

Arc Marine as a Spatial Data Infrastructure: A Marine Data Model Case
Study in Whale Tracking by Satellite Telemetry

by
Brett K. Lord-Castillo

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of
Master of Science

Presented November 7, 2007
Commencement June 2008

AN ABSTRACT OF THE THESIS OF

Brett K. Lord-Castillo for the degree of Master of Science in Geography
presented November 7, 2007.

Title: Arc Marine as a Spatial Data Infrastructure: A Marine Data Model
Case Study in Whale Tracking by Satellite Telemetry

Abstract approved:

Dawn J. Wright

The Arc Marine data model is a generalized template to guide the implementation of geographic information systems (GIS) projects in the marine environment. Arc Marine developed out of a collaborative process involving research and industry shareholders in coastal and marine research. This template models and attempts to standardize common marine data types to facilitate data sharing and analytical tool development. The next step in the development of Arc Marine is adaptation to the problems of specific research communities, and specific programs, under the broad umbrella of coastal and marine research by community specific customization of Arc Marine. In this study, Arc Marine was customized from its core model to fit the research goals of the whale satellite telemetry tagging program of the Oregon State University Marine Mammal Institute (MMI). This customization serves as a case study of the ability of Arc Marine to achieve its six primary objectives in the context of the marine animal tracking community. These objectives are: 1) to create a common model for assembling, managing, and publishing tracking data

sets; 2) to produce, share, and exchange these tracking data in a similar format and standard structure; 3) to provide a unified approach for community software developers extending the capabilities of ArcGIS; 4) to extend the power of marine geospatial analysis through a framework for incorporating object-oriented behaviors and for dealing with scale dependencies; 5) to provide a mechanism for the implementation of data content standards; and 6) to aid researchers in a fuller understanding of object-oriented GISs and the power of advanced spatial data structures.

The primary question examined in this thesis is:

How can the Arc Marine data model be customized to best meet the research objectives of the OSU MMI and the marine mammal tracking community, in order to explore the relationship of the distribution and movement of endangered marine mammal species to underlying physical and biological oceanographic processes?

The MMI customization of Arc Marine is focused on the use of Argos satellite telemetry tagging. The customized database schema was described in Universal Markup Language by modification of the core Arc Marine data model in Microsoft Visio 2003 and implemented as an ArcGIS 9.2 geodatabase (personal, file, and ArcSDE). Tool development and scripting were carried out predominantly in Python 2.4.

The two major schema modifications of the MMI customization were the implementation of the Animal and AnimalEvent object classes. The

Animal class is a subclass of Vehicle and models the tagged animal as a tracked instrument platform carrying an array of sensors to measure its environment. The AnimalEvent class represents interactions in time between the Animal and an open-ended range of event types including field observations, tagging, sensor measurements, and satellite geolocating. A programming interface is described for AnimalEvent (AnimalEventUI) and the InstantaneousPoint feature class (InstantaneousPointUI) that represents observed animal locations. Further customization came through the development of a comprehensive development framework for animal tracking in Arc Marine. This framework implements front-end analysis tools through Python scripting, ArcGIS extensions, or standalone applications developed in VB.NET. Back-end database loading is implemented in Python through the ArcGIS geoprocessing object and the DB-API 2.0 database abstraction layer.

Through a description of the multidimensional data cube model of Arc Marine, Arc Marine and the MMI customization are demonstrated to be foundation schemas for a relational database management system (RDBMS), object relational database management system (ORDBMS), or enterprise spatial data warehouse. This modeling method shows that Arc Marine is built upon atomic measures (scalar quantities, vector quantities, points, lines, and polygons) that are described by related dimensional tables (such as time, data parameters, tagged animal, or species) and

concept hierarchies of different levels generalization (for example, tag < animal < social group < population < species). This data cube structure further shows that Arc Marine is an appropriate target schema for the application of on-line analytical processing (OLAP) tools, data mining, and spatial data mining to satellite telemetry tracking datasets.

In this customization case study, Arc Marine partially meets each of its six major goals. In particular, the development of the MMI application development platform demonstrates full implementation of a unified approach for community software developers. Meanwhile, the data cube model of Arc Marine for OLAP demonstrates a successful extension of marine geospatial analysis to deal more effectively with scale dependencies and a mechanism for the expansion of researchers' understanding of high power analytical methods.

©Copyright by Brett K Lord-Castillo
November 7, 2007
All Rights Reserved

Master of Science thesis of Brett K. Lord-Castillo presented on November 7, 2007.

APPROVED:

Major Professor, representing Geography

Chair of the Department of Geosciences

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Brett K. Lord-Castillo, Author

ACKNOWLEDGEMENTS

First, I want to thank my committee members: Dawn Wright, Bruce Mate, Jon Kimerling, and René Reitsma, for their advice, encouragement, and support.

I would also like to thank Dawn Wright for convincing me to come to Oregon State over her doctoral alma mater. Dr. Wright has provided me with a continuous education in GIS beyond all of my classroom experiences. She has been integral to every part of this project, from my first placement with the Mate lab to my conference experiences and early versions of GIS for a Blue Planet to pushing me down the stretch to pull everything together even as I was constantly looking for new directions to my work. And most of all, thank you Dawn for the weekly meetings that provided me with direction, balance, confidence, advice, and a deep understanding of how GIS changes the world.

I want to thank Gordon Matzke and Larry Becker for their insights into this academic discipline we call Geography, and for a ride to San Francisco that was also a trip into West and East Africa. Thank you to Laurie Becker for the opportunity to write the Katrina exercise and all that I learned as your teaching assistant. That experience is one of the key reasons I am in emergency management today. A cheerful thanks to Dr. K, who always made my day brighter even in the middle of an Oregon winter.

Thank you to the staff and faculty of the Oregon State University Marine Mammal Institute for all the insights, advice, specs, guidelines, plans, and encouragement. I am especially grateful to Tomas Follett for his tireless work on the Arc Marine customization, Andy Weiss for his project management expertise as well as his career advice, Joel Ortega for his leadership and knowledge, and Bruce Mate for being the one who made the MMI GIS project happen. I hope you will have continued success and look forward to sharing more Python tools with you in the future.

A scurvy thank you to Celeste Barthel, Jed Roberts, Michelle Kinzel, and the rest of the Rogues of Davy Jones' Locker. Somehow this crazy crew kept me sane as I descended into the abyss of database design. Thanks to Lydia Kamaka'eha Wright, the OSU Geosciences Departmental Dog, for keeping the lab clean of stray snacks.

I want to extend my appreciation to Hamilton Smillie and the NOAA Coastal Service's Center for their support of my work through the Education and Research Opportunities in Applying GIS and Remote Sensing in Coastal Resource Management program (NOAA Grant # NA04NOS4730181). I also want to recognize the family of Arthur Parenzin, providers of the Arthur Parenzin Graduate Fellowship. With their support, I was able to travel to the 2007 ESRI International Conference and present my research. My experiences at that conference provided me new insights that were central to this research work.

Thank you to the students and faculty of the OSU Marine Resource Management Program for providing me a challenging academic home outside the world of GIS and Geography. I want to particularly thank Michael Harte, Richard Hildreth, Robert Allen, Lori Hartline, Chris Pugmeier, Miller Henderson, Daniel Smith, Cathleen Vestfals, Susan Holmes, Topher Holmes, and Mel Agapito for making me feel especially welcome.

Thank you to my family and my in-laws, who have supported Erika and I throughout our time in Oregon. I am especially grateful to my sister, Kendra, and her husband, Clarence, and little Dylan for hosting me in Milpitas while I attended conferences and seminars.

And finally, my eternal gratitude and love to my wife, Erika Lord-Castillo. She let me move her away from the Midwest so she could put up with my stress, exhaustion and late nights as I worked on my classes and my writing. She handled all those little day to day things (which I really need to get better at) so I could stay focused on school, all while trying to manage her own career too. And she even put up with two Oregon winters. Thank you so much Erika, and I look forward to doing the same for you now as you dive deeper into the world of Suzuki violin.

TABLE OF CONTENTS

	<u>Page</u>
Introduction	1
The Arc Marine Initiative.....	4
The Marine Mammal Institute	8
Advantages of Satellite Telemetry.....	11
Research Questions	12
Methods	14
Argos Satellite Telemetry	14
A Brief Tour of Arc Marine	16
Programming languages and software	19
Extending Arc Marine	22
Results	25
Customizing Arc Marine	25
Animal	26
Telemetry	29
Operations	31
Tag.....	33
Filtering	34
Development Framework	36
Interfaces.....	43
Discussion.....	46
Enhancing Satellite Telemetry.....	46
Defining feature behaviors.....	48
Defining information services	52

TABLE OF CONTENTS (Continued)

	<u>Page</u>
Data warehouse.....	53
Data Stream.....	54
Client platform.....	56
Low level access.....	57
The Client Side	58
OLTP and OLAP.....	59
Arc Marine as a data repository.....	62
Conceptual Multidimensional Model of Arc Marine.....	66
The Arc Marine Data Warehouse Design Schema.....	68
The MeasuredData Star	72
The LocationSeries Point Star.....	75
Database Abstraction	76
Conclusion	79
References Cited	86
APPENDICES	93
Appendix A. Arc Marine Data Model Diagrams	94
Appendix B. MMI Customization Data Model Diagram.....	99
Appendix C. GIS Procedures	105
Loading LocationSeries Point from Excel.....	105
Loading InstantaneousPoint from OBIS-SEAMAP	114
Point to Path in Third-Party Extensions.....	120
Using LocationSeries in Model Builder.....	124
Appendix D. Python Codebase	127

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. The solution pair generated by Service Argos geolocation.	15
2. Snowflake schema of the Animal class object.	28
3. AnimalEvent and context-dependent sub-dimensions.	30
4. Application Framework.	39
5. Multidimensional data cube.	67
6. Concept hierarchy for the Time dimension.	68
7. Multidimensional model of the Arc Marine MeshFeature class.	71
8. MeasuredData data cube model in core Arc Marine.	73
9. MeasuredData data cube in the MMI customization.	74

LIST OF APPENDIX FIGURES

<u>Figure</u>	<u>Page</u>
A1. Structure of InstantaneousPoint	106
A2. Creating a feature class from an Excel table	107
A3. Arguments for Create Feature Class From XY Table	108
A4. Attributes of XYDemonstration	109
A5. XYDemonstration with InstantaneousPoint schema	110
A6. Executing the loading of InstantaneousPoint.....	111
A7. Field matching in the Simple Data Loader.....	112
A8. InstantaneousPoint populated from DemonstrationSet.xls	113
A9. OBIS-SEAMAP dataset browsing screen.....	114
A10. OBIS-SEAMAP dataset download screen.....	115
A11. Creating a feature class from the CSV file.....	116
A12. Arguments for Create Feature Class From XY Table for any OBIS- SEAMAP CSV file	117
A13. “Dataset” OBIS-SEAMAP schema.....	118
A14. “Owner” OBIS-SEAMAP schema	118
A15. MMI sperm whale data with datasets from OBIS-SEAMAP	119
A16. Setting the Definition Query to exclude “bad” points.....	121
A17. Using LocationSeries with Hawth’s Tools.....	122
A18. Using LocationSeries with XTools	123
A19. Animal paths loaded from XTools output into Track.	124

LIST OF APPENDIX FIGURES (Continued)

	<u>Page</u>
A20. Three selection paths.	125
A21. InstantaneousPoint input to the Kernel Density tool.	126
A22. LocationSeries input to Standard Distance geoprocessing script.	126

Arc Marine as a Spatial Data Infrastructure: A Marine Data Model Case Study in Whale Tracking by Satellite Telemetry

Introduction

Within the current presidential administration, President Bush's 2004 Ocean Action Plan (Committee on Ocean Policy, 2004) includes reauthorization of the Magnuson-Stevens Fisheries Conservation and Management Act, the designation of the Northwest Hawaiian Islands Marine National Monument, and the establishment of the Gulf Coast Regional Plan. Meanwhile, in the current legislative cycle, the United States Congress will once again address the National Oceanic and Atmospheric Administration Act (U.S. House, 2007b), authorization of the Marine Mammal Rescue Assistance Grant Program (U.S. House, 2007a), and revision of the Marine Mammal Protection Act (U.S. House, 2007c). In the atmosphere of these major legislative movements, research interest continues to grow in the designation and management of protected regions, habitats, and species in the territorial and economic waters of the United States.

The management of the nation's marine ecological resources depends on the constant improvement of scientific methods and information resources among the researchers of the marine community. These improvements must come in the form of better information and better access to information. To this end, the marine community must

develop standard methods of data management and analysis which provide rapid dissemination of data, easy comparability of research findings, and simple means to carry out complex analysis. These are among the goals of the proposed community developed template, the Arc Marine data model. The research presented here is foremost an evaluation of the ability of Arc Marine to reach such goals in the context of a standard methodology in marine ecology research: marine animal tracking.

Marine animal tracking is a central component of research into the patterns of movement and distribution for endangered and economically important marine species. These movement patterns and distributions, coupled with habitat classification, drive the management of marine ecological resources. Marine animal tracking attempts to answer a series of question critical to marine resource management.

- *What are the spatial and temporal distributions of key animal populations?*
- *How do species interact with each other across their ranges?*
- *What is the relationship between physical and biological processes and the distribution and movement of critical marine species?*

Discovering the answer to these questions has always required an expertise in the biological sciences. Now though, remote sensing in the marine animal tracking field is making major contributions to these questions.

Satellite telemetry through the Argos system (Boehlert et al., 2001; Le Boeuf et al., 2000) and the Global Positioning System or GPS (Teo et al., 2004), geolocation through recorded day length (Hill, 1994; Welch and Everson, 1999; Boustany, et al., 2002; Shaffer et al., 2005; Wallace et al, 2005; Weng et al., 2005), and archival popup tags (Block et al., 1998; Dewar et al., 2004; Teo et al, 2004; Block et al., 2005) allows researchers to examine the behaviors of marine vertebrates without time consuming and costly direct observation. The gathered data are spatially oriented and globally distributed. The analysis of these data requires a spatial context, integration of multiple environmental datasets, and examination at a wide range of extents and grains. Marine animal tracking researchers are therefore transforming from marine biologists into marine biogeographers. With the large datasets involved, researchers now turn to geographic information science to answer to the basic question of how to explore these data as well as the deeper question of how to find relationships between animal distributions and oceanic processes. At this time, such questions are only being explored at the level of data repositories and scattered pilot projects. Even at the largest such repository, the Ocean Biogeographic Information System - Spatial Ecological Analysis of Megavertebrate Populations project (OBIS-SEAMAP), tools exist for mapping and animation of any dataset, but only four species specific pilot projects have begun to relate marine biogeographic data to other spatial datasets (Read

et al., 2006). One of the key challenges to the advancement of the research field is the development of a common set of computational and data management approaches (Block, 2005).

The Arc Marine Initiative

The focus of this case study is further specialized customization of the Arc Marine data model to meet the data management needs of the marine animal tracking community. Arc Marine is a geodatabase schema created by researchers from Oregon State University (OSU), Duke University, the Danish Hydrological Institute-Water-Environment-Health (DHI-Water-Environment-Health), and the Environmental Systems Research Institute (ESRI), as well as a larger team of reviewers and the input of the marine GIS community at large. Although Arc Marine is currently application-specific to ArcGIS, it represents a developing and evolving community standard for marine research. The data model facilitates the transition of marine geospatial applications to an object-oriented data model and data structure such as the geodatabase structure utilized by the ESRI software package ArcGIS 9 (Wright et al., 2005). With a formal object-oriented data structure, marine biologists can accelerate analytical research, facilitate collaboration, and increase the efficiency of ongoing field research.

Arc Marine is the result of an initiative that began with the first ESRI marine special interest group (SIG) meeting at the ESRI International User Conference in July 2001 in San Diego, CA (Breman et al., 2002). The history of Arc Marine, though, extends back beyond that first meeting to where the marine GIS community expressed interest in developing a marine data model. In 1999, ESRI introduced ArcGIS 8, and with it the geodatabase data model (ESRI, 1999). This data model opened up object-oriented modeling within ArcGIS, and brought a GIS implementation of relational database management. The generic geodatabase is, thus, a foundation for application specific data models; and to encourage these application specific data models, ESRI created the industry data model initiative (ESRI, 2000b). Starting with the Water Facilities Model, this initiative has developed a series of industry specific GIS solutions developed as extensions to the geodatabase data model (ESRI, 2000a). Intended to reflect best practices in the field, the data models are built on community standards. In ESRI's defined development process, an ESRI industry manager (for Arc Marine, Joe Breman) organizes a core group representing public and private sectors, business partners, and user communities (ESRI, 2007b). More essential than the industry manager is community interest such as that expressed at that 2001 marine SIG meeting.

For Arc Marine, the initial working group included representatives of ESRI, Duke University, Oregon State University, and DHI-Water-Environment-Health. The core group developed a draft marine data model with informal input throughout the marine GIS community. This community includes GIS users in academia, resource management, marine industry, and government who apply GIS solutions to all marine environments from the deep ocean to coastal estuaries. Through a series of workshops at Redlands, CA, in 2002 and 2003, the core group took the input of an expanded informal review team to refine the marine data model from an initial draft into a mature industry solution. The model was further refined with the input of initial case studies, technical workshops, and conference paper sessions. The end result of this phase of development was the publication of the Arc Marine data model in book form (Wright et al., 2007). See Wright et al. (2007) for a detailed history and discussion of the development of Arc Marine.

Though the final data model has been published, this is only the beginning of community development of the data model. This case study represents one facet of this community development as an attempt to customize Arc Marine for the marine animal tracking user group. As such, this case study begins with the archived and live satellite telemetry datasets of the Oregon State University Marine Mammal Institute (MMI). Through the customization of Arc Marine, the assembled data of the MMI

can be linked to any other marine data set conforming to the Arc Marine standards. Information methods developed in this case study can benefit not just the research of the MMI, but any research program that utilizes geolocation to study marine ecology. The Arc Marine data model carries six specific goals (Wright et al., 2007), which can be adapted to the marine animal tracking community as follows:

- 1) Create a common model for assembling, managing, and publishing tracking sets, following industry-standard methods for dissemination (such as XML and UML).
- 2) Produce, share, and exchange these tracking data in a similar format and following a standard structure design.
- 3) Provide a unified approach that encourages development teams to extend and improve ArcGIS for marine applications.
- 4) Extend the power of marine geospatial analysis by providing a framework for incorporating object-oriented rules and behaviors into data composed of animal instance locations and dealing more effectively with scale dependencies.
- 5) Provide a mechanism for the implementation of data content standards, such as the OBIS schema extension of the Darwin Core Version 2 (OBIS, 2005).
- 6) Aid researchers in a fuller understanding of object-oriented geographic information systems (GISs), so that they may transition to

powerful data structures such as geographic networks, regions, and geodatabase relationships within an easily managed context.

Previous case studies have addressed the general suitability of Arc Marine for most of these goals (Aaby, 2004; Halpin et al., 2004; Andrews and Ackerman, 2005; Wright et al., 2007). The OBIS-SEAMAP group at Duke University produced the first application of marine animal tracking with Arc Marine (Wright et al., 2007, pp. 45-80). This thesis further examines the effectiveness of Arc Marine in meeting the six goals in a marine animal tracking context. Particular emphasis is placed on two areas: first, the implementation of Arc Marine in on-line analytic processing and data warehousing to enhance data exchange and provide access to high level data mining techniques; and second, the definition of a marine animal tracking specific program interface and application framework to facilitate the development of back end (data extracting, loading, and cleaning) and front end (querying and analysis) tools.

The Marine Mammal Institute

Since 1983, the MMI has used satellite telemetry tags to track the movements of the great whales. These investigations have unlocked the migratory routes and habitats of Right whales, Blue whales, Humpback whales, Fin whales, Gray whales, and Sperm whales. By mapping the distributions and abundance of whales throughout their migration, feeding,

and breeding activities, the Marine Mammal Institute hopes to identify anthropogenic activities, which stifle the recovery of the species. This research will thus ultimately lead to solutions to enhanced recovery of the great whales (Sherman 2006). Over the last three decades, the tagging program has moved from short-range conventional radio tracking to satellite based radio tags to track whales and dolphins, primarily along the Pacific and Gulf coasts of North America (Mate, 1989; Mate et al., 1994). Through this tagging program, the MMI is discovering the distributions and movements of endangered species whose critical habitats are unknown for most of the year.

