets created contains the expected data. However, when the raster is displayed within the tool's view, the raster is shown as a grey rectangle rather than as one detailing depth. The figure above contains the grey rectangle which would represent the depth raster. For reasons yet to be determined, the application is displaying a raster that has not had statistics calculated (the min, max, and mean values) which are needed for display. The images in Figure 22 show the two rasters as a side by side comparison after calculating the statistics manually for the raster generated with the new tool. As with other cases, no major differences are noticed between the two rasters.

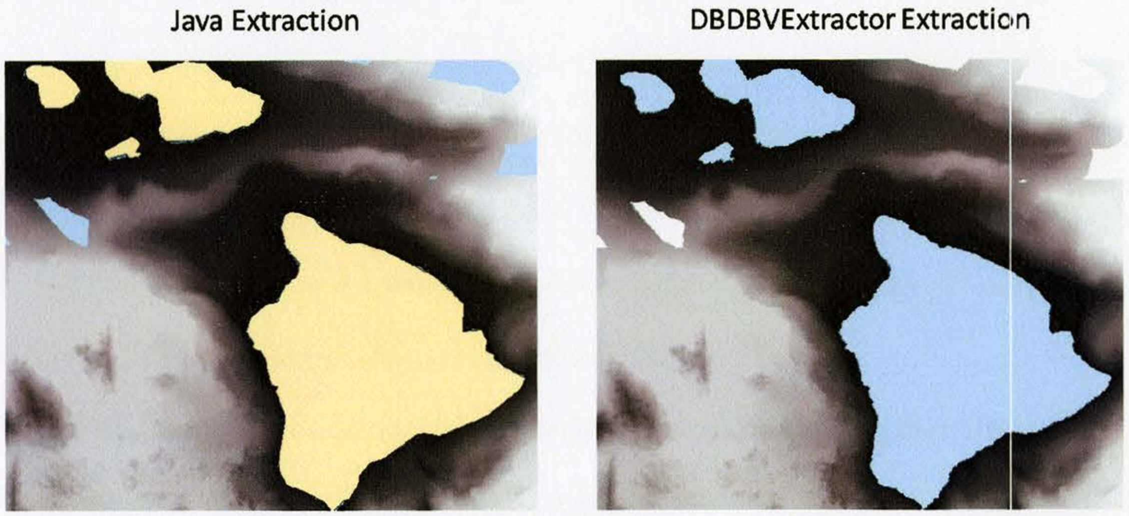


Figure 22. 0.05 Minute Resolution Mosaic Comparison. This image is a comparison of the rasters that were extracted from multiple tiles. DBDBVExtractor is not automatically generating the statistics for the image on the right.

CHAPTER V

CONCLUSION

The purpose of this research was to explore the possibilities of creating an alternative method of integrating DBDB-V data within an ESRI based GIS application. The project looked at the current state of the DBDB-V and how it can currently be ingested into the ESRI systems. A system was designed that would accommodate all of the information stored in the current DBDB-V dataset, both attributes and depth values. A tool was built that would construct and populate that storage facility and another tool was built that would search and pull information from it.

Three questions were considered during this research that would justify it as viable effort to improve the efficiency of ingesting DBDB-V data into ESRI products. First, would the storage of the data in a new format require more space on disk and if so, would the extra space be within the tolerance of the system it was used on and would the additional size degrade functionality? Second, would the process of converting the database be an effort worth undertaking? Third, will the new tools perform efficiently enough to operate in place of the current toolset?

Quite simply, the answers to all of these questions were positive. The file geodatabase format selected to hold the DBDB-V information only increased the required storage space by 86MB, or approximately fourteen percent. Such a small increase would be irrelevant on most systems and would not make any significant difference when searching for or processing information. The process of converting the HDF version of the DBDB-V to the file geodatabase format was somewhat time-extensive but not as much so to warrant abandoning this research effort. Although a full conversion required over seven hours to complete, the conversion could be left to run on its own without user interaction and would only need to be done once with each new release of the DBDB-V. Finally, the DBDBVExtractor tool performed very well when compared to the Java tools used with the HDF version of the DBDB-V. In the several tests that were conducted the new tool performed at or above the level of the Java tools.