Since 2000, the MMI has worked with the Census of Marine Life's Tagging of Pacific Pelagics (TOPP) project. This pilot program, funded by grants from the Alfred P. Sloan Foundation, NOAA Office of Exploration, the Office of Naval Research, and many other sources, seeks to explore the Eastern Pacific from the perspective of twenty-one selected predator species divided into seven groups: cetaceans, fish, pinnipeds, sea turtles, seabirds, sharks, and squid (TOPP 2006). Bruce Mate of the MMI is the Cetacean Group leader. Four cetaceans of the nine species studied by the MMI are included in the twenty-one selected species: Blue, Fin, Sperm, and Humpback whales. Under the guidance of TOPP and the MMI, the group has filled unknown portions of the life histories of these species including feeding patterns, breeding areas, and migration routes. As the

dataset continues to expand and develop, it is becoming a necessity to tie this spatial information to the whales' environment: the physical, chemical, and biological components of the marine ecosystem.

The four cetacean species (Tagging of Pacific Pelagics, 2006) in the TOPP project have been selected in part for their significantly overlapping ranges. Not only does this aid in data analysis, but these overlapping ranges also allow for the tagging of multiple species in deployment operations. The datasets produced by the cetacean group are widely dispersed chronologically, seasonally, and spatially, and are used with data from bathymetry and chemical, physical, and biological oceanography. Though much can be learned just from the descriptive aspects of this information base, ultimately unlocking answers to this central question will require the consolidation and aggregation of this array of data into correlative statistical analysis. To this end, the information must be integrated into a unified spatial database.

The primary goal of the OSU MMI and TOPP Cetacean group is to find the environmental preferences that determine the critical habits of endangered whales (Block, 2005). As this body of knowledge develops, the group can better address the information demands of policy makers attempting to designate protected areas for the preservation of these critical habitats and the species which depend on them.

Advantages of Satellite Telemetry

In pursuing the movement and ranges of the great whales, satellite telemetry provides four key advantages: timeliness, continuous coverage (Lagerquist et al., 2000), relationships to environmental data (Block, 2005), and autonomous profiling of the animal's environment (Boehlert et al., 2001). Timeliness or responsiveness allows remote sensed observations to translate into key management information in a matter of hours or even minutes, as opposed to the months needed to fully realize the returns from marine surveys. The continuous coverage of satellite telemetry means knowledge of not only an animal's current location, but also where the animal was last week, next month, next season, or possibly even next year. The individual can be followed as it moves from breeding grounds along migration routes to summer ranges. This not only allows the identification of key habitats and migration routes, but also the determination of the timing of migration, feeding, and breeding.

The spatial character of these individual movement paths allows precise coordination with dynamic environmental datasets (Block, 2005). This allows a deeper understanding of the natural variability in whale movements and how this variability relates to season, changing environmental conditions, and underlying geography. Finally, the tagged whale is able to act much like an autonomous profiling glider (Boehlert et al., 2001), moving through its critical habitat and recording the

environmental conditions from the animal's point of view. This autonomous profiling provides a remotely observed view of the critical factors that drive the decision processes that determine when and where the individual whale moves.

Research Questions

Bringing the MMI satellite telemetry program together with Arc Marine generates the central research question of this case study.

How can the Arc Marine Data Model be customized to best meet the research objectives of the OSU MMI and the marine animal tracking community?

This primary question leads to two secondary questions.

1) How can a geographic information system implementation enhance the key advantages of satellite telemetry?

2) In a marine environment with dynamic environmental conditions across a three-dimensional space, what is the optimal application framework to allow multi-level access from multiple users?

Customization of Arc Marine for the purposes and requirements of the MMI is the first step to addressing these questions. This customization also provides further insight into Arc Marine as a historical data repository, or data warehouse. Finally, this case study presents the development tools of a proposed programming structure, application framework, and

multidimensional data view derived from the MMI customization. These development tools, in turn, facilitate the development of community-wide tools for data warehousing, back-end data loading, and front-end analysis (including data mining) that will speed the adoption of Arc Marine as a marine GIS community standard.

Methods

Argos Satellite Telemetry

The central focus of this Arc Marine case study is the creation of a research repository for satellite telemetry data gathered by the OSU-MMI. The specific field methods employed in this tagging are outside the scope of this case study, but are covered in detail (including historical development of tag hardware and deployment techniques) by Mate et al. (2007). The most important factor in this case study where field methods are concerned is the use of the Argos data collection and location system.

Argos platform transmitters resolve location using Doppler shift principles. Argos consists of a network of four polar-orbiting satellites which circle the earth every 101 minutes. Using the assumption of a stable transmission frequency and motionless platform, the system can use multiple measures of Doppler frequency shift to construct a circular solution set (intersections of spherical distance) around the satellite's path of motion. By collecting multiple transmissions (Service Argos requires a minimum of four passes), the location of the platform is determined with greater accuracy, resolving into the intersection of this circular solution with the elevation sphere of the platform (Figure 1). In an ideal situation, the satellite passes directly over the platform, creating a single point of tangency with the elevation sphere, but in most the intersection creates a true location and a mirror point on the opposite side of the satellite's path.

Although the algorithm for deriving this set is far more complex than presented here, there are two critical pieces of information for creating tag behavior. First, nearly all location sets consist of a true position and a

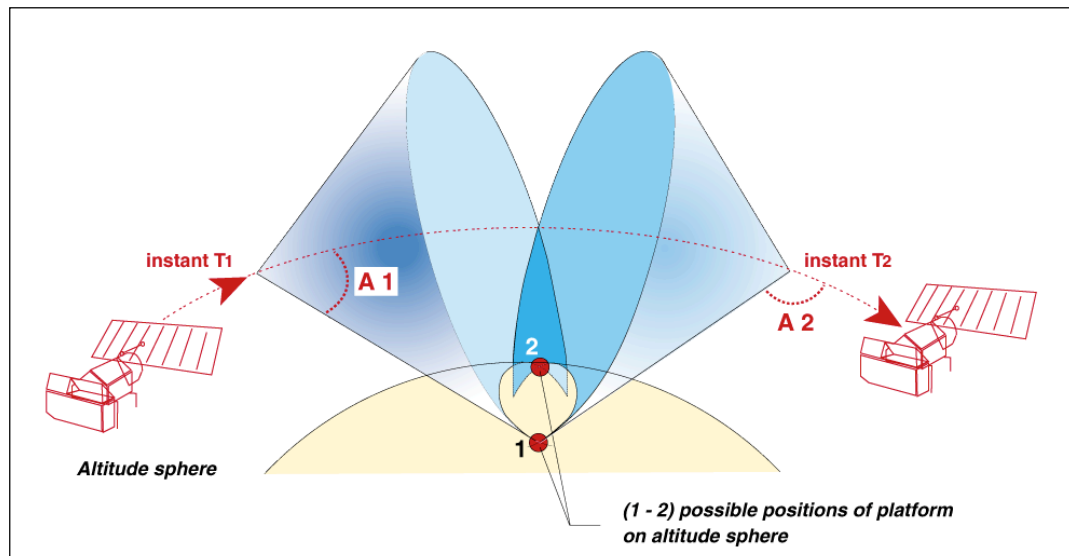


Figure 1. The solution pair generated by Service Argos geolocation. Adapted from Liaubet and Malardé (2003).

mirror position. Second, the residual error will vary; more transmissions received and true location closer to the poles will reduce error. Service Argos returns a probable solution based on the least residual error, frequency continuity with the last calculated position, and minimum movement from the last calculated position (Argos, 1990; Kinzel, 2002; Liaubet and Malardé, 2003). Thus, the data received consist of a solution pair with one point designated as the most probable solution. In addition to the probable solution, many other algorithms exist for selecting between

solution pair points. Austin, McMillan, and Bowen (2003) provide an overview of three common methods used for this process.

The Argos service also returns information on location accuracy. This accuracy is reported as a location class based on the number of transmission messages received from the transmitting platform. For class A and class B, no location accuracy is available due to too few satellite messages. For class zero location accuracy is 1500 m; class one 1000 m, class two 350 m, and for class three 150 m (Argos, 1990). In a sample of sperm whale locations used to test components of the customization, there was an average difference of 3.6 km between successive locates. Hence, location accuracy is a small but significant source of error, especially for those locations with unknown accuracy.

A Brief Tour of Arc Marine

Much of this case study focuses on the creation of a customized version, or extension in the terminology of class inheritance, of the Arc Marine data model to fit the specific research objectives of marine animal tracking. Arc Marine is, foremost, a schema to support data collection on dynamic and multidimensional marine phenomena in a manner that models the real world in an object-oriented geodatabase (Wright et al., 2007). This schema creates distinct community-wide advantages by increasing data interoperability, reducing analytic complexity, and facilitating tool

development and dataset exchange. For the purposes of customization, the most important structural aspect of the data model is the availability of standardized classes to represent model entities and the relational joins, built in to the model, which provide guaranteed relationships between data tables for complex querying.

In this study, as suggested by Wright et al. (2005), the core of the data model is kept intact. All core classes retain their original attributes to ensure compatibility with code from other sources. As such, class inheritance is used to create customized versions of core classes. As well, customizations added to the data model are made as generic and universal applicable as possible to encourage reuse of the customization by other developers from the marine animal users group.

A detailed diagram of the Arc Marine schema is presented in Appendix A, but of the core Arc Marine objects, the three critical objects to this customization are the InstantaneousPoint feature class, Vehicle marine object class, and MarineEvent marine object class. InstantaneousPoint is a point feature representing a unique observation defined in time and space by geographic coordinates and a timestamp. LocationSeries, a subtype of InstantaneousPoint, allows for the spatial and temporal sequencing of a series of points moving through space. Again, each point represents a single unique observation. In this study, this subtype holds the critical geometry of satellite locates from the Argos telemetry messages. Animal

tracks (as Track feature classes) are composited from the interpolation of movement between points. As such, tracks are calculated as on-the-fly features and not stored in the geodatabase. Typically, this class combined with the Series object class would define the movement of an animal.

An animal may be modeled in one of several different forms, taking on different classes depending on the animal's behavior compared to the object model. Most often, this representation is either as a MeasurementPoint representing a single observation of the animal in a survey or as a Series of LocationSeries points and associated Track representing the movement of a single animal. Yet in this case study, point modeling was not deemed appropriate for representing a tagged animal. Complex multi-dimensional data are difficult to connect to single points, particularly when the data come from multiple sensors that are collecting between satellite fixes. Much of these data are associated with a time and an instrument rather than a specific location.

The Vehicle object class is a less utilized class which generally stores information about a vehicle used during a survey run. Hence, the object class relates to both the MeasuringDevice object and Track feature class. Here, the important characteristic of the Vehicle object class is that it models a moving, instrument-carrying platform. Generally, this would be a survey vessel, but in this instance the moving, instrument-carrying platform is an animal.

MarineEvent is meant to be used for linear referencing of time or distance mileposts (M-values) along linear features such as coastlines or ship tracks. As mentioned above, data collected from tag instruments are often associated with a timestamp rather than a location. Thus, MarineEvent in a timestamp mode is a natural choice for the dynamic segmentation of animal movement path's to create spatial locations for these timestamped data.

In addition to the information in Appendix A, Arc Marine is also available as an XML schema view from Rehm (2007), as a GML/XML ontology from Lassoued (2007), and in various diagram forms (including the UML diagram in Appendix A from Wright, 2007).

Programming languages and software

Development of the MMI extensions to Arc Marine was carried out primarily with Microsoft Visio 2003 software, with some use of browser-based UML viewers. After conceptual development of the new class objects, the objects were converted into UML. This conversion began with the core data model UML to avoid repetition of the conceptual design efforts of the Arc Marine group in defining common marine data types. New domains were added to the existing Domain layer while new classes were added to additional layers in the UML. All classes inherit behavior from ESRI Classes::Object. The Visio UML template does contain additional

notes about the usage of certain fields in classes, but no additional code is contained in the UML template. Within Visio 2003, the UML is exported to an XML Interchange file to be used as a database template. This XMI file can then be used with CASE tools in ArcCatalog to create a new instance of the extended data model to populate a new geodatabase (personal, file, or ArcSDE) with classes. Records are populated with developed data loading scripts. For programmatic interactions, field contents are transferred from database tables into programmatic objects via loading functions.

As an important note to the process of database schema building in UML, this export is carried out using the Visio 2003 UML to XMI export utility available from Microsoft Corporation(2003) in combination with an ESRI methodology (ESRI, 2003). Microsoft has not made an UML to XMI export utility available for Visio 2007. Thus, Visio 2007 cannot be used at this time to generate modified database schemas for use by ESRI CASE tools. IBM's Rational Rose product presents an alternative option for a supported UML building tool with XMI export.

This case study was developed on ArcGIS 9.2 using SQL Server 2005, ArcSDE on SQL Server, and Microsoft Access. The developmental databases have been instantiated in personal geodatabases while production is carried out in ArcSDE and SQL Server. The legacy database resides on a personal geodatabase in Microsoft Access as well as

additional data in Excel spreadsheets and text files. Based on the ArcGIS 9.2 environment as well as the programming team expertise, the choice of programming languages for development came down to the .NET framework (in particular VB.NET) and Python. While there are other languages to consider including C#, C++, and Java, the easy access to geoprocessor scripting in these two languages and to geodatabase records through the geoprocessor made VB.NET and Python the natural options in ArcGIS 9.2. Existing reusable code base for database querying and the download of satellite telemetry results further supported these choices.

Python development is based on Python 2.4. Not only is this consistent with the latest release of ArcGIS, but Python 2.4 also makes available the *datetime* module that simplifies comparisons between timestamps. This module is not available in Python 2.1, the version supported by earlier versions of ArcGIS. As this is an ArcGIS 9.2 programming environment, python modules use the `arcgisscripting` module. Previous to version 9.2, geoprocessing scripts relied upon a call to COM IDispatch to create a geoprocessing dispatch object (`esriGeoprocessing.GPDispatch.1`). While this procedure of accessing the geoprocessor is still available with ArcGIS 9.2, the dispatch object limits the script execution platform to Windows operating systems. With the native `arcgisscripting` module, the scripts are truly cross-platform, but also are not backwards compatible with the Python 2.1 and COM IDispatch

based environment of earlier versions of ArcGIS. This creates a significant advantage over Perl, VBScript, and JScript which still rely on the Windows platform dispatch object for access to the geoprocessor. Use of the arcgisscripting module, like the GPDispatch, gives cursor access to database records and provides access to attributes and methods of geodatabase tables and feature classes. This means that inheritance can be used to extend the base data model objects and add custom behavior to the object classes.

Extending Arc Marine

Data model extension development followed an abbreviated form of the data model design process (Li, 2000; Wright et al., 2005) from external design to conceptual design to logical design to physical design. The external design (the simplification from the real world to application scope) and the bulk of the conceptual design (development of entity-relation diagrams to populate the model with objects) are represented by the core Arc Marine data model and should not be replicated. Rather, this customization required only rudimentary conceptual design of components specific to this case study and absent from the core model. In particular, no new spatial objects were developed. Spatial entities were created as child classes of core classes, inheriting and extended these generic objects with additional methods (behaviors) and attributes, but not new geometry. All

new entities added to the model are represented in conceptual design as object classes related to feature classes and not as feature classes themselves. This particular step ensures that the new classes will interact appropriately with any spatial analysis tools developed for Arc Marine.

The bulk of the design process occurred in the logical design phase as entity-relation components were added to the core Arc Marine UML schema using Microsoft Visio. All elements of the schema at a higher level than the extended classes were simply carried over without modification from the ESRI data model and the core Arc Marine data model, greatly minimizing the scope of the logical design phase.

This implementation in UML, though, only addresses the attributes of the new object classes and class extensions. Object behavior was implemented programmatically with further subclassing in the software development language. Essentially, while the attribute customization takes place in the logical design, the behavior customization is dependent on the hardware and software implementation of the model in the physical design.

While development in VB.Net places behavior implementation squarely in the physical design for the ArcGIS/Windows operating environment, Python implementations blur the line between logical and physical design. The cross-platform compatibility of Python means that the actual programmatic representation of behavior is platform independent. Only the physical recording of the outcomes of behaviors are specific to the

physical implementation. Essentially, the Python codes become the importable schema; a strong argument for separating analytic and behavioral code modules from code modules dealing purely with reading and updating database records.

Even though the core of the data model is fully retained without modification, the customization still employs complex database structures. The implemented geodatabase is still contains highly normalized tables, join tables, sub-dimensions, context-dependent joins, indexes, and other optimizations to allow complex querying and reduce redundancy. Though in some cases, such as with the Animal object class that extends the Vehicle class, subclassing is used to extend the attributes of core classes, most often the model is extended by creating relationships to new object classes subclasses from the ESRI Classes::Object class.

Results

Customizing Arc Marine

The Arc Marine data model customization developed for the MMI consists only of non-spatial object classes, but with explicit relationships to spatial marine feature classes. Basic conceptual design for the customization identified three groups of objects to add to Arc Marine for the purpose of marine animal tracking. The animal group develops and expands the base representation of animals as MarinePoints. The telemetry group directly represents raw data and transformations of the data including location returns, data quality information, and data collected from tag sensors. The operations group was developed in relation to the Cruise object class and Track feature class to provide auxiliary information about the events represented by those feature classes.

Two new auxiliary entity-relationship groups were also developed: telemetry tags (and hardware components) and data filtering (with object modeling of component functions to create a normalized filtering audit trail). These two groups are used, respectively, for back-end and front-end functionality outside the data model and intended as a tool for developers. The new objects in the descriptions below are interchangeable referred to as class objects and tables. Class objects are, more specifically, the entity representation in the data model; tables are the logical implementation of these class objects in a database. The term “table” is used when

discussing how the database representation of the class object is used to store data or build join relationships to other database tables. The overall customization is depicted in the UML diagrams of Appendix B.

Animal

The Animal class is a child class of the Vehicle parent class. This choice is dependent on this particular marine application of the data model. In other cases, an animal might be better represented as MeasurementPoint, possibly with related survey data. In this case though, the animal itself is not an observation but rather it is its own instrument platform. The animal is carrying a collection of measuring devices that measure a range of quantities including location, depth, temperature, salinity, and incident radiation, or even complex attributes like surfacing rate or dive profile. This is analogous to a ship carrying a conductivity, temperature, and depth array with a GPS (though greatly miniaturized). Like a vehicle, an animal creates a Track (recorded by a measuring device) along which the attached MeasuringDevice array (the tag) records data.

An animal though is not simply a vehicle. As a specialized type of vehicle, an animal has a species, genotype, sex, social group, and length (the latter two based on the initial observation of the animal). The Animal object class is also related to BiopsyInfo, AdoptionInfo, and Species object classes. The BiopsyInfo object class represents data on individual biopsies

(and the related approach to the animal) and will eventually link to genetic information beyond the genotype as that part of the MMI program is developed. AdoptionInfo is an administration table related to fundraising that can be extremely helpful for such tasks as transmitting a tracking map for a specific whale to a donor who has adopted that whale. The Species object class not only avoids the redundancy of repeatedly storing genus, species, and common name in the animal table, it also allows linking to species specific information such as maximum speed parameters (in the SpeedLimit table).

Since the animal is a specialized type of vehicle, it can carry MeasuringDevices (in this case, the satellite tags) that relate directly to MeasuredData. These measured data, though, are often derived from raw satellite telemetry data that can carry poor location accuracy or no location at all. Additionally, with Argos fixes there are two possible locations. The animal and the measured data are linked to these quality data and alternative locations through the AnimalEvent table that is the core fact table for much of the database (See Figure 2).

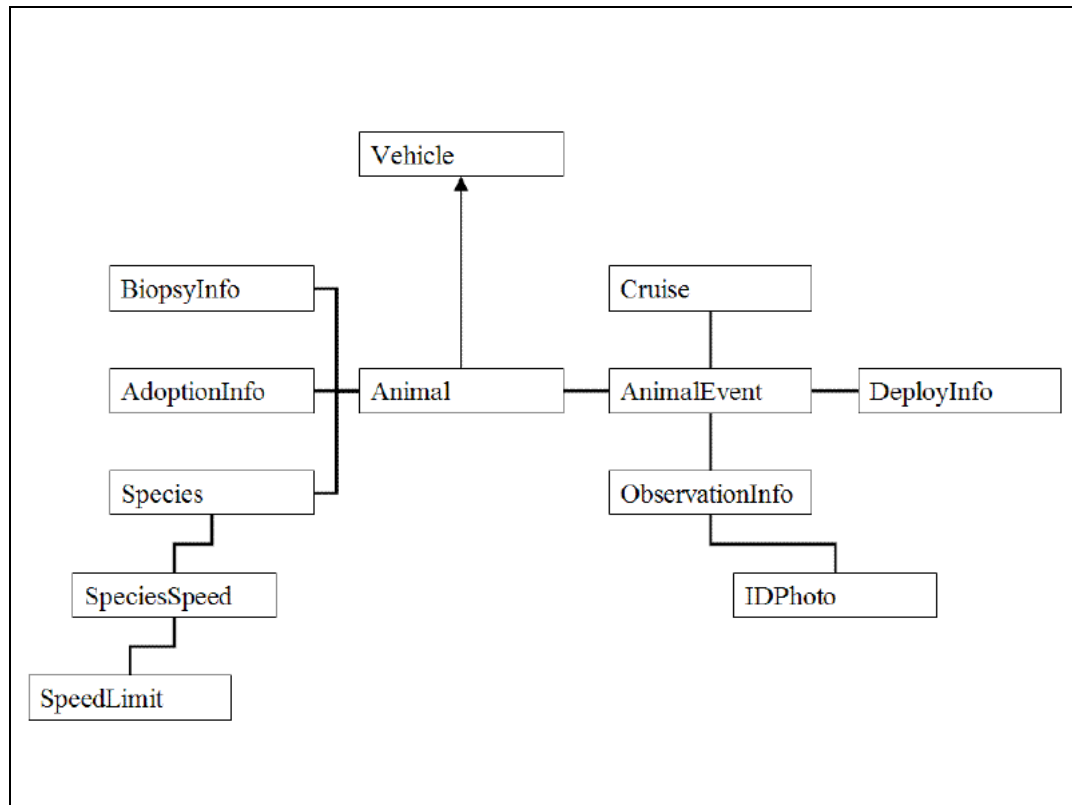


Figure 2. Snowflake schema of the Animal class object. Animal is a dimension table of BiopsyInfo, AdoptionInfo, and AnimalEvent, while Species is a higher level hierarchy concept of Animal.

As a class object, Animal can be extended with additional methods and attributes. In operations such as agent-based modeling, this allows the use of object-oriented programming instead of procedural programming. The agent model is designed programmatically as a child class of the Animal class object, allowing full access to the attributes, relationships, and spatial context of instances of the Animal class. Applying similar object-oriented strategies to other object entities, such as dynamic coastlines, environmental mesh models, or prey agents (which can be a separate child

class of Animal) allows for a full agent-based modeling environment in which individual researchers and developers can add or remove components without major restructuring of the model code base.

Telemetry

AnimalEvent is the core relational table, or fact table, of the telemetry portion of the database schema. It anchors the LocationSeries and Track feature classes to telemetry information stored in the extended database as well as tying together the animal, tag, and tag deployment (part of operations) in a star schema. AnimalEvent is similar to the MarineEvent class, but for time referencing rather than linear referencing and for both object classes and feature classes. As noted by Wright et al. (2007, pp. 45-80), MarineEvent is intended to hold only a single value and cannot respond to the many parameters of an animal sighting. Similarly with telemetry, a MarineEvent can tie a single value to a specified a start and end location along a Track. AnimalEvent though can relate complex parameters (through sub-dimension tables and a relationship to measured data) to start and stop points in time. Dynamic segmentation along a timestamped Track fulfills the same geolocating purpose as MarineEvent.

AnimalEvent sub-dimension tables are context dependent (Figure 3). The table joined by AnimalEvent is dependent on the context of the event. Argos tag collection events link to ArgosInfo, tag deployments link to

DeployInfo, field observations link to ObservationInfo. The number of potential sub-dimensions is limited only by the number of types of

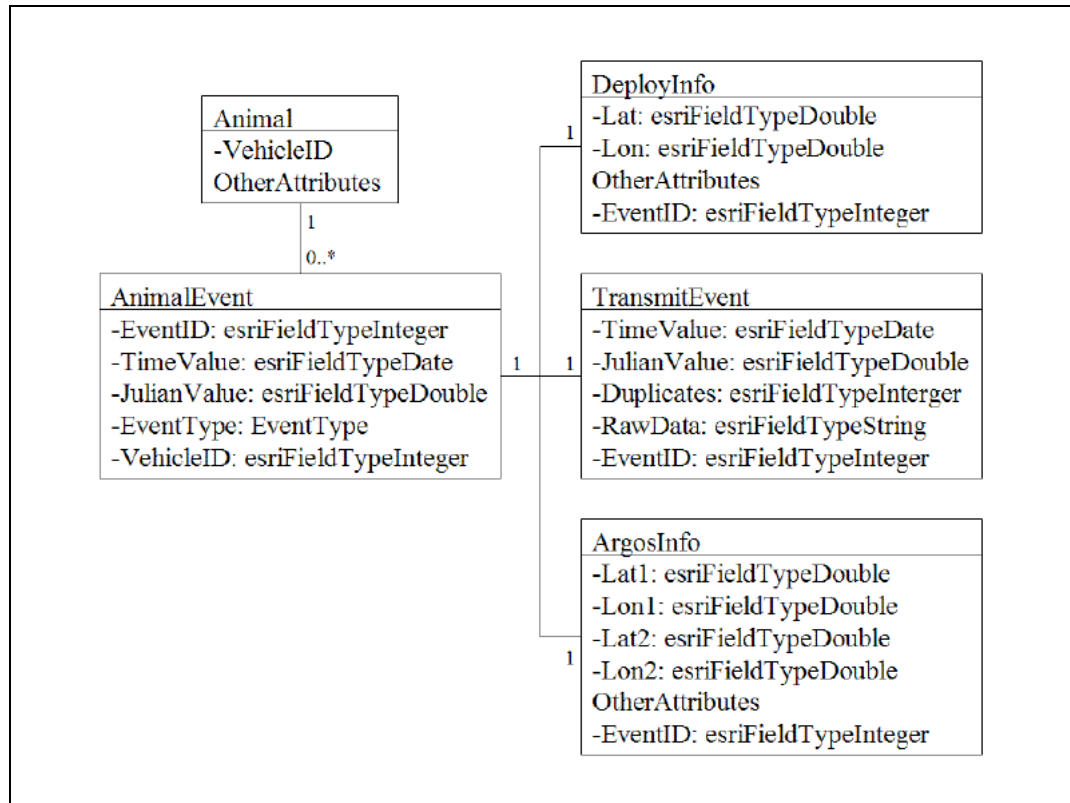


Figure 3. AnimalEvent and context-dependent sub-dimensions.

interactions with the animal. In particular, each new tag type links to a new sub-dimension table. As new tag types are added with different auxiliary attributes, new sub-dimension tables will be added.

Table 1. Context-dependent sub-dimensions of AnimalEvent

DeployInfo	Deployment of a measuring device onto an animal
ArgosInfo	Auxiliary information, Argos locations
ObservationInfo	Auxiliary information, field observations and photos
DerivedInfo	Auxiliary information, interpolated or derived location
GPSInfo	Auxiliary information, FastlockGPS locations

These sub-dimension tables each carry a one-to-zero-or-one relationship with the AnimalEvent table; thus the joins from the animal to event information are context-specific (Figure 3). Though context-specific joins increase the complexity of query building, this aspect should be handled seamlessly by the interacting analysis tool. In exchange, the cardinality of the sub-dimensions is significantly reduced (particularly low frequency events such as DeployInfo). Note that attributes specifically needed for analysis are still stored in MeasuredData and spatial information is still stored in the geometry of feature classes. The AnimalEvent sub-dimension tables only provide access to auxiliary information related to a specific event.

Operations

The operations group is divided into two areas, CruiseOperations and Approaches. CruiseOperations involves a small number of generic object classes to link field observations to the person making the observation. Approaches handle the specific operational situation of approaching an animal and deploying a tag.

CruiseOperations is essentially a customization of the SurveyInfo aspect of Arc Marine. SurveyInfo links an InstantaneousPoint to a unique survey operation. This point may represent a sighting, photograph, deployment, telemetry location, or a wide variety of other features. When

this point is linked to a survey though, that survey has a specific crew, identified by CrewKey, and specific crew members in that crew, identified by the Crew class object. Thus, a crew has crew members and carries out one unique survey. SurveyInfo is also a dimension of the ApproachEvent, a linking dimension table for the Approach object group.

ApproachEvent is a series of one-to-one related class objects which describe the specific instance of deploying a tag to an animal. This is an important special case, as this particular event ties together an Animal and MeasuringDevice to begin a Series. It is possible to completely omit the ApproachEvent and simply record which MeasuringDevice has been deployed to which Animal, but the significance of the event to marine animal tracking (particularly with the permitting requirements of marine mammal tracking) warrants specific inclusion in the database schema. DeployInfo is the linking table for this group. First, this table records a wide range of event parameters as an AnimalEvent sub-dimension. After all, deploying a tag to an animal is a rather monumental interaction in the study of that animal. This table also links to the specific tag deployed in MeasuringDevice and the ApproachEvent (which links to SurveyInfo and additional information about the specific approach). Thus, from Animal to AnimalEvent to DeployInfo to MeasuringDevice, the animal is initially linked to the tag instruments that it carries.

Tag

The Tag group is the first of the auxiliary groups developed in this customization. Tag is not directly a necessary component of the complex relationship between sensor measurements, telemetry, and animal movement. Rather, the objects in the tag group supply information critical to the preprocessing of satellite returns as well as the planning of hardware for future tag deployments. This group is a snowflake schema with MeasuringDevice, modeling deployed tags, as the central fact table. The dimensions of this schema are TagType and BitStructure. TagType represents the specific hardware construction of the tag, including individual components as a sub-dimension. BitStructure is a binary decoding class object used in back-end data loading to supply the structure of raw binary messages from a specific tag.

While Transmitter carries the one-to-many relationship typical of a dimension table, it is only a descriptive table which supplies the Argos platform transmitter terminal (PTT) assignment so that the tag's returns can be automatically extracted from the text files supplied by Service Argos. Schedule, and the related ScheduleType and ScheduleDetail, is also not a dimension of MeasuringDevice. It cannot be used as an aggregating classification, as indicated by its many-to-one relationship with MeasuringDevice. Instead, it is another descriptive class object indicating the duty schedules of a specific tag.

Filtering

While the Tag group including several class objects useful to back-end data loading, the filtering group is designed to handle the front-end analytic task of selecting between Argos mirror points. In the context of other tag types, the filtering group can also be used to indicate variable uncertainty, accuracy, or the exclusion of potential telemetry zingers.

There are two aspects to the filtering group. Flag, FlagParameter, Functions, and FunctionParameter, represent information attached, through flag, directly to an InstantaneousPoint feature. Filter, with FilterStep and FilterStepParameter, is an audit trail of the specific processing steps taking to attach flags to an InstantaneousPoint.

Flag, by itself, conveys no information other than the priority, or reliability, assigned to a point. Reading along a Series of LocationSeries points, the flags would indicate whether to use a point, use its mirror, move the point (for example, if it is on land), interpolate a new location, or skip the point altogether. The FlagParameters indicate the decision process (in terms of the output of filter functions) used to flag the point. Meanwhile, Functions and FunctionParameters handle the tasks of moving or interpolating or even carry instructions on how to construct linear interpolations between the point and its Series neighbors.

The actual outcome point set from applying filters is not recorded, only the filtering methodology to go from a base flag set to the flag set used

by the researcher in an analysis. This provides three features of filtering: 1) the filtering methodology is recorded by research and date used, i.e. an audit trail for use in later publication, 2) the filtering methodology is readily repeatable for additional analysis experiments, and 3) the resulting flag set is readily updated when the base flag set is refreshed with new telemetry locations from an active tag. To store data filtering and querying choices by researchers, `InstantaneousPoints` are assigned a default processing flag that indicates the origin, validity, and priority of the point for analytic processing (but no points are discarded). A researcher then works with a snapshot of the point features by applying a series of filters with arguments. The first filter will normally execute a query string against the default flagged set, but the researcher may also first remove default processing ("unsetting" the flags). This sequence of filter functions and parameters are saved with a `UserID` and creation date in `Filter`, creating an audit trail of research decisions. As each filter applies a specific function (`FilterStep`) with a specific parameter (`FilterStepParameter`), the snapshot can be recreated simply by re-executing the saved list against the default processing set. Filtering methodologies are not part of this object group. Instead, the `FilterStep` makes a call to coded filtering methodologies. The parameters used in that step are called from `FilterStepParameter`. Finally, the stack of applied filters is stored with a user stamp and timestamp in `Filter`, allowing for the three features mentioned above.

Development Framework

The choice of an application development framework is ultimately not an exclusive choice. With the geodatabase as a central connection between modules, it is possible to use a Python script toolbox linked to a Visual Basic based ArcMap extension, and all in coordination with a standalone .NET application utilizing multiple languages. In selecting an application framework, each of the three forms – toolbox, extension, or stand-alone application – has distinct advantages and disadvantages.

The stand-alone application carries an immediate advantage in licensing and accessibility. Not every researcher is a GIS analyst; an individual researcher or lab may have a preference for analysis in MatLab, R, S+, Excel, or other statistical applications. When distributing the tools to the tracking community, licensing requirements are reduced to ArcGIS Engine Runtime rather than an ArcInfo or even ArcSDE license. Yet, this accessibility of the application is offset by accessibility of the code. VB.NET carries a high learning curve with less modular code than Python scripting. Development will likely be centrally driven and the development time for a full stand alone application can be considerably higher than for an extension. As well, there must be a limit on the analytical scope of the application. While .NET allows access to the wide range of ArcGIS application programming interfaces or APIs, a standalone application cannot replicate the full spatial analysis and mapping capabilities of

ArcGIS. As a final advantage of the application, many operations, such as tag hardware inventory and documentation of ship operations, have no need for the full power of a GIS, and may even be hindered by the reduced querying abilities of ArcInfo. These types of activities may even be carried out by a technician or research assistant who does not need access to the larger geospatial dataset.

An ArcGIS extension carries lower programming overhead than a stand-alone application, but does not have the modularity and code accessibility of scripting. An extension, though, does offer true one-stop access to processing and analysis. The extension gives the ability to handle auxiliary data and metadata, spatial analysis, mapping, and querying all within ArcMap or ArcInfo, but with the licensing requirements of those applications. While such an extension can incorporate import functions to third-party statistical applications, carrying out data management in ArcGIS can limit adoption by research centers without appropriate licenses. Realistically though, many research groups in animal tracking will have institutional access to these licenses. Once again, the main disadvantage of an extension is the development time and centrally driven development. Without widespread adoption of the specific extension, a situation may emerge where every research group is reinventing the wheel.

This leaves the final option, Python scripting. The most pressing disadvantage with developing around a Python toolbox is a lack of access to the full ArcGIS APIs. While geoprocessor access can carry out many of the critical analysis and database updating functions, it cannot handle mapping and visualization tasks. Python, though, is supported by a quick learning curve that allows rapid development of sophisticated applications. The language is also strongly supported in multiple scientific communities, leading to the independent development of advanced graphical user interfaces (GUIs), statistical modules, and even tools for cross-platform development with .NET and COM which may provide workarounds for the API access disadvantage.

The chosen solution uses aspects of each form, though with a focus on Python scripting. Building on existing code, the standalone application will handle research and technician level access to update hardware and operations information or to create filtered data snapshots for analysis. The first priority, though, is the construction of automated tools for data updating, filtering, querying and export to analysis datasets. Generating the procedures in Python creates a rapid, modular development path, while the scripts can serve as the underlying code behind ArcMap toolbar buttons or a full-fledged Python-implemented GUI.

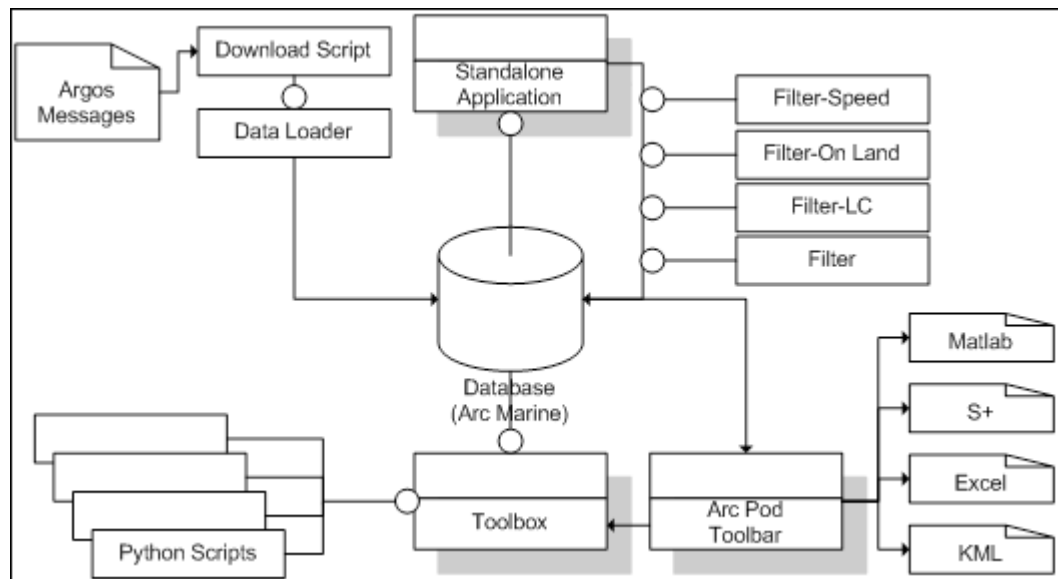


Figure 4. Application Framework.

The five major sections of the application framework are the database, download and data loader, standalone application with filter functions, analytic toolbox with python geoprocessing scripts, and integrated ArcGIS extension.

Figure 4 depicts the five major sections of the application framework that outlines potential future development for the Marine Mammal Institute. All pieces of the framework center on the animal tracking customized version of Arc Marine, including snapshot LocationSeries point sets used in data warehousing. In the upper left, automated daily updating is carried out by two Python scripts designed in a modular sequence. The first download script takes a series of connection parameters as an argument. That connection is used to download text results from Service Argos. The results are parsed to generate a Transmission container which holds, for each satellite transmission, PASS and DATA objects which respectively

carry AnimalEvent (ArgosInfo) data and raw binary (MeasuredData). The second data loader script takes this Transmission container and iterates through the PASS and DATA objects to construct LocationSeries points from the PASS data and new MeasuredData records from the DATA object. MeasuredData records are then reconstructed from the new MeasuredData (as well as the appropriate AnimalEvents such as GPSInfo for binary encoded Fastlock GPS results) according to stored procedures based on tag type. The binary translation procedures are separate from the data loader which is separate still from the download script. In this manner, changes to the tag program, database structure, or download service can each be dealt with separately without having to change the structure of the other components of the data update path. Each module simply has to generate return objects conforming to the argument requirements of the next module.

The next major component of the application framework, in the upper right of Figure 4, is the standalone application, or researcher/technician access level. This VB.NET based application implements only the key advantage of the standalone application: protected access to feature class snapshots and non-spatial information tables outside of ArcGIS. This application allows the execution of a limited number of short repeated operations in a low overhead program. More importantly, it ensures that the application of data selection filters takes place through a

controlled portal where an audit trail is fully implemented. With a limited scope, the application also requires a minimal amount of development time, though it may be replaced in the future by a full-fledged Python application using Python implementations of Data Access Objects (DAO).

The lower left represents the current emphasis of development, the analytic toolbox. This toolbox represents a series of Python scripts which rely on expected interfaces to Arc Marine objects. Thus, these tools can be shared with other researchers and applied to shared datasets as long as each data table implements the same standard interface, in this case an interface based on the `InstantaneousPoint` class. Though early emphasis has been on procedural automated geoprocessing, more sophisticated scripting will be able to handle tasks such as metadata generation, editing, path generation, and movement modeling.

Finally, the lower right depicts the integrated application, or extension within ArcGIS. Rather than rewrite effective Python in Visual Basic or a similar language, this toolbar instead relies mostly on calls to proven geoprocessing scripts. Unlike the tools, which address Arc Marine object interfaces, the toolbar will directly access the geodatabase as well as available ArcGIS APIs. The most critical function of the toolbar will be the mapping tasks that are not handled effectively by ArcGIS Engine or by geoprocessing scripts, and menu based access to common export formats.