The components that have been created for this research comprise a promising implementation of a storage facility and associated tools for the DBDB-V. However, there is still work that could be done that could improve them. For example, when executing the DBDBVConverter tool the conversions for the data within the two polar coordinate systems contain errors and therefore were not considered for this project. Also, there was an issue discovered when mosaicking multiple rasters when executing the DBDBVExtractor tool. When multiple rasters from the file geodatabase were used for an extraction, the display of that extraction was erroneous even though the actual raster data was sound. While this did not affect the results when saving the extraction to file it is a small nuisance when using the tool. Furthermore, two operations originally discussed as major pieces of the overall project design are ongoing. First, a method to provide "data smoothing" needs to be created that will allow depth data from various resolutions of DBDB-V to be joined to provide the best available depth representation possible. This would be similar to the feathering techniques available with the current release of the DBDB-V. Second, consideration should be given to the management of the data once the geodatabase is in place. Updates should be handled by the software rather than needing to recreate a new file geodatabase at each new release of the DBDB-V.

APPENDIX

GLOSSARY OF TERMS

A

Application Programming Interface – "A set of interfaces, methods, protocols, and tools that application developers use to build or customize a software program. APIs make it easier to develop a program by providing building blocks of prewritten, tested, and documented code that are incorporated into the new program. APIs can be built for any programming language." (GIS Glossary).

ArcCatalog – The application within the ArcGIS desktop that deals with GIS data management.

ArcGIS Desktop – A collection of applications developed by ESRI that allow interaction with Geospatial data. These applications include ArcMap, ArcCatalog, ArcGlobe, and ArcScene.

Arc Minute – An angular measurement that is equal to 1/60 of one degree.

Attributes – Characteristics that are associated with geospatial elements of a file geodatabase, identified as the fields of the tables and raster catalogs.

B

Bathymetry – The science of measuring water depths in an effort to determine seafloor topology and/or morphology.

C

Callbacks – Programming methods that provide interaction between the user and the GUI, often providing a means to call desired processing operations.

Cardinality – For database relationships, determines how the objects from various classes are associated with one another.

Cell Size – the size of each cell in a raster.

CHRTR – A NAVOCEANO data format that holds gridded data. The format may be binary or ASCII.

Commercial Joint Mapping Toolkit (CJMTK) – A Department of Defense program between ESRI and NGA TASC that provides a collection of ESRI applications and extensions free of charge to approved federal programs.

Constructor – The method that is called when an object of a class is instantiated.

Cross-platform – In programming, having the ability to utilize a piece of software or a dataset on more than one operating system.

C# Programming Language – A simple, object-oriented programming language that is part of the .NET Framework. It has syntax based on C++ and Java.

D

DBDB-V – Digital Bathymetric Database – Variable Resolution – A bathymetric database that provides water depths at various resolutions.

E

Embedded Components – In ESRI, a tool or command that is embedded into a toolbar that runs inside the ArcGIS Desktop.

F

Feathering – In this application, feathering refers to the smoothing of data along edges where the resolutions of the data are different.

File Geodatabase – an ESRI geodatabase that is stored as a collection of files in a folder.

G

Geoprocessing – “A GIS operation used to manipulate GIS data” (GIS Glossary).

Geospatial Information System – “An integrated collection of computer software and data used to view and manage information about geographic places, analyze spatial relationships, and model spatial processes. A GIS provides a framework for gathering and organizing spatial data and related information so that it can be displayed and analyzed” (GIS Glossary).

GeoTIFF – A TIFF image file that provides geo-referenced information.

Graphical User Interface – The graphical part of an application that allows user interaction for input into an application.

H

Hierarchical Data Format – Represents a data storage facility that can handle very complex datasets and a suite of tools/APIs to handle that data.

I

Integrated Development Environment – An application that provides all of the tools needed to develop applications for a given programming environment such as .NET.

Interpolate – To define new data points within the range of a set of known data points.

Interpretive Programming Language – a language whose programs are translated and run at the same time.

M

Methods – Function calls in a programming language that allows specific processing to take place.

ModelBuilder – "The interface used to build and edit geoprocessing models in ArcGIS" (GIS Glossary).