As identified by Rodman and Jackson (2006), currently available Python libraries allow for the eventual development of a standalone Python application in place of the present VB.NET implementation. In particular, the availability of the geoprocessor and DAO in Python allows for the combined use of the customized Arc Marine geodatabase, an external relational database holding non-spatial information, and external DAO read access to the geodatabase, allowing for faster and more complex queries (particularly where clauses) among AnimalEvents and auxiliary data tables. Like in Rodman and Jackson, the MMI project uses WxPython as the primary GUI library and will use this library not only for stand-alone application development but also to develop advanced toolbox scripts and toolbar wizards. The native interface access of WxPython combined with the cross-platform development of Python (there is even a .NET implementation known as IronPython) also means that the entire framework can eventually evolve into a native look and feel cross-platform application. As a final note, three additional Python libraries of importance in tool development are Numpy, Matplotlib, and Makepy/pywin32. The first two libraries allow for the implementation of complex statistical operations, including an implementation of the plotting library of MatLab. Makepy/pywin32 is a utility that provides access to COM objects within Python, creating the potential for access to the ArcObjects COM API from within Python.

Interfaces

In order to maintain the modularity of code, and thus improve collaboration in tool development, programmers should attempt to program to an interface rather than to a specific object implementation. This case study introduces two such interfaces: `InstantaneousPointUI` and `AnimalEventUI`. `AnimalEventUI` is specific to this customization of the data model, while `InstantaneousPointUI` is a concept that should be applicable to any implementation of Arc Marine and provides a solid introduction to the concept of programming to the interface of core classes.

`InstantaneousPointUI` can be developed by examining the attributes of the `InstantaneousPoint` class. Based on its inheritance, this class contains `OBJECTID`, `Shape`, `FeatureID`, `FeatureCode`, `CruiseID`, `TimeValue`, `ZValue`, `SurveyID`, `SeriesID`, and `PointType`. Any `InstantaneousPoint` table can be expected to have these attributes, with all `LocationSeries` points in the table carrying `PointType=4`. Thus, the corresponding interface should implement read access to each of these attributes, and write access to attributes in non-key fields (`TimeValue`, `ZValue`, and the `Shape` geometry). Rather than creating encapsulated code for the class, the read and write behaviors are controlled by editing and update rules on the database. A tool should not, therefore, be written to access these key fields unless the tool itself implements update functions. The tool also should not rely on fields not present in the base

InstantaneousPoint class; only TimeValue, ZValue, and Shape (and indexes as appropriate).

What does this imply for the development environment? While these rules may seem simple and straightforward, all too often custom tools are written to conform to specialized tables. As a result, the user has to go through a series of exports and transformations to even make a dataset usable by the tool. Any tool written to conduct analysis on LocationSeries points should require the InstantaneousPointUI. As a result, the tool will be guaranteed to function with any other LocationSeries point table. Once a research group has imported its animal tracking point observations to a LocationSeries table, no further transformations should be required to use a shared LocationSeries based tool.

For AnimalEventUI, the rules become more complicated. AnimalEvent also has context-dependent joins to sub-dimension tables. The linked sub-dimensions, though, are stored in the EventType field. Thus, with only access to the EventType domain and AnimalEvent, it is possible for a tool to gain access to the full event information through AnimalEventUI. The guaranteed components of AnimalEvent are:

- Index fields (MarineEventID, FeatureID, VehicleID)
- FromLocation (Inherited)
- ToLocation (Inherited)
- DataValue (Inherited)
- TimeValue
- JulianValue
- EventType
- Attribute list of the joined sub-dimension

- Record of the joined sub-dimension

Most AnimalEvent operations will involve either the creation of LocationSeries points from latitude and longitude information stored in sub-dimension tables or dynamic segmentation to assign locations to timestamped events stored in sub-dimension tables. As such, AnimalEventUI only needs to implement access to the index fields (to reach the related linear feature and animal), FromLocation and ToLocation (to dynamically segment the linear feature), TimeValue or JulianValue (for timestamp dynamic segmentation), and the Attribute list to find locational attributes. In order to implement AnimalEventUI, sub-dimension tables must also have the appropriate Lat or Lon prefix on locational attributes (a simple requirement when Service Argos results already include these prefixes on downloaded results). Through this interface implementation, any tool requiring the AnimalEventUI interface will be able to operate on any AnimalEvent table and its associated sub-dimension tables without additional transformation of the data. Thus, an operation as complex as data loading operations on multiple tag types can be executed using a universal shared tool.

Discussion

Enhancing Satellite Telemetry

One of the key research questions outlined in the introduction of this thesis was: “How can a GIS enhance the research advantages of satellite telemetry?” To reiterate, the key advantages are timeliness, relationships to environmental data, autonomous profiling of the animal’s environment, and continuous coverage of the animal’s movements. The above results suggest several key ways in which this customization optimizes Arc Marine and the application development framework to produce a geographic information system that maximizes these advantages.

First, the introduction of automation into back-end data loading greatly increases the timeliness of satellite telemetry data. As processing time narrows from days to minutes and as manual roadblocks are removed from the workflow, key findings can be in the hands of researchers more quickly and with greater relevance to current conditions. Further, the output of these loading processes to a community standard feature class (the LocationSeries subtype of InstantaneousPoint) and the steps of the data loading sequence are modularized. This eases and speeds the transition of a project to new source data forms, or the addition of loading tools for new source data forms outside of Argos such as Fastlock GPS.

Second, the introduction of a standardized data model increases the ability of the researcher to take advantage of the data relationships

between individual animals and their environment as well as spatial and temporal relationships between animals. These relationships are further enhanced by introducing interoperable tools operating on interfaces that allow consistent operation on multiple datasets. Different end user groups in the marine GIS community will formulate their own community customizations of Arc Marine. While this will result in a significantly different appearance from user group to user group and even project to project, ultimately key geometry and critical supporting object classes will be nearly universal from group to group and project to project. The result will be that environmental datasets developed by oceanographers, ecologists, sociologists, resource managers, marine industries, or any other GIS users operating in the marine environment will be transferable to the customized framework of the MMI GIS with minimal transformation and adaptation. In particular, MMI developed tools that operate on core Arc Marine classes in the MMI database schema will operate on these same classes in shared datasets. In turn, shared tools will function on the core classes retained in the MMI customization.

Third, autonomous profiling of the marine environment is directly addressed by one of the key MMI customizations. The addition of the Animal subclass of Vehicle represents a change in modeling viewpoints from the core Arc Marine schema. Following Boehlert et al. (2001), this customization fully realizes the conceptualization of the tagged animal as

its own environmental profiling vehicle. The Vehicle platform allows an Animal to carry an unlimited (at least in the model) array of data collecting instruments across multiple spatial dimensions and time, moving far beyond the singular scalar relationships of LocationSeries points to a Series.

Finally, customization of the data model coupled with a multilevel development platform allows for the full retention of the complex multidimensional data observed in continuous coverage of the animal's movement paths. In particular, Arc Marine and MMI customized Arc Marine contain the essential fact table schemas to construct these data models as data warehouse model. As data warehouses, Arc Marine can be fully utilized for deep analysis and data mining of integrated historical archives of multiple marine data types. This change schema function, though, requires a similar change in mode from the on-line transactional processing (OLTP) typical of operational relational databases to the on-line analytical processing (OLAP) characterized by the vast data warehouses of big business and genetic research.

Defining feature behaviors

For software developers in the marine animal tracking community, one of the central goals will be to create “smart features” which implement complex behaviors through relationships, validation rules, topology, and

extended software code (Zeiler, 1999). These behaviors will be primarily attributed to the LocationSeries points and Track lines which are the spatial representation of marine animals. The behaviors of these points and lines reflect both behaviors attached to the tracking device and to the animal.

These behaviors can be divided into two major classes: locating behaviors and reporting behaviors. Locating behaviors are the methods through which a specific instance of a given animal is assigned an x, y, and z coordinate as well as positional error and a timestamp. Reporting behaviors are methods by which a location series point attaches non-positional data, for example water temperature measured at the tag, to the given animal and a location in time and space. Reporting behaviors are assumed to be the same for all existing tag types even though the types of non-positional data transmitted may vary from project to project.

There are three categories of reporting behavior: decode data stream, assign location to data stream, and assign timestamp to data stream. Data stream decoding is the most fundamental reporting behavior, but also carries the most complex set of rules. This behavior pertains to recorded data, such as dive frequency, incident light, pressure, and temperature, which are linked to a series of tracking positions. Sources can include live satellite transmission, archived satellite transmission, physical download from a tag archive, and direct field observations. As these data are most often received as a binary stream, behavior rules may have to

break this stream down into individual measurements. This type of behavior is not directly handled in the MMI customization of core Arc Marine. Instead, the auxiliary object classes of the tag group allow for storage of the key parameters to parse this stream as part of a back-end loading tool or front-end analysis tool. While this behavior is generally handled by processing software, the outputs of this behavior will need encapsulated object class rules when they are loaded directly into the tables of the geodatabase.

The assign behaviors give an index to these data, either location or timestamp. Most often the initial assign behavior will come in the form of a timestamp collected with the data. The use of this timestamp will vary depending on the specific parameters of the data, and hence assign rules can be linked to data parameters. Subsequently, through this use of this timestamp and existing timestamped LocationSeries points for the same animal, a geographic coordinate can be assigned to the data. This assignment is controlled by rules for the assign location behavior, which are in turn are governed by interpolation methods for animal routes.

Locating behaviors are dependent on the technology of each tag type. Although locating behavior rules will have to be redefined as new technologies are developed, many rules will be reusable from tag type to tag type. The MMI uses positions predominantly from Argos service tags, although some positions (especially tag deployment locations) come from

GPS receivers. Initial software development has focused strongly on the Argos tag type, though cooperative efforts with Wildlife Computing are opening up access to Fastloc GPS tagging raw data that will allow the programming of advance behavior for those tags.

Although locating behavior rules will be more varied than reporting behavior, there are five basic categories of locating behaviors: set latitude, set longitude, set elevation/depth, derive new location, and set quality flag. For the Argos locations, with solution pairs, the first three behaviors are all carried out by creating a LocationSeries point for the probable true solution and the probable mirror solution. The most basic behavior for deriving a new location is to swap the probable true location with the probable mirror location, although many different rules can be generated to control when this swap should happen. Examples of these rules will be discussed later, but it should be noted that while the rules are dictated by the tracking technology, the parameters to these rules are based on the animal being tracked. Lastly, there must be behavioral rules for rejecting both locations. A basic example of such a rule, assuming the animal is a cetacean species, would be excluding a solution pair for which both locations fall on land and outside of beach regions.

The day length-SST tag (Wallace et al., 2005; Weng et al., 2005) provides a different example of these five behaviors. This tag, designed for species that spend considerable time near, but not at, the surface, takes

advantage of measured light intensity to calculate day length, and hence latitude. Longitude can be crudely calculated based on sunrise/sunset, but this longitude calculation is significantly refined by the addition of sea surface temperature (SST) from a sensor in the tag instrument array. Sea surface temperature at the tag is matched with satellite remote sensed measurement of sea surface temperature to create a refined location region for the animal. Thus, for the daylight-SST tag, the assign of latitude and longitude are two separate behaviors with distinctly different data requirements. Meanwhile, the assign longitude behavior through sea surface temperature also acts as a separate behavior for deriving a new location, pointing to the potentials for code reuse even within the same tag type.

Developing these three reporting behaviors and five locating behaviors for each tag type increases the potential for generalized tools for the animal tracking community. When a software developer can rely on the same methods, regardless of tag type, to prepare data input for analysis, the development task is simplified considerably while the cross-compatible of tools between projects is greatly increased.

Defining information services

The MMI schema has three levels of services: back-end or data stream, data warehouse, and front-end or client platform. These services

can respectively be thought of as input, storage, and output, although each service type handles all three of those functions to varying degrees. Only the data warehouse is essential to enterprise application development, whereas the data stream and client platform represent additional capabilities, such as automated tag download or Argos filtering, that take advantage of procedures and parameters stored in the expanded data model.

Data warehouse

The *data warehouse* consists of the objects from the base Arc Marine data model and additional object classes of the animal and telemetry groups. These classes present a multidimensional view of spatial geometry and measured data through which front-end services can conduct detailed analysis. Since the data warehouse classes are guaranteed to be available to any data stream, they are the primary target of any back-end data loading applications. Classes outside of this data warehouse are more project specific and oriented to the development of project specific tools. Meanwhile, front-end analysis tools are guaranteed a multidimensional view presented by the data warehouse rather than the structure of any specific class objects.

Conversely, a data stream back-end or client platform front-end must function with a geodatabase that implements the data warehouse

objects. So, any additional object requirements for a service must be added by that service if not already present in the system. Using Argos filtering as an example, the Argos filtering application expects a table with Argos flags for each LocationSeries point. If these flags are not available, then the Argos filtering client will add this table and create flags for all of the points. The application, though, will not attempt to build a LocationSeries table nor extract out a LocationSeries table from other data in the data warehouse.

Data Stream

A *data stream* is a service that routinely updates the data warehouse from an external source. Or more ideally, the data stream would update a transactional form of the MMI customized schema which then, in turn, would be used to add new data to the analytical, or data warehouse, form of the schema. A data stream is not a one-time data loading program that initially populates the data warehouse with records. Such a one-time use data loader would not have any requirements to store information. A data stream, though, can store server names, login information, downloading parameters, records of download times, and other information useful to automation and coordinating of updating.

An example of a data stream in this case study is the Argos download service. Service Argos can supply results from a telnet

connection to base server that stores recent satellite returns. This server interactively provides text results which can be captured.

These satellite returns can be managed interactively. A researcher telnets to the correct server, enters a name and password, sends a command to request a data in a specific format for a certain program and time period, copies that information into a text file, and manually enters it into a spreadsheet or database. Even the Argos download service can be used interactively. Once a day the service is manually started and given the correct login information, program, and time period; the service then connects to the server, sends the correct command, captures and parses the text and inserts that information into the database. This interactive operation of the service, like a one-time use data loader, does not need to store any parameters in the database, but it does require someone to send the command every day.

In order to become automated, the data stream needs to keep track of the same parameters that the user entered. These parameters could be stored in the program, but placing the storage within the geodatabase allows the program to simply be pointed at the correct database regardless of the type of database, the operating system, the file system, or any other system dependent properties. Even the procedure itself can be stored directly in the database as a set of several automated services. These parameter tables and stored procedures are not part of the core data

warehouse, but instead cumulatively make up the auxiliary data stream level of this customized Arc Marine schema.

Client platform

An interesting analogy for the front-end client platform is the Facebook Platform of facebook.com (Facebook, 2007). The Facebook Platform is a standard used when other software authors create programs to interact with facebook.com. End users of the site first interact with Facebook's version of the data warehouse (technically a transactional database with a significantly different structural optimization from Arc Marine). They upload a photo and enter their name, email, birthday, hometown, schools attended and other information in the base Facebook profile (literally an online version of the classic freshman facebook). This profile information is stored on the Facebook server according to a specific data model.

The Facebook Platform then defines what information from this data model is available to other applications, how this information is accessed, and finally, how these data interact with the information provided by the external application. The external application, though, often needs additional information. Whether this information is survey results, a user's movie ratings, or statistics for a simple game, this information is not stored in facebook.com's data model. This would require adding thousands of

new fields and a constantly expanding database. Instead external applications, or clients, store this new information in a separately maintained database and relate the two information sources through the user's identification. When the user's profile page is constructed, it relies on the data warehouse and then on the external client database to bring in all the necessary information to present the overall output of the profile page.

Low level access

Low level access refers to directly accessing and working with the information in the data warehouse. In other words, using database software (such as Microsoft Access or SQL Server SQL Analyzer) to directly enter input, view data, or send SQL queries to update, select, append, etc. Normally such query tasks are handled through a database abstraction layer. For now, a database abstraction layer can be simply defined as a unified interface to access the features of multiple database software packages. Rather than reprogramming for the specific syntaxes and structures of MySQL, SQL Server, or Oracle, the abstraction layer provides software drivers for each of these systems while a standardized command set runs each driver in a similar manner, allowing for unified code. With the existence of data streams and clients, low level access is restricted down to complex platform-dependent queries (for example, self-

joins, sub-queries with row counts, or correlated queries) that cannot be handled by database abstraction layer. Further, low level access should only be utilized for one-time tasks such as initial data loading, backup, or replication.

Even the data streams and clients should not utilize low level access, as this removes the universality of these components. These services should always utilize a database abstraction layer to remove dependency on the specific physical implementation of the database schema. For programmed applications, only one-time use data loaders should ever take advantage of low level access (as these loaders will often be specifically configured or programmed for a specific project).

The Client Side

The *client platform* consists of tools for extracting output from the data warehouse. Clients retrieve snapshot record sets through queries to the database and use these record sets in reports, statistical analyses, and models. These clients also utilize the database to store complex parameters and outputs, such as binary input bit structures and SQL query strings or final model mesh grids and random walk geometries. An extensive example of client parameter storage in the case study is the series of tables that make up the Tag objects. These objects are related to the core data warehouse classes, but also store additional information on

tag components, bit structures, duty schedules, and other information that can be used to plan deployments or Argos binary data messages. Yet, they are only necessary to the function of client tools that access tag information and are not universal to animal tracking projects. These tables can be constructed on the fly when an appropriate client application is added without any changes to the data or structure already present in the database other than building appropriate table relationships.

OLTP and OLAP

While the front-end and back-end tools present the promise of automated data processing and simple user interfaces to data, the data warehouse aspects of Arc Marine and the MMI customization present the greatest potential for higher level analytic techniques. To reach this level, the marine geodatabase must be brought over from the desktop use of transactional relational databases into the enterprise use of on-line analytical processing.

On-line analytical processing (OLAP) carries a different approach to data than the more traditional database function of on-line transaction processing (OLTP). OLTP reflects an operational relational database (Han and Kamber, 2006). This is like a checkbook register handling day-to-day tasks like entering new purchases and deposits, correcting mistakes, and reconciling with the bank's records. An OLTP system must handle entry,

updating, changes, and all from multiple users making simultaneous changes and accessing quick views of sections of the database. The emphasis is on the transaction, a maximum number in a minimum amount of time. These transactions are of a limited type on a relatively small number of records, but they require speed and currency with read and write access.

In contrast, OLAP emphasizes the analysis over the transaction. An OLAP system is built for a smaller number of users but with more extensive read access. Write transactions are limited to data cleaning and loading, while read transactions take the form of complex queries, including high accuracy consolidation and aggregation over historical data. OLAP is more akin to a library, where the emphasis is on access instead of updates. The library is organized along an indexed detailed classification system (such as the Library of Congress Classification) and uses subject orientation to summarize and aggregate, optimizing for complex information queries that can span a wide range of media or sources. Whereas OLTP focuses on the database client (data entry, clerical, IT staff), OLAP focuses on the database subject (analysis, management, research staff). In short, OLTP provides operational support with data input, OLAP provides decision support with analysis output.

So where does the MMI case study fit into this classification of on-line transactional processing systems and on-line analytical processing

systems? On the surface, the MMI case study has many operational requirements, the most prominent of which is the regular download of Argos satellite returns and the processing of raw data messages from those returns. Add in the specific operational requirements of linking derived results to filtering choices, maintaining a filtering audit trail, recording field data, and storing tag hardware details, and the MMI system starts to look transaction oriented.

However, the primary goals of the MMI system all reflect a strong analysis and informational processing focus, ultimately reflecting the broader scope of decision support for marine managers. The raw data repository carries an emphasis on long-term information requirements; the updating from new satellite returns is purely a data loading function. In turn, the preserved linkage between derived results and processing choices reflects a subject orientation on experiment repetition rather than transactional processing of research methods. For an OLTP orientation, the focus would be instead on preserving the derived snapshots for continuous manipulation and updating. Even the storage of field data and hardware details are an OLAP function to be used to examine historical patterns of data collection and to perform complex analysis (in this case, binary decoding) on the historical records of raw data messages. The direct selection of filters in interactive analysis requires a transaction

emphasis, and that aspect of the system falls outside Arc Marine and within the scope of database meta tables.

Arc Marine as a data repository

Generically, Arc Marine can be referred to as a relational database model (Codd, 1970), though whether a data storage system based on Arc Marine constitutes a relational database would depend greatly on the physical implementation and would be unlikely with modern software under the formal rules specified by Codd (1985a, 1985b). Arc Marine, though, branches off into advanced forms of databases: spatial, spatiotemporal, and object-relational. Ultimately, to meet the goals of this case study as well as widespread community implementation, it might best be implemented as a data warehouse rather than a transactional database.

The foundation of Arc Marine is an entity-relationship (ER) model (see Appendix A). Formulated from the Common Marine Data Types built by the Arc Marine development team (Wright et al., 2007), the Arc Marine ER model is the semantic representation of the database management system that would physically host data based on these Common Marine Data Types. Each entity in the model corresponds to a relation table with a key identifier and set of attributes (as depicted in the ER model) that describe the key, including foreign keys, which represent a description by the records of another relation table. The relation table itself also holds a

set of attribute tuples, the records or rows of the table, each with a unique key value. The relationships in the ER model depict those foreign key links between relation tables. Through the foreign keys, an attribute tuple in one table, such as the descriptive attributes of a specific whale species in the Species table, can be linked to that descriptive attribute in another relation table, the Species foreign key (i.e. species name) for a specific individual animal. This entity-relationship semantic model provides a direct foundation for relational database design (Chen 1976; Teoroy et al., 1986). The advantages of the relational model for databases are discussed further by Codd (1970 and 1982).

Therefore, taken at its core alone, Arc Marine is a relationship database schema, but this schema also includes complex objects beyond simple attribute tuples. In particular, one of Arc Marine's primary purposes is to integrate spatial data, in the form of feature classes (vectors) and mesh grids and volumes (rasters). These spatial data put implementations of Arc Marine into the realm of spatial databases and spatiotemporal databases.

Arc Marine can also act as a model for an object-relational database. Following on the concept of object-oriented databases (Atkinson et al., 1989), object-relational databases consist of object classes and instance objects of those classes. Classes can be thought of as the tables and objects as the tuples of those tables. Indeed, the complex multi-

dimensional modeling representations in Arc Marine might best be represented in an object-relational form (Stonebraker et al., 1990). Under this form, the classes of Arc Marine are represented as constructed object types (e.g., LocationSeries Point, Vehicle, Track, MarineArea) composed of base types (integer, character field, point, polygon) or other constructed types. Inheritance in object-relational databases also allows for easy implementation of class extension. In this case study, Animal is merely ANIMAL (sex=integer, genotype=c40, estlength=float, social=integer) inherits Vehicle, in the language of PostgreSQL (Stonebraker et al., 1990) the Object-Relational Database Management System used by the open-source GIS GRASS. Or to rephrase, an Animal is an object with the same attributes of Vehicle (an instrument carrying platform) with the additional attributes of sex, genotype, estimated length, and social group. The primary advantage to this object-relational implementation is the ability to encapsulate data and code into a single object. Thus, the object class implements behavior as well as descriptions and allows for the implementation of smart objects. The most fundamental disadvantage, though, is that there is no support for PostgreSQL, or a similar ORDBS, in ArcGIS at this time, though such support will be available for ArcSDE in ArcGIS 9.3 (ESRI, 2007c). Construction of the Arc Marine schema for GRASS would require an extensive logical reconstruction to replace the underlying ESRI geodatabase object model (and its corresponding

geodatabase physical implementation) with the PostGIS geodatabase object model. While this would move Arc Marine outside its application-specific constraints, as will be reviewed later, object-relational implementation is more easily achieved through database abstraction and the encapsulation of complex object behaviors in scripting code.

There is a special consideration for ArcSDE when using Arc Marine in an OLAP role. ArcSDE has a transaction-optimized query system specifically geared towards on-line transactional processing. Because of ArcSDE's enterprise role in a geographic information system, the software is geared towards the handling the series of editing transactions that build up to continuous update of a spatial dataset. As a result, ArcSDE requires the underlying database structure to be optimized for OLTP and not data warehousing (particularly important when building ArcSDE on an Oracle database), resulting in reduced efficiency and accuracy of complex queries (particularly aggregation across large tables). Despite the attractiveness of the "enterprise" tag, be aware that enterprise relational database software is built towards operational processing and may not be the best solution for scientific analysis (ESRI, 2007a).

Arc Marine's OLAP role in this case study suggests a different data repository role for the data model. Arc Marine can serve as the unifying schema for a data warehouse, as mentioned early in the discussion of data streams and the client platform. In this study, the data warehouse

encompasses the consolidated multiple source data of the Marine Mammal Institute. With the generalized schema of the case study, this data warehouse can be scaled up to multiple projects covering other species, tracking methods, and auxiliary data types. This data unification role would cross Arc Marine over from a geodatabase schema into a spatial data warehouse.

Conceptual Multidimensional Model of Arc Marine

A common conceptual model for the numeric measures accessed by OLAP tools is the data cube (Gray et al. 1997). This view developed out of the pivot-table view of front-end spreadsheet software such as Microsoft Excel (Chaudhuri and Dayal, 1997), and is commonly used to develop OLAP and data warehousing systems (Li and Wang, 1996; Harvel et al., 2004; Jensen et al., 2004; Li et al., 2004; Miller 2007). The base of this model is the atomic numeric measures that are the target of analysis. These numeric measures carry a set of dimensions, the context of the measures. A classic example comes from Chaudhuri and Dayal (1997):

For example, the dimensions associated with a sale amount can be the city, product name, and the date when the sale was made. The dimensions together are assumed to *uniquely* determine the measure. Thus, the multidimensional data views a measure as a value in the multidimensional space of dimensions. Each dimension is described by a set of attributes. For example, the Product dimension may consist of four attributes: the category and the industry of the product, year of its introduction, and the average profit margin. For example, the soda Surge belongs to the category

beverage and the food industry, was introduced in 1996, and may have an average profit margin of 80%.

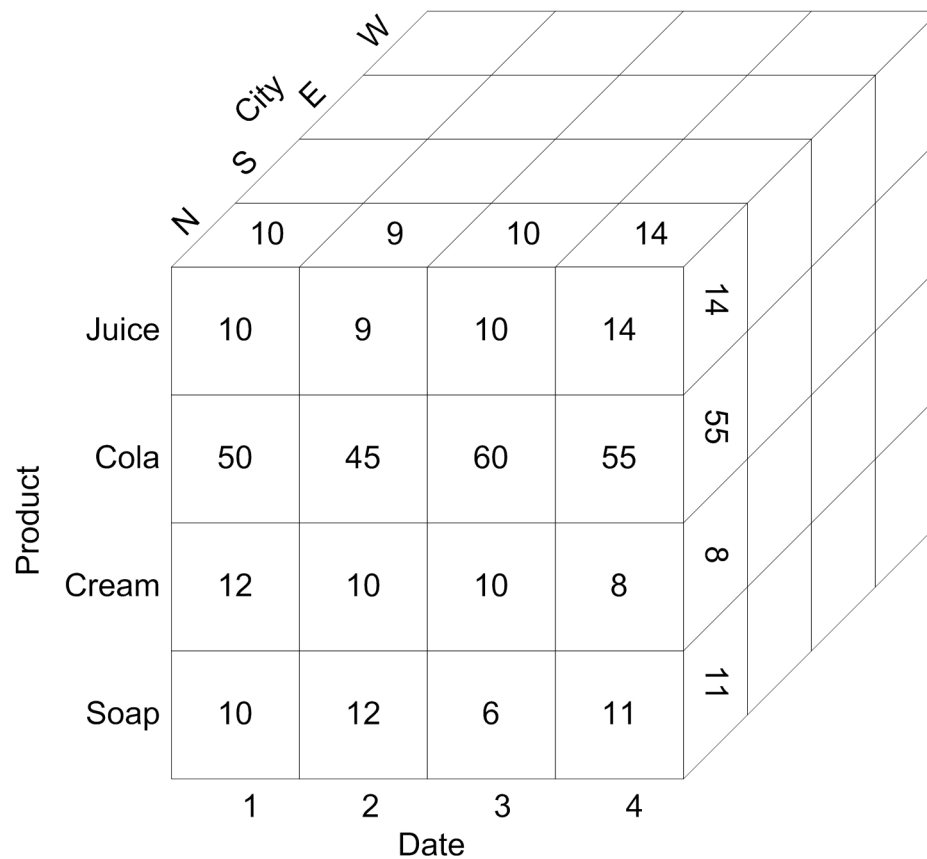


Figure 5. Multidimensional data cube.
The atomic measure Sales has three dimensions: Date, Product, and City (Chaudhuri and Dayal, 1997).

Dimensions may also carry a concept hierarchy. In a concept hierarchy, each node represents a level of abstraction, arranged from specialized to generalized. A concept hierarchy may be rolled up (generalized) or drilled down (specialized) to create different data views for data exploration (Miller 2007). As an example, a time dimension can have a hierarchy of “day < {month < quarter; week} < year” (Figure 6).

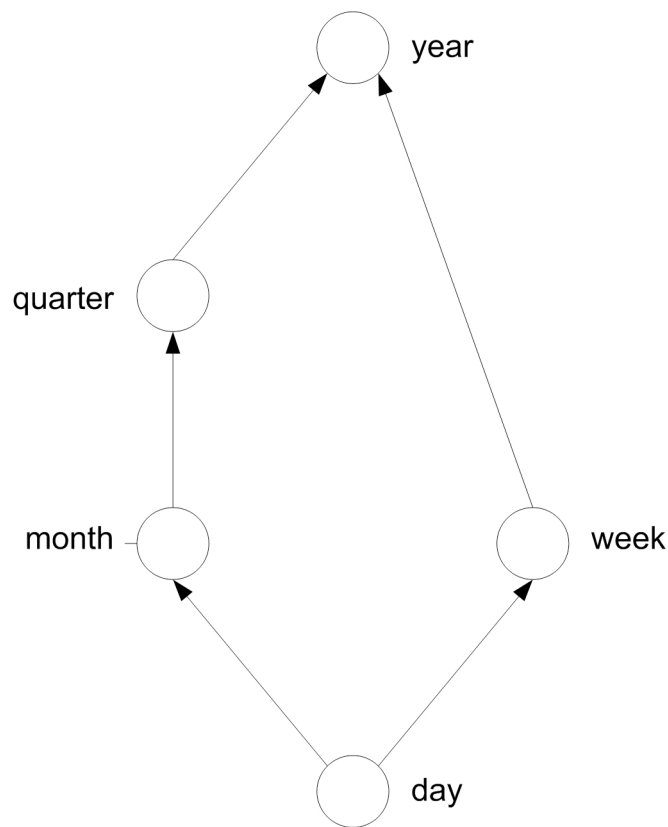


Figure 6. Concept hierarchy for the Time dimension.
 Day can be rolled up day < month < quarter < year or day < week < year.
 Drill down operations proceed year > {month > quarter; week} > day (Han and Kamber 2006).

The Arc Marine Data Warehouse Design Schema

What follows is a description of the data warehouse design schema for Arc Marine as a multidimensional model. While this schema will support the use of geographic data mining, the data mining techniques themselves fall outside the scope of this study. For a more extensive discussion of spatial data mining, the use of OLAP tools in geographic knowledge discovery, and an overview of spatial data mining techniques see Miller

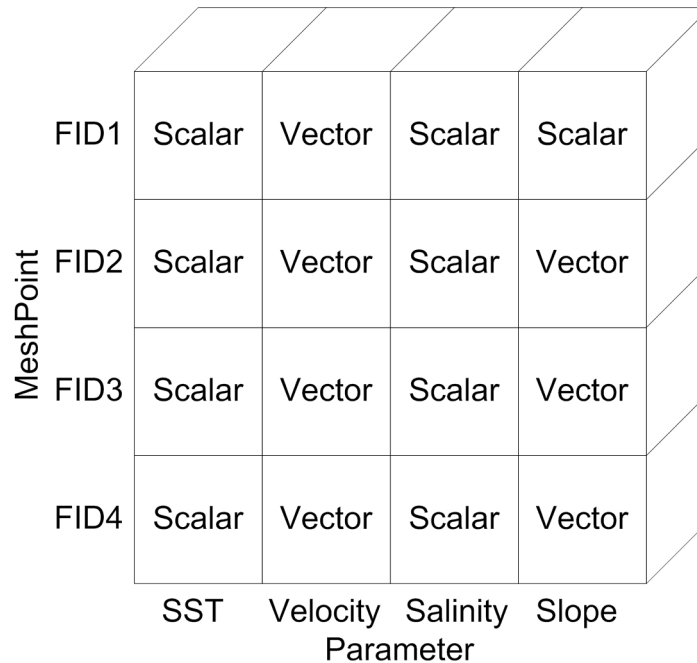
(2007); Miller and Han (2001) Chapters 1, 3, and 4; and Han and Kamber (2006) Chapters 3, 4, and 10.

To begin with, there are three common data warehouse design schemas: star, snowflake, and fact constellation (Han and Kamber 2006). A *star schema* is the most common form and is characterized by a normalized central fact table containing the atomic measure and dimension table keys, and a set of denormalized dimension tables. The structure is termed a star because of the schema graph is typically displayed with the dimension tables in a radial pattern around the fact table. The *snowflake schema* differs from the star schema in that the dimension tables are normalized into further sub-dimension tables. This normalization reduces redundancy, saves space, and is easier to maintain. The tradeoff though is a greater number of joins in query execution. When the dimension tables are small relative to the fact table (the most common case), the advantages of normalization are minimal. The *fact constellation* is a schema containing multiple fact tables sharing dimension tables. As an example, a data warehouse containing historical fact tables for shipping, inventory, and sales would share location and product dimension tables between the three fact tables. This schema can be thought of as a collection of star schema, hence the terminology fact constellation.

The mesh features of Arc Marine present a solid example of a data cube within the Arc Marine data model. The base of this star schema is

also the atomic measure of the mesh feature, the Scalar or Vector Quantity. Note that the X, Y, and Z components of a vector quantity are attributes of the fact table, not separate dimensions; aggregation by individual vector components is not likely to be a useful operation. The two types of quantities can either be thought of as a single base fact table, or more accurately two fact tables of a fact constellation which share an identical set of dimension tables. One of the dimensions of these fact tables is MeshPoint and the other is Parameter. Each of these represents a different common form of aggregation for scalar and vector quantities: aggregating by the same location and aggregating by the same measurement type. The Mesh itself is not a dimension. Instead, it is part of the concept hierarchy for the MeshPoint. Though not present in the Arc Marine schema, this hierarchy can be generalized further from mesh point to mesh to catalog (an assembly of meshes covering a specific area and time interval). Additional hierarchies can be added including regions and time periods. Parameters, as well, can be grouped into higher hierarchies of common parameter types. This schema is technically a snowflake schema as MeshPoint is actually a normalized table with dimension Mesh. It would be possible to create a denormalized view by combining the tables. In practice, a Mesh is often a composite structure (such as a raster or grid) containing Mesh Points and quantities without their expression as

separate fact tables, thus representing such a denormalized view of the multidimensional data cube, or square in this case (Figure 7).



FID1	Scalar	Vector	Scalar	Scalar
FID2	Scalar	Vector	Scalar	Vector
FID3	Scalar	Vector	Scalar	Vector
FID4	Scalar	Vector	Scalar	Vector
	SST	Velocity	Salinity	Slope
	Parameter			

Figure 7. Multidimensional model of the Arc Marine MeshFeature class.

For this case study, LocationSeries Point and MeasuredData are the central fact tables, with LocationSeries point serving as part of the concept hierarchy for the Measurement dimension of Measured Data.

MeasuredData represents the atomic measure within the schema, while features, in this case LocationSeries Point serve as part of the concept hierarchy for Measurement. LocationSeries (as well as other features) is also an atomic measure of its own fact tree (sharing a fact constellation with other feature classes) when it is used for purely analyzing spatial distribution or movement. A relevant example should help clarify this.

Surfacings are a quantity commonly measured by satellite telemetry tags on whales. In this simple example, surfacings are just a count of the number of times the animal reaches the surface (there are other ways to measure this metric). A data view of animal surfacings focuses on measured data from the tags and may or may not have a spatial component. When a spatial component is used, it merely represents an aggregating spatial area for a count of surfacings. That count is still contained within the measured data themselves, and would rely on a data view based on the MeasuredData star. Meanwhile, a kernel density or home range analysis relies only on animal locations and no elements of measured data. There may be dimensions to the animal locations (animal, species, location quality), but the atomic measure used in the analytic calculations is the point feature. Hence, the fact table for kernel density would be the LocationSeries Point table and the data view would be based on that table's star schema.

The MeasuredData Star

MeasuredData is the fact table of star schema represented by a three dimensional data cube of Measurement, MeasuringDevice, and Parameter (Figure 8). These dimensions alone do not present interesting levels of aggregation, but the concept hierarchies for MeasuringDevice and

Measurement introduce significant analytical aggregations, while Parameter defines data of common types. The

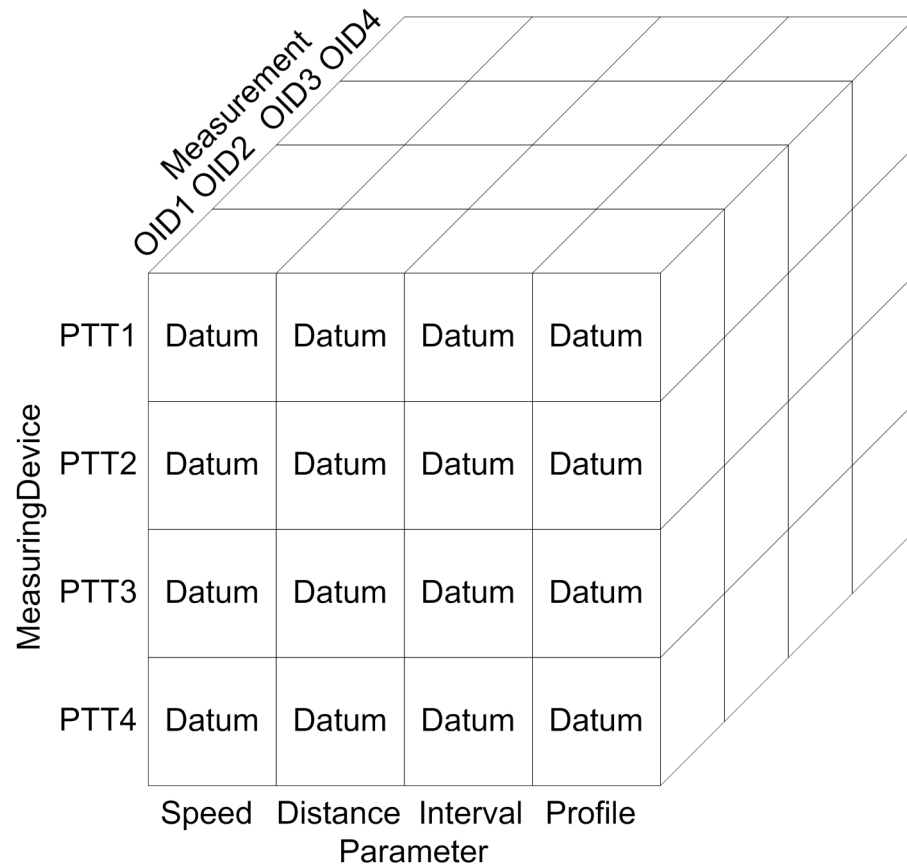


Figure 8. MeasuredData data cube model in core Arc Marine.

MeasuringDevice hierarchy roles up from MeasuringDevice to Vehicle to the feature class Track. Meanwhile the Measurement concept hierarchy rolls up to multiple parallels. One of these parallels is the TimeSeries object class which subsequently rolls up to the MarineFeature classes. The rest of these parallel hierarchies are the feature classes themselves which embody spatial and temporal quantities as well as rolling up to higher

aggregations such as surveys, cruises, and series. In the context of the MMI customization, a fourth dimension is added in the form of AnimalEvent (Figure 9). The MeasuringDevice concept hierarchy develops a more significant aggregation by substituting Animal (and hence Species and higher levels of Animal) for the Vehicle concept level. AnimalEvent not only allows another route to aggregation by Animal or feature classes, it also opens up a route to aggregation by the wide array of context-dependent sub-dimensions. Denormalization relative to specific sub-dimensions can create multiple sub-cubes by tag type that will allow the relation of MeasuredData all the way back to raw data messages.