*Mosia*c – To merge all of the information from several rasters seamlessly, resulting in one final raster.

N

.NET Framework – A Microsoft software framework that is intended to be used for modern Windows programming.

O

Operating System – Software that is responsible for the management and functionality of the hardware inside a computer.

P

Personal Geodatabase – In ESRI, a data storage component that is based on the Microsoft Access database scheme.

Places of Interest – Points on a map that have some geospatial meaning.

Planar Coordinate System – A coordinate system that is projected and drawn on a plane (flat surface).

Private Methods – Methods that are only visible to the class that defines them.

Public Methods – Methods that are visible to properly configured calling classes.

Q

Query Interface – The act of determining if a pointer of a given type can be cast on a specific object.

R

Raster Catalog – A table that contains raster datasets as its records and attributes that are associated with that set of raster datasets.

Read-only – Concerning a user's access to a file or file system, read-only states that a user can only read the given file and does not have permission to write, delete, or execute.

Relationships – Defines how objects in a database schema are related to one another.

S

Schema – A design for databases that defines the entities, their attributes, and the relationships between this entities.

Scripting – Using a set of instructions from an interpreted programming language, such as Python, to run some geoprocessing.

Spatial Query – A query on geospatial information based on a spatial relation between the two objects.

Spherical Coordinate System – A coordinate system based on elevation and azimuth (lat/lon).

V

Visual Basics for Applications – A subset of the Visual Basic programming language that allows programming components inside Windows applications. It was formerly the language of choice for ArcGIS desktop components.

W

Wrapper – A method or set of methods that act as an interface between different programming languages or libraries.

WORK CITED

ArcGIS Desktop 10 | ArcGIS resource Center. Environmental Systems Research Institution. n.d. Web. June 3, 2012.

Braud, Jim, Breckenridge, John, Current, J., and Landrum, Jerry. "Proposed Internal Database Structure for Digital Bathymetric Database Production." *Oceans 1989 Proceedings* 3 (1989): 914–919. Print.

Burke, Rob. "Getting to Know ArcObjects." *Getting to Know ArcObjects free pdf download.* FreeDownload.IS. 2011. Web. June 3, 2012.

Desktop Help 10.0 – Welcome to the ArcGIS Help Library. Environmental Systems Research Institution. May 14, 2012. Web. June 3, 2012.

Fabre, Josie. and Fabre, David. "Examples of Carter Corrected DBDB-V Applied to Acoustic Propagation Modeling." *Naval Research Laboratory Internal Report NRL/MR/7182—08-9100 (2008).* Print.

GIS Glossary – GIS Wiki|The GIS Encyclopedia. November 9, 2010. Web. June 26, 2012.

HDF5 Technologies. The HDF Group. May 16, 2011. Web. June 3, 2012.

Lockheed Martin. "Database Design Description for the Digital bathymetric Database – variable Resolution (DBDB-V) Version 6.0." Stennis Space Center, MS. 2010. Print.

NIMA – MP. "MIL-PRF-32030 - Database Design Description for the Digital Bathymetric Database - Variable Resolution (DBDB-V) Version 6.0." Reston, VA. 2010. Print.

Roberts, Jason J., Best, Ben D., Dunn, Daniel C., Treml, Eric A., and Haplin, Patrick N. "Marine Geospatial Ecology Tools: An Integrated Framework for Geoprocessing with ArcGIS, Python, R, MATLAB, and C++." *Environmental Modeling & Software* 25:10 (2010). 1197-1207. Print.

Sandy, Richard J. "The Navy's Bathymetric Databases… From the Sea." *Sea Technology* November 1996. 53–56. 1996. Print.

Steed, Chad A. Braud, James E., and Koehler, Kim.A. "VGRID: A Generic, Dynamic HDF5 Storage Model for Georeferenced, Grid Data." *Oceans 2002 MTS/IEEE Proceedings* 25 (2002). 900–907. Print.

Steed, Chad A. and Rankin, William E. "OAML Feathering Algorithm Overview." *Formal Report NRL/FR/7440--03-10,052.* U.S. Naval Research Laboratory. 2003. Print.