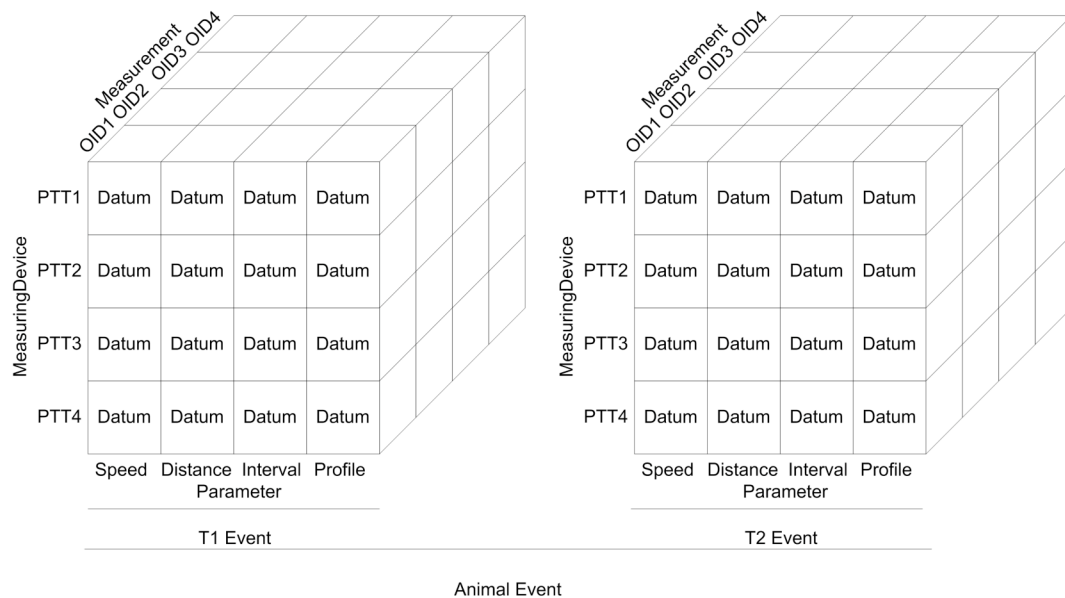


Figure 9. MeasuredData data cube in the MMI customization. Note that this is a four-dimensional cube, with the fourth dimension, AnimalEvent, represented as multiple cubes.

One extremely important aspect of the spatial data warehouse is the ability to aggregate spatially. Any concept hierarchy which can roll up to a feature class can further roll up to spatial generalizations and even take on additional dimensions from spatial joins with environmental rasters as defined in spatial data mining techniques (See Miller 2007 for an extensive discussion of spatial OLAP operations).

The LocationSeries Point Star

This discussion of the MMI customization of Arc Marine as a spatial data warehouse closes with an exploration of the spatially oriented LocationSeries point star. In the MMI customization, LocationSeries point carries only two object class dimensions, Animal Event and Animal (as linked through Series). The implications of each of these classes in a concept hierarchy have been discussed above. It should be particularly noted that the multidimensional cube of this fact star can be drilled down, for example from Animal to MeasuringDevice to MeasuredData. Yet, LocationSeries point, as a spatial feature class, carries an additional spatial dimension as defined by its spatial geometry. The concept hierarchy of geographic space has potential levels limited only by the grain and extent of the dataset. As mentioned above, this spatial dimension also contains spatial joins to other spatial datasets introduced into the data warehouse. As a result, the potential for data mining expands out to any form of marine

data linked to Arc Marine, and with it, the potential to more deeply explore the central question of the MMI program, what is the relationship between physical and biological process and the distribution and movement of endangered whale species.

Database Abstraction

As a final special topic, one of the largest barriers to creating a cross-platform spatial data warehousing solution is the physical implementation, or adapting the geodatabase schema to the specific hardware, operating system, and database software. The syntax for a query in Microsoft Access or SQL Server will differ from the syntax for a query to MySQL, Oracle, or PostgreSQL. And even the same software platform can differ when run on Windows, UNIX, Linux, or Max OS.

The Open Database Connectivity (ODBC) specification is one such abstraction layer in which software specific drivers conform to a defined application programming interface (API). Rather than having to create a new code for every software package, a programmer just has to match the requirements of the API. The API compliant drivers then transform the programmer's database requests into the appropriate syntax for that software.

The ESRI Geodatabase is a similar abstraction layer. The implementation of the geodatabase will vary depending on the underlying

GIS software (an SDE database is very different from a file geodatabase or personal geodatabase), the operating system, the file system, and the database file structure (such as MS Access or Oracle). Yet, a user accessing the geodatabase through ArcGIS can expect analysis tools, editing operations, and most of all mapping to work almost identically regardless of the physical implementation. Even a programmer using a scripting language or ArcObjects to access the geodatabase (but not the underlying database) has a defined model (an API) to follow so that the code functions identically on any combination of hardware and software running ArcGIS.

For the Python scripting language used by the ArcGIS 9.2 software environment, access to the underlying database comes through a database abstraction API known as DB-API 2.0. This Python specific database abstraction layer is based upon a series of independently written modules that can connect to a wide range of database types while using only one program syntax. With the cross-platform compatibility of Python (as well as other advantages previously explained), the use of DB-API 2.0 allows for cross-platform low level database access as well as higher level access to the geodatabase through the ArcGIS geoprocessing object.

DB-API 2.0 based code can also read and update a spatial database independent of ESRI software. This means an ESRI geodatabase can be directly updated without an active ArcInfo license. A

spatial database implemented in open source PostGIS can be accessed in the same form with the same code. Therefore, if the Arc Marine schema is implemented in open source GIS software such as GRASS or MapServer, Python code written for Arc Marine with the DB-API 2.0 database abstraction layer will be able to access the implementation without software changes. Going beyond current software, any future implementation of the Open Geospatial Consortium's Simple Features standard will only require a compliant DB-API 2.0 module in order to be used with the same code base.

Conclusion

Through the course of this case study in Arc Marine customization, two key concepts emerged that can help guide the future development of Arc Marine for animal tracking and as an enterprise on-line analytical processing structure.

First, Arc Marine takes advantage of the multidimensionality of the geodatabase object model. The MMI customization pushes that multidimensionality outwards to add more dimensions (such as time through AnimalEvent) and broader levels of hierarchy concepts, rolling up from timestamped data acquisition events to aggregated spatial regions. This multidimensional view of marine data opens a pathway to the implementation of high level analytical tools including data warehouses, OLAP techniques, data mining, and spatial data mining.

Second, Arc Marine creates an expandable platform to drive community application development. By defining a tracking community framework with the MMI customization, researchers and programmers from different projects can develop compatible tools and share compatible datasets. This will speed data extraction from online repositories such as OBIS-SEAMAP. The additions of cross-platform Python scripting and database abstraction will help separate physical implementation decisions from processing tool choices. Tools developed for the back-end framework will facilitate data loading and ease the transition to Arc Marine. Tools

developed for the front-end will open up more powerful analytical techniques and make the adoption of Arc Marine more attractive across the marine animal tracking community.

This leads to a reevaluation of the six goals of Arc Marine in the context of the animal tracking community and of the Marine Mammal Institute customization of Arc Marine.

1) Create a common model for assembling, managing, and publishing tracking sets, following industry-standard methods for dissemination (such as XML and UML). Methods and mechanisms for metadata dissemination were not explored. Despite this, the back-end framework creates a standardized process for the transfer of datasets (see Appendix C for an example of an importation tool from OBIS-SEAMAP published data). By attaching to this data transfer process, the metadata transfer process can be similarly automated and standardized.

2) Produce, share, and exchange these tracking data in a similar format and following a standard structure design. The case study demonstrates the flexibility of the LocationSeries subtype of InstantaneousPoint in handling a wide array of data types and tag types. As import/export tools develop, the LocationSeries feature class can evolve into a system for transfer between geodatabases as well as an archival form for data warehouses.

3) Provide a unified approach that encourages development teams to extend and improve ArcGIS for marine applications. Arc Marine's rigorous yet general coverage of marine data types and related object classes proved to be extremely useful in defining a larger application framework. In particular, the generalized architecture provides a modeling syntax that can be readily adapted to specific project questions. The feature class and class object standards act as a de facto API for programmers in the research field, minimizing the amount of community-wide redundancy in application development.

4) Extend the power of marine geospatial analysis by providing a framework for incorporating object-oriented rules and behaviors into data composed of animal instance locations and dealing more effectively with scale dependencies. While data exchange was facilitated by the back-end structure, analytical power is increased by the front-end structure. The easy transformation of Arc Marine into multidimensional views unlocks higher analysis power. One of the most significant of these powers is the concept hierarchy that allows aggregation and summarization to move fluidly between different concept scales, including physical scales. The role of object-oriented rules still needs further exploration, but the exposure of the Arc Marine object classes through programming interfaces is an essential element for meeting this particular goal.

5) Provide a mechanism for the implementation of data content standards, such as the OBIS schema extension of the Darwin Core Version 2 (OBIS, 2005). As demonstrated in Appendix C, the standardized structure of Arc Marine feature classes can be directly translated from a standardized data content standard. In effect, this provides a reading mechanism from standardized content into geodatabase storage and direct display in ArcMap. What is left is to fill the translation gap with other standards.

6) Aid researchers in a fuller understanding of object-oriented geographic information systems, so that they may transition to powerful data structures such as geographic networks, regions, and geodatabase relationships within an easily managed context. Perhaps the most significant finding of this case study is that it is possible to build powerful data models on top of the generic geodatabase data model, and still present a level of abstraction between the end user and those data structures. Even though the structure underneath may be multi-dimensional with branching hierarchies, complex joins, and multiple levels of processing interfaces, the end user can ultimately manipulate this structure through a short Model Builder model, simple Python script, quick ArcMap view, or even an Excel spreadsheet. Powerful modeling concepts take form in similarly powerful UML visualization tools accessed through a

Visio viewer or web browsers. Real-world phenomena can be represented as defined objects with expected behaviors and descriptive attributes.

In this MMI customization case study, Arc Marine has provided a vital link to match a series of XY locations to a broader understanding of the relationship of those points to their spatial context, the animal, tagging hardware, locating methods, and the wider array of marine data. It has opened these points up to new avenues for the programmer, the data manager, and the analytical researcher. While Arc Marine has shown to be effective to vary degrees in each of these goals, perhaps in this case study it has been the most successful in connecting the marine mammal tracking research field to the broadest powers of GIS and geospatial analysis.

Finally, this study closes with the original research questions posed to it:

How can the Arc Marine Data Model be customized to best meet the research objectives of the OSU MMI and the marine animal tracking community?

How can a GIS implementation enhance the key advantages of satellite telemetry?

In a marine environment with dynamic environmental conditions across a three-dimensional space, what is the optimal application framework to allow multi-level access from multiple users?

This study is an attempt at a systematic examination of the methods by which the OSU MMI can push closer towards linking physical and biological processes to the distribution and movement of endangered whale species across the many scales of their range. The underlying goal has been to harness the timeliness, continuous coverage, environmental relationships and autonomous profiling of satellite telemetry.

New definitions of programmatic and data management frameworks, from loading to warehousing to analysis, will provide the structure for a high speed and accurate automated workflow from satellite download to deep end user analysis. The encompassing object definitions of the core Arc Marine classes provides a standardizing framework, pointing towards common paths of import and export between research initiatives that will be able to publish and subsequently share faster than ever before. Finally, a fully-developed multidimensional framework allows for the development of analytical tools across a limitless range of environmental variables and into the narrowest and broadest scales of the spatial concept hierarchy crossed by tracks of these critical species.

As automation speeds data acquisition and analysis, the resource manager will have access to more timely decision support. With standardization, that same manager will compare across individuals, populations, species, communities, and regions to delineate critical resources and critical habitats. As the development community and the

community standard matures, so too will the analytical and visualization tools, opening up new levels of communication and understanding, not only for the resource manager but also for the public served by that manager. Arc Marine will expanded the role of automation, integration, and communication in the marine resource management dialogue. Through community support, the Arc Marine data model can transform the marine mammal community and ultimately impact the overarching goal of the MMI to push the edge of “best science” and ensure the future survival and success of endangered whale populations.

References Cited

- Aaby, A. A. 2004. *Testing the ArcGIS Marine Data Model: Using Spatial Information to Examine Habitat Utilization Patterns of Reef Fish along the West Coast of Hawaii*. M.S. Thesis. Oregon State University, Corvallis, OR.
- Andrews, B. and S. Ackerman. 2005. Geologic sea-floor mapping: Marine Data Model case study. *Proceedings of the 25th Annual ESRI International User Conference*. San Diego, CA: ESRI.
- Argos. 1990. *User's manual*. Service Argos, Inc.: Landover, MA.
- Atkinson, M., F. Banchilhon, D. Dewitt, K. Dittrich, D. Maier, and S. Zdonik. 1989. The object-oriented database system manifesto. *Proceedings of the Deductive and Object Oriented Database Conference*: 223-240.
- Austin, D., J. I. McMillan, and W. D. Bowen. 2003. A three-stage algorithm for filtering erroneous Argos satellite locations. *Marine Mammal Science*. 19: 371-383.
- Block, B. A. 2005. Physiological ecology in the 21st century: Advancements in biologging science. *Integrative and Comparative Biology* 45: 305-320.
- Block, B. A., H. Dewar, C. Farwell, and E. D. Prince. 1998. A new satellite technology for tracking the movements of Atlantic bluefin tuna. *Proceedings of the National Academy of Sciences USA* 95: 9384-9389.
- Block, B. A., Teo, S. L. H., Walli, A., Boustany, A., Stokesbury, M. J. W., Farwell, C. J., Weng, K. C., Dewar, H. and Williams, T. D. (2005). Electronic tagging and population structure of Atlantic bluefin tuna. *Nature* 434,1121 -1127.
- Boehlert, G. W., D. P. Costa, D. E. Crocker, P. Green, T. O'Brien, S. Levitus, and B. J. Le Boeuf. 2001. Autonomous pinnipeds environmental samplers: Using instrumented animals as oceanographic data collectors. *Journal of Atmospheric and Oceanic Technology* 18: 1882-1893.
- Boustany, A. M., S. F. Davis, P. Pyle, S. D. Anderson, B. J. Le Boeuf, and B. A. Block. 2002. Expanded niche for white sharks. *Science* 415: 35-36.
- Breman, J., D. J. Wright, and P. N. Halpin. 2002. The inception of the ArcGIS Marine Data Model in *Marine Geography: GIS for the Oceans and Seas* (ed. Breman, J.) 3-9. ESRI Press: Redlands, CA.

Chaudhuri, S. and D. Umeshwar. 1997. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*. 26(2): 65-74.

Chen, P. P. 1976. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*. 1(1): 9-36.

Codd, E. F. 1970. A relational model of data for large shared data banks. *Communications of the ACM*. 13(6): 377-387.

Codd, E. F. 1982. Relational database: a practical foundation for productivity. *Communications of the ACM*. 25(2): 109-117.

Codd, E. F. 1985a. Is your DBMS really relational? *Computer World*. 10/14/1985.

Codd, E. F. 1985b. Does your DBMS run by the rules? *Computer World*. 10/21/1985.

Committee on Ocean Policy. 2004. U.S. Ocean Action Plan. ONLINE. Available: <http://ocean.ceq.gov/actionplan.pdf>, 5/23/2007

Dewar, H., M. Domeier, and N. Nasby-Lucas (2004) Insights into young of the year white shark, *Carcharodon carcharias*, behavior in the Southern California Bight. *Environmental Biology of Fishes* 70, 133-143.

Duke University Marine Laboratory. 2004. Duke North Atlantic Harbor Porpoise Tracking. ONLINE. OBIS-SEAMAP. Available: http://seamap.env.duke.edu/datasets/detail_test/83, 10/9/2007.

Environmental Systems Research Institute (ESRI). 1999. ArcInfo 8: A new GIS for the new millennium. ONLINE. Available: http://www.esri.com/news/arcnews/summer99articles/ai8special/ai8_a_new_gis.html, 10/5/2007.

ESRI. 2000a. ArcGIS Water Facilities Model. ONLINE. Available: <http://www.esri.com/news/arcnews/fall00articles/arcgis-wfm.html>, 10/5/2007.

ESRI. 2000b. ESRI develops industry data models. ONLINE. Available: <http://www.esri.com/news/arcnews/fall00articles/esridevelops.html>, 10/5/2007.

ESRI. 2003. HowTo: Visio 2003 Professional UML to XML export facility installation. ONLINE. Available: <http://support.esri.com/index.cfm?fa=knowledgebase.techarticles.articleShow&d=26105>, 5/23/2007.

ESRI. 2007a. ArcGIS Desktop Help 9.2 – About updating statistics. ONLINE. Available: http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=About_updating_statistics, 9/30/2007.

ESRI. 2007b. Data Models. ONLINE. Available: <http://www.esri.com/software/arcgis/geodatabase/about/data-models.html>, 10/5/2007.

ESRI. 2007c. ESRI Developer Summit 2007 – Questions & Answers. ONLINE. Available: <http://events2.esri.com/uc/QandA/index.cfm?ConferenceID=3B67AFC7-D566-ED85-A18E8EFF9B63B57B>, 10/7/2007.

Facebook. 2007. Facebook Developers. ONLINE. Available: <http://developers.facebook.com/>, 9/30/2007.

Gray, J., S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. . 1: 29-53.

Halpin, P., B. Best, E. Fujioka, and M. Coyne. 2004. Marine animal analysis applications. *Proceedings of the 24th Annual ESRI International User Conference*. San Diego, CA: ESRI.

Han, J. and M. Kamber. 2006. *Data Mining: Concepts and Techniques*. Elsevier, Inc.: San Francisco, CA.

Harvel, L., L. Liu, G. D. Abowd, Y. X. Lim, C. Scheibe¹, and C. Chatham. 2004. Context Cube: Flexible and Effective Manipulation of Sensed Context Data. *Pervasive Computing: Lecture notes in computer science*. 3001:51-68.

Hill, R. D. 1994. Theory of geolocation by light levels in *Elephant Seals: Population Ecology, Behavior, and Physiology* (eds. Le Boeuf, B. J. & Laws, R. M.) 227-236. University of California Press: Berkeley, CA.

Kinzel, M. R. 2002. Green sea turtles migration in the Gulf of Mexico: Satellite telemetry and GIS in *Marine Geography: GIS for the Oceans and Seas* (ed. Breman, J.) 25-33. ESRI Press: Redlands, CA.

Lagerquist, B. A., K. M. Stafford, and B. R. Mate. 2000. Dive characteristics of satellite-monitored blue whales (*Balaenoptera musculus*) off the Central California Coast. *Marine Mammal Science*. 16(2): 375-391.

Lassoued, Y. 2007. Arc Marine Ontology in GML/XML Schema. ONLINE. Available: <http://cmrc.ucc.ie/ontologies/interrisk/doc/index.html>, 10/5/2007.

Le Boeuf, B. J., D. E. Crocker, J. Grayson, J. Gedamke, P. M. Webb, S. B. Blackwell, and D. P. Costa. 2000. Respiration and heart rate at the surface between dives in Northern Elephant Seals. *Journal of Experimental Biology* 203: 3265-3274.

Li, C. and X. S. Wang. 1996. A data model for supporting on-line analytical processing. *Proceedings of the fifth international conference on Information and knowledge management table of contents*. Rockville, Maryland: ACM, 81-88.

Li, R., 2000. Data models for marine and coastal geographic information systems in *Marine and Coastal Geographical Information Systems* (eds. Wright, D. J. and Bartlett, D. J.) 25-36. Taylor & Francis: London.

Li, X., J. Han, and H. Gonzalez. 2004. High-dimensional OLAP: A minimal cubing approach. *Proceedings of the 30th VLDB Conference*. Toronto, Canada: Very Large Data Base Endowment, 528-539.

Liaubet, R. and J. Malardé. 2003. Argos Location Calculation. *Proceedings of the Argos Animal Tracking Symposium*. Annapolis, MD.

Mate, B., R. Mesecar, and B. Lagerquist. 2007. The evolution of satellite-monitored radio tags for large whales: One laboratory's experience. *Deep Sea Research II: Topical Studies in Oceanography*. 54(3-4): 224-247.

Mate, B. R. 1989. Watching habits and habitats from Earth satellites. *Oceanus*. 32:14-18.

Mate, B. R., K.M. Stafford, R. Nawojchik, and J. L. Dunn. 1994. Movements and dive behavior of a satellite-monitored Atlantic white-sided dolphin (*Lagenorhynchus acutus*) in the Gulf of Maine. *Marine Mammal Science*. 10: 116-121.

Microsoft Corporation. 2003. Visio 2003 UML To XMI Export. ONLINE. Available:

<http://www.microsoft.com/downloads/details.aspx?familyid=3DD3F3BE-656D-4830-A868-D0044406F57D&displaylang=en>, 2/18/2006.

Miller, H. J. 2007. Geographic data mining and knowledge discovery in *Handbook of Geographic Information Science* (eds. Wilson, J. P. and A. S. Fotheringham). Blackwell Publishing: Malden, MA.

Miller, H. J. and J. Han. 2001. *Geographic Data Mining and Knowledge Discovery*. Taylor and Francis: London, 74-109.

Mote Marine Laboratory. 2007. Casey Key Loggerheads – 2005-2006. ONLINE. OBIS-SEAMAP. Available: <http://www.mote.org/>, 10/9/2007.

Ocean Biogeographic Information System (OBIS). 2005. Ocean Biogeographic Information System. ONLINE. Available: <http://iobis.org/faq/>, 2/19/2006.

Read, A. J. and Westgate, A. J. 1997. Monitoring the movements of harbour porpoises (*Phocoena phocoena*) with satellite telemetry. *Marine Biology*. 130: 315-322.

Read, A. J., Halpin, P. N., Crowder, L. B., Best, B. D., Fujioka, E. (Editors). 2006. OBIS-SEAMAP (Spatial Ecological Analysis of Megavertebrate Populations): mapping marine mammals, birds and turtles. ONLINE. <http://seamap.env.duke.edu>, 2/18/2006.

Rehm, E. 2007. ArcGIS Marine Data Model, Example #1: Simple XML using MDM XML Schema. ONLINE. Available: <http://staff.washington.edu/erehm/mdm/mdm.html>, 10/5/2007

Rodman, L. C. and J. Jackson. 2006. "Creating Standalone Spatially-Enabled Python Applications Using the ArcGIS Geoprocessor," *Proceedings of the Twenty-Sixth Annual ESRI User Conference*, San Diego, CA, August 2006.

Shaffer, S. A., T. Yann, J. A. Awkerman, R. W. Henry, S. L. H. Teo, D. J. Anderson, D. A. Croll, B. A. Block, and D. P. Costa. 2005. Comparison of light and SST-based geolocation with satellite telemetry in free-ranging albatrosses. *Marine Biology* 147: 833-843.

Sherman, L. 2006. Tracking the Great Whales. *Terra*. 1(2):2-8.

Stonebraker, M., L. A. Rowe, and M. Hirohama, The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*. 2(1): 125-142.

Tagging of Pacific Pelagics. 2006. ONLINE. Available: <http://www.toppcensus.org/web/Background/Overview.aspx>, 2/18/2006.

Teo, S. L. H., A. Boustany, S. Blackwell, A. Walli, K. C. Weng, and B. A. Block. 2004. Validation of geolocation estimates based on light level and sea surface temperature from electronic tags. *Marine Ecology Progress Series* 283: 81-98.

Teoroy, T. J., D. Yang, and J. P. Fry. 1986. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*. 18(2): 197-222.

U.S. House, 110th Congress, 1st Session. 2007a. *H. R. 1006, Marine Mammal Assistance*. ONLINE. GPO Access. Available: <http://www.gpoaccess.gov/bills/index.html>, 4/04/2006.

U.S. House, 110th Congress, 1st Session. 2007b. *H. R. 2327, To amend the Marine Mammal Protection Act of 1972 to strengthen polar bear conservation efforts, and for other purposes*. ONLINE. GPO Access. Available: <http://www.gpoaccess.gov/bills/index.html>, 5/23/2006.

U.S. House, 110th Congress, 1st Session. 2007c. *H. R. 250, National Oceanic and Atmospheric Administration Act*. ONLINE. GPO Access. Available: <http://www.gpoaccess.gov/bills/index.html>, 4/04/2006.

Wallace, B. P., C. L. Williams, F. V. Paladino, S. J. Morreale, R. T. Lindstrom, and J. R. Spotila. 2005. Bioenergetics and diving activity of interesting leatherback turtles *Dermochelys coriacea* at Parque Nacional Marino Las Baulas, Costa Rica. *Journal of Experimental Biology* 208: 3873-3884.

Welch, D. W. and J. P. Everson. 1999. An assessment of light-based geoposition estimates from archival tags. *Canadian Journal of Fisheries and Aquatic Sciences*. 56: 1317-1327.

Weng, K. C., P. C. Castilho, J. M. Morrisette, A. M. Landeira-Fernandez, D. B. Holts, R. J. Schallert, K. J. Goldman, and B. A. Block. 2005. Satellite tagging and cardiac physiology reveal niche expansion in salmon sharks. *Science* 310:104-106.

Wright, D. J. 2007. UML diagrams, Case studies of Arc Marine: The ArcGIS marine data model. ONLINE. Available: <http://dusk.geo.orst.edu/djl/arcgis/diag.html>, 10/5/2007.

Wright, D. J., M. J. Blongewicz, P. N. Halpin, and J. Breman. 2005. A new object-oriented data model for coasts and oceans. *Proceedings of CoastGIS 2005, 6th International Symposium on Computer Mapping and GIS for Coastal Zone Management*. Aberdeen, Scotland: CoastGIS International Executive.

Wright, D. J., M. J. Blongewicz, P. N. Halpin, and J. Breman. 2007. *Arc Marine: GIS for a Blue Planet*. ESRI Press: Redlands, CA.

Zeiler, M. 1999. *Modeling our World*. ESRI Press: Redlands, CA.

APPENDICES

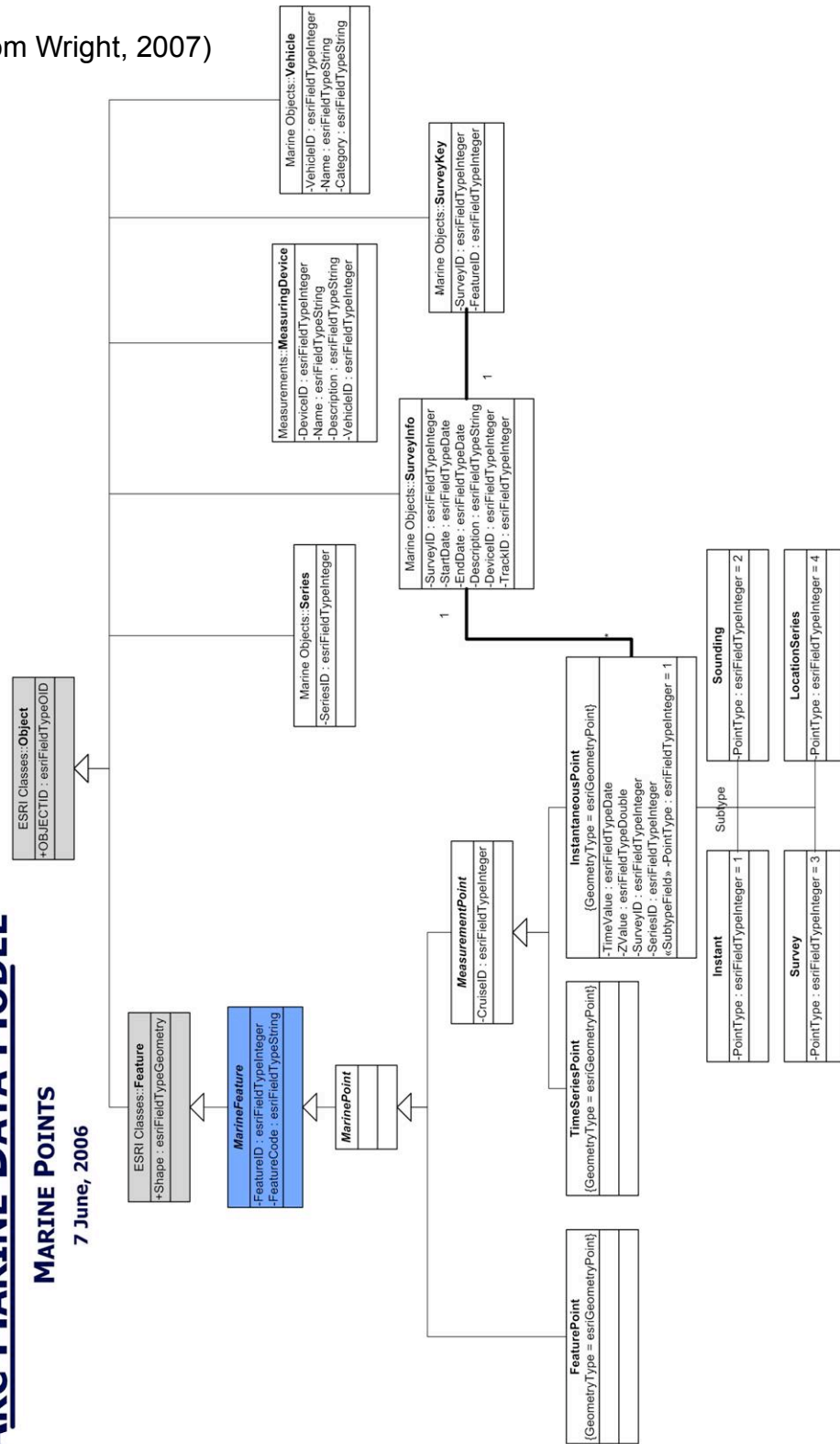
Appendix A. Arc Marine Data Model Diagrams

(from Wright, 2007)

ARC MARINE DATA MODEL

MARINE POINTS

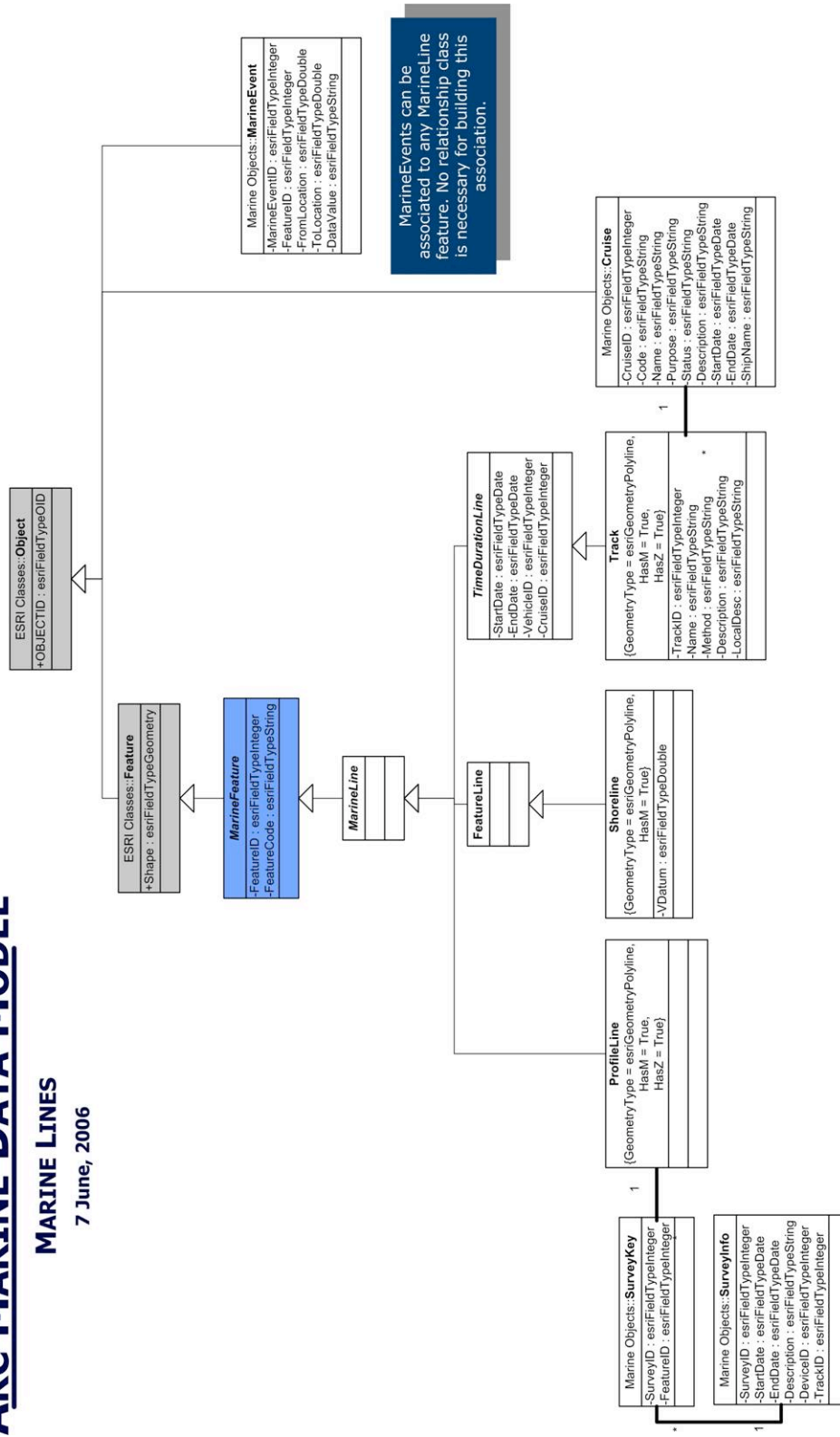
7 June, 2006



ARC MARINE DATA MODEL

MARINE LINES

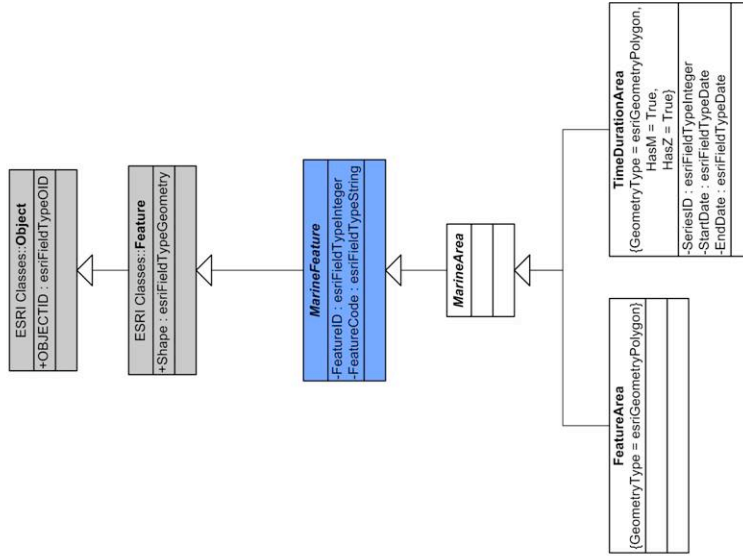
7 June, 2006



ARC MARINE DATA MODEL

MARINE AREAS

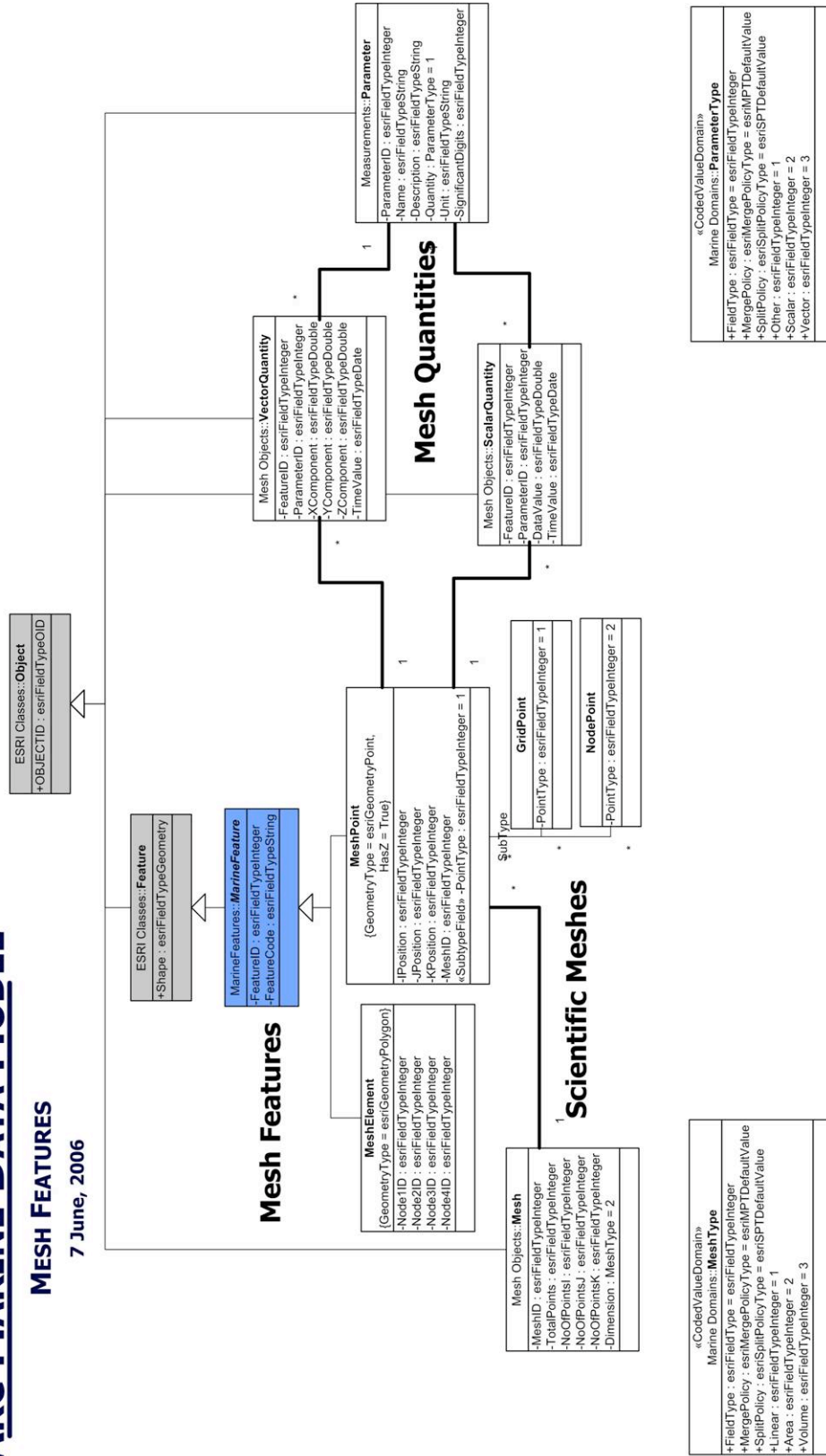
7 June, 2006



ARC MARINE DATA MODEL

MESH FEATURES

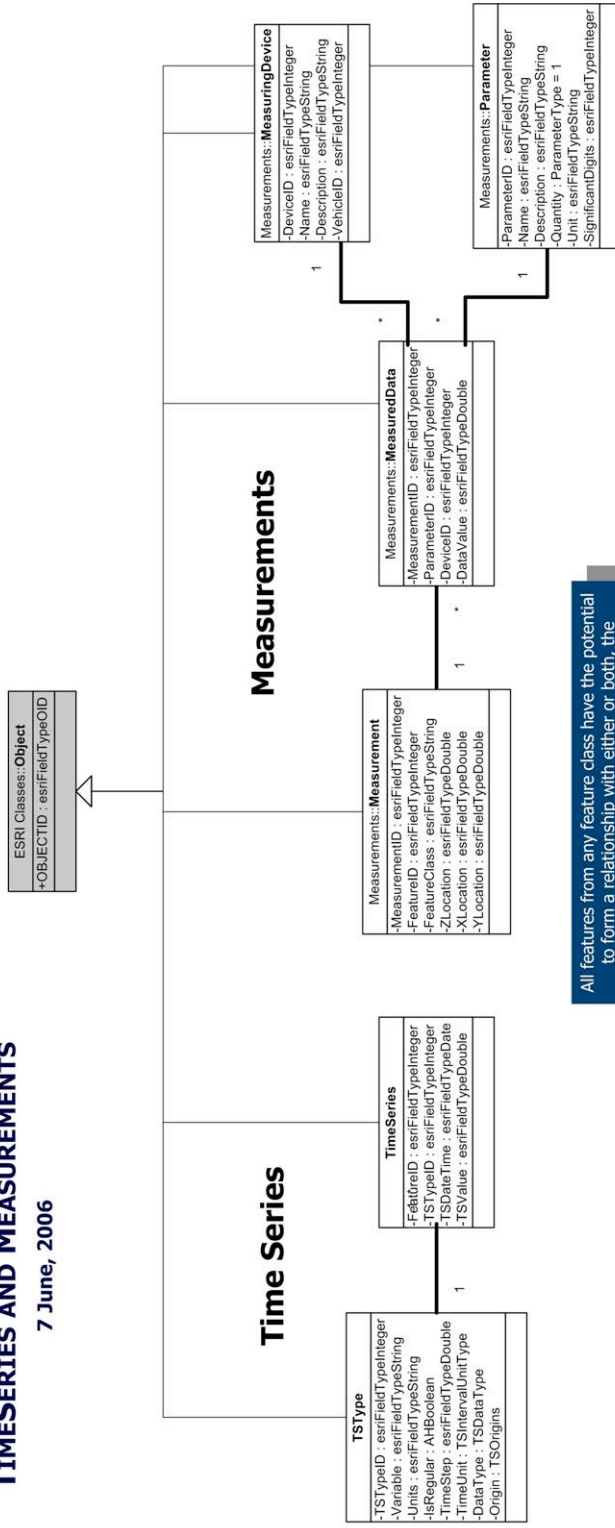
7 June, 2006



ARC MARINE DATA MODEL

TIMESERIES AND MEASUREMENTS

7 June, 2006



All features from any feature class have the potential to form a relationship with either or both, the Measurement or TimeSeries classes. Generally, TimeSeries data is collected at various depths and therefore the relationship from the FeatureClass to TimeSeries should go through Measurement

«CodedValueDomain»
Marine Domains: TSIntervalUnitType
+Field Type : esriFieldType = esriFieldTypeInteger
+MergePolicy : esriMergePolicyType = esriMPTDefault
+SplitPolicy : esriSplitPolicyType = esriSPTDefault
+Second : esriFieldTypeInteger = 1
+Minute : esriFieldTypeInteger = 2
+Hour : esriFieldTypeInteger = 3
+Day : esriFieldTypeInteger = 4
+Week : esriFieldTypeInteger = 5
+Month : esriFieldTypeInteger = 6
+Year : esriFieldTypeInteger = 7

Marine Features: InstantaneousPoint
-TimeValue : esriFieldTypeDate
-ZValue : esriFieldTypeDouble
-SurveyID : esriFieldTypeInteger
-SeriesID : esriFieldTypeInteger
«SubtypeField» -PointType : esriFieldTypeInteger = 1

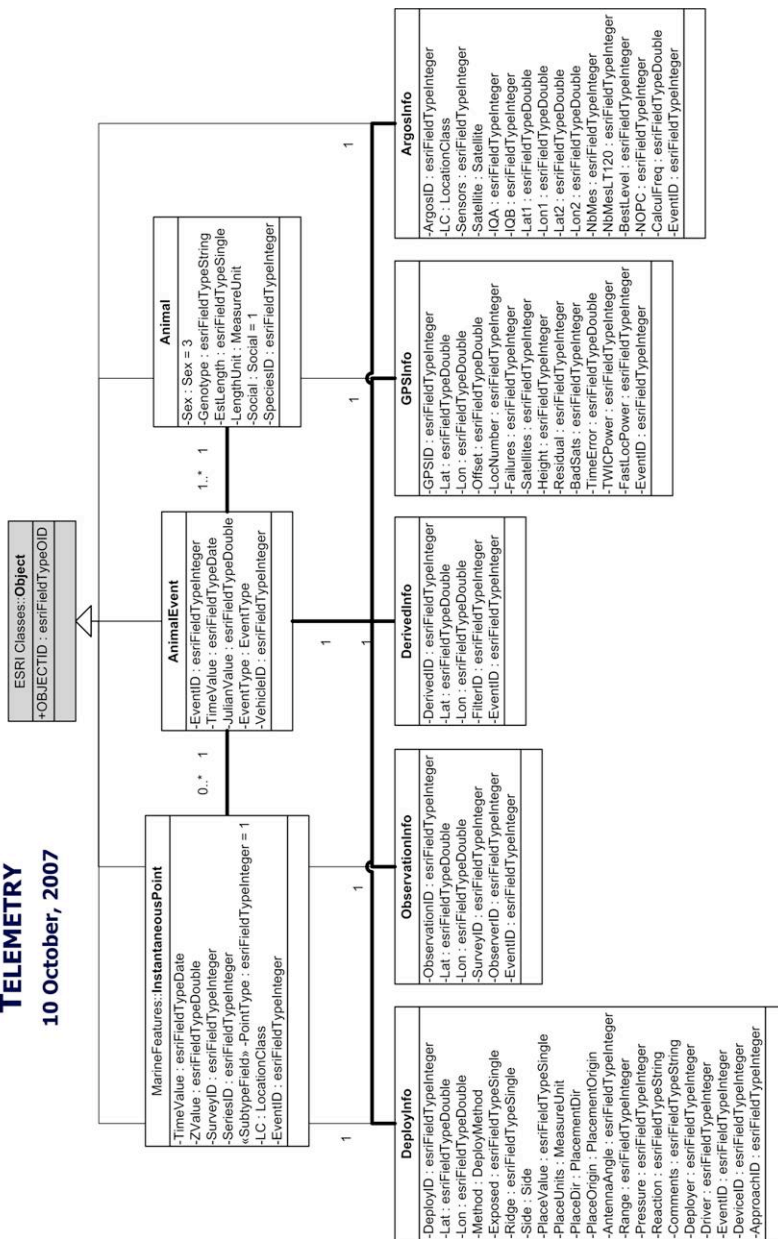
Marine Features

Appendix B. MMI Customization Data Model Diagram

ARC MARINE DATA MODEL

TELEMETRY

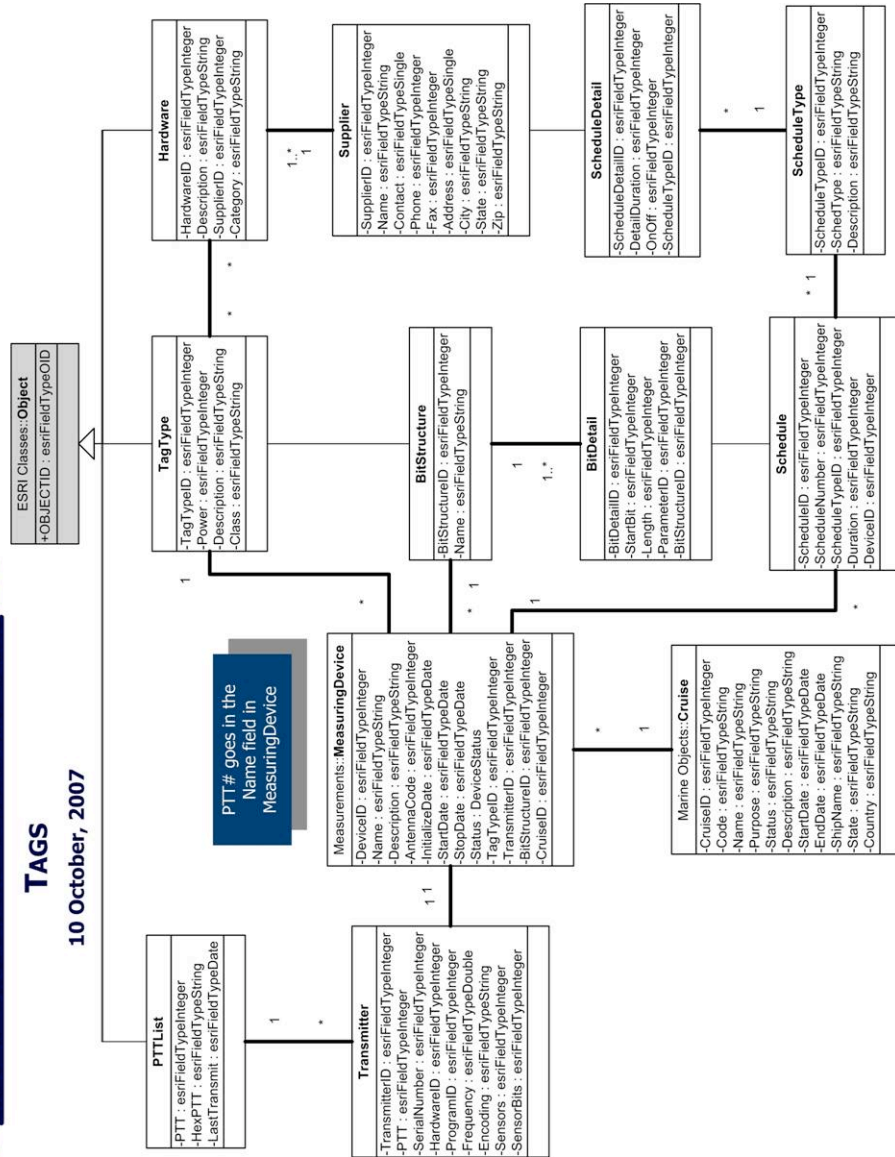
10 October, 2007



ARC MARINE DATA MODEL

TAGS

10 October, 2007

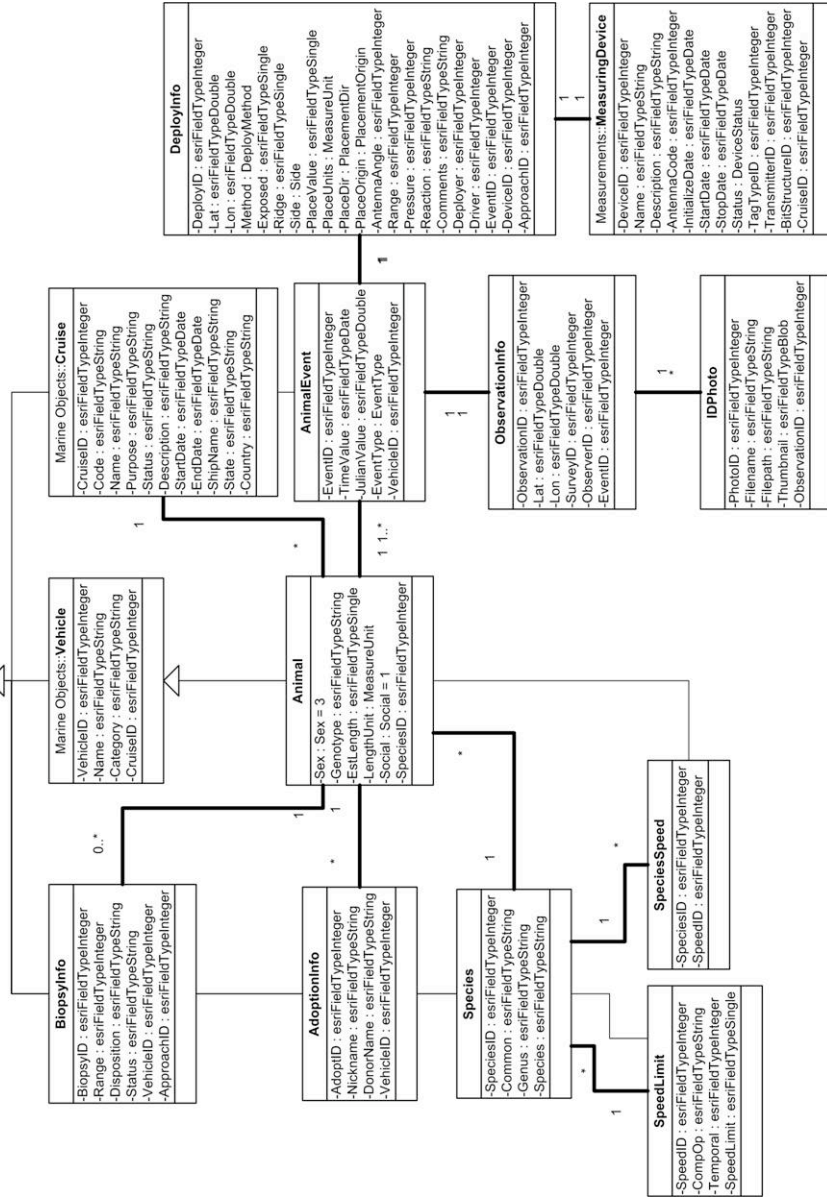


ARC MARINE DATA MODEL

ANIMALS

10 October, 2007

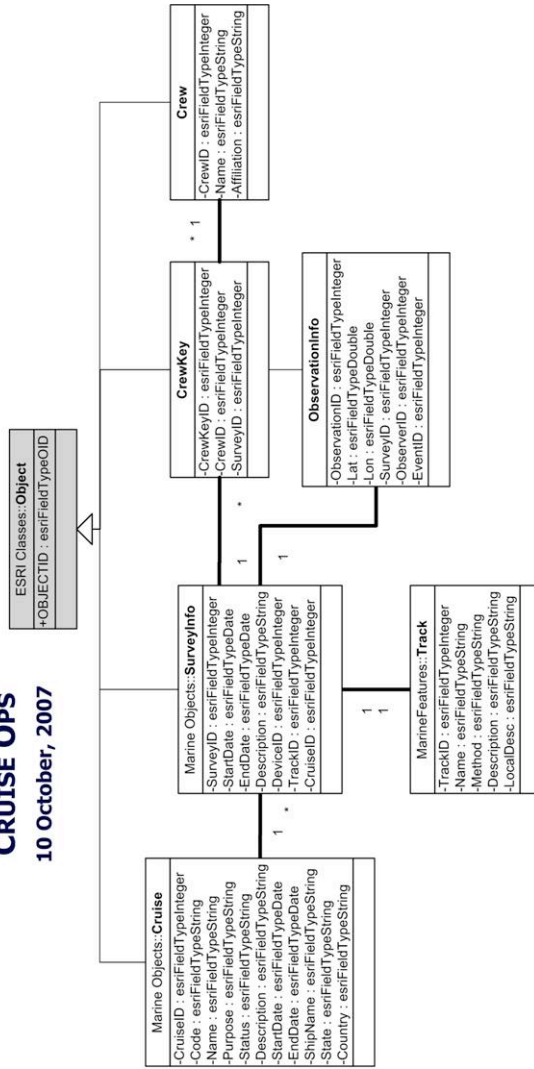
ESRI Classes: Object
+OBJECTID : esriFieldTypeOID



ARC MARINE DATA MODEL

CRUISE OPS

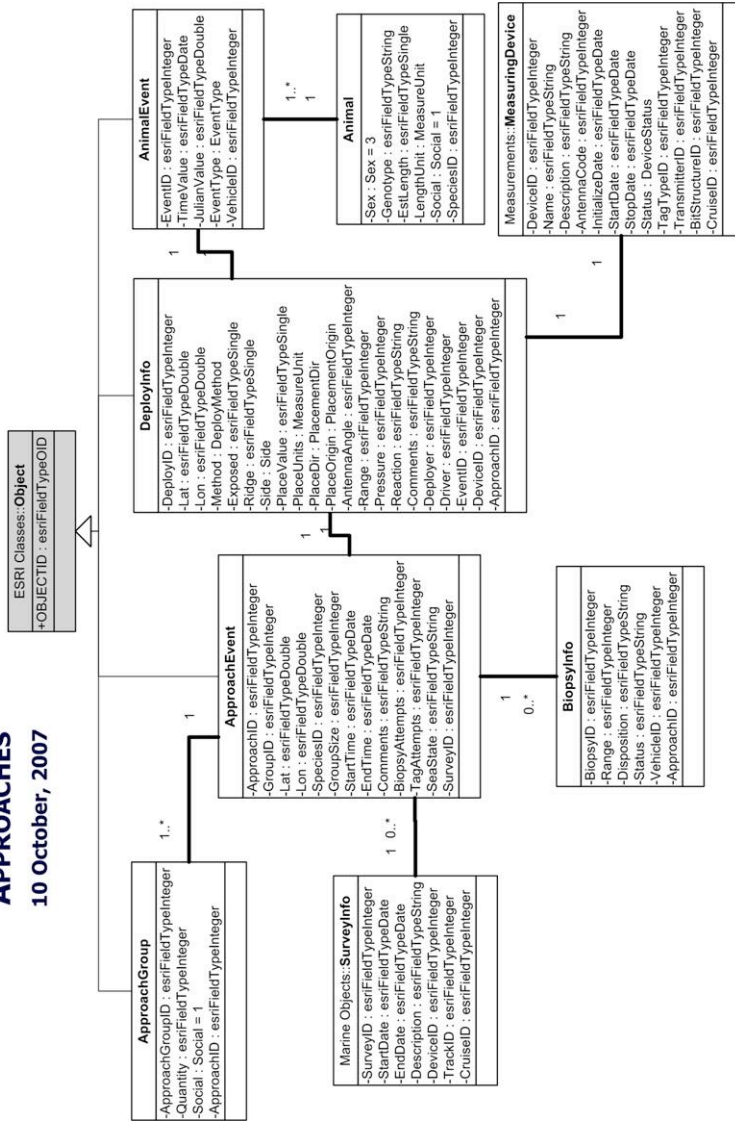
10 October, 2007



ARC MARINE DATA MODEL

APPROACHES

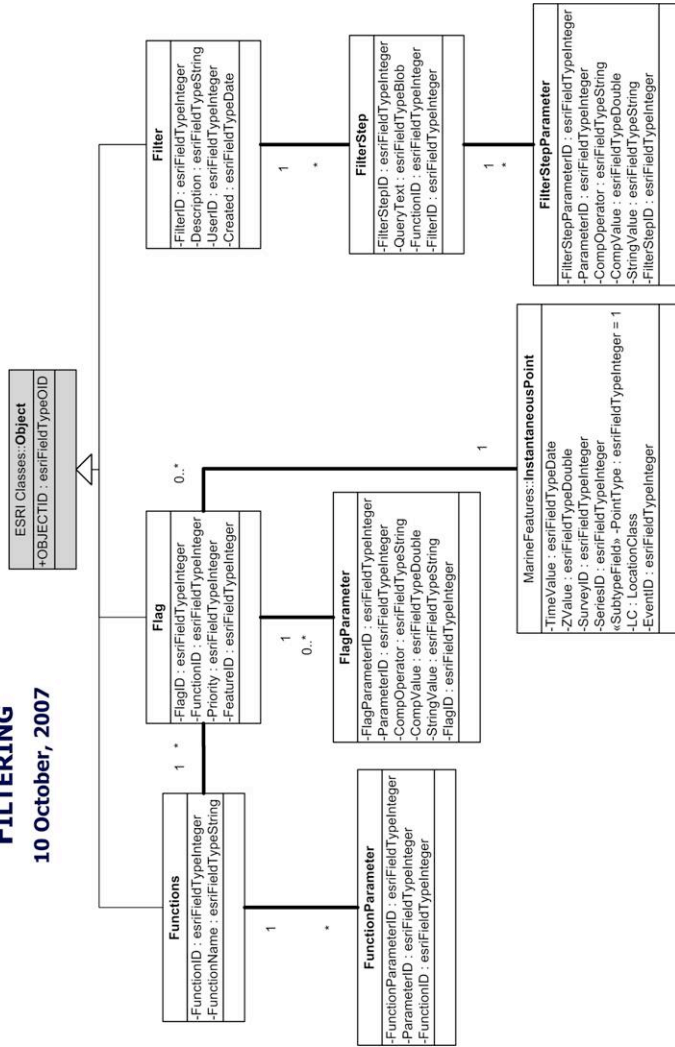
10 October, 2007



ARC MARINE DATA MODEL

FILTERING

10 October, 2007



Appendix C. GIS Procedures

This appendix represents a sampling of basic GIS procedures and automation methods as they relate to the use of Arc Marine in marine animal tracking. The Excel worksheet, *DemonstrationSet.xls* is an archived subset of sperm whale tracking data from the Marine Mammal Institute. These examples use both file geodatabases (.gdb) and personal geodatabases (.mdb). Each example is appropriate for both types as well as ArcSDE.

Loading LocationSeries Point from Excel

For an overview of creating an Arc Marine geodatabase from the core schema or a customized schema, see the Arc Marine Tutorial available at:

http://dusk2.geo.orst.edu/djl/arccgis/ArcMarine_Tutorial/

1) To begin with, examine the fields in the core version of Instantaneous Point (Figure A1). Note that the PointType field is set to the integer “4” for all LocationSeries points in the table. LocationSeries is not loaded to a separate table, but rather collected with all subtypes of InstantaneousPoint.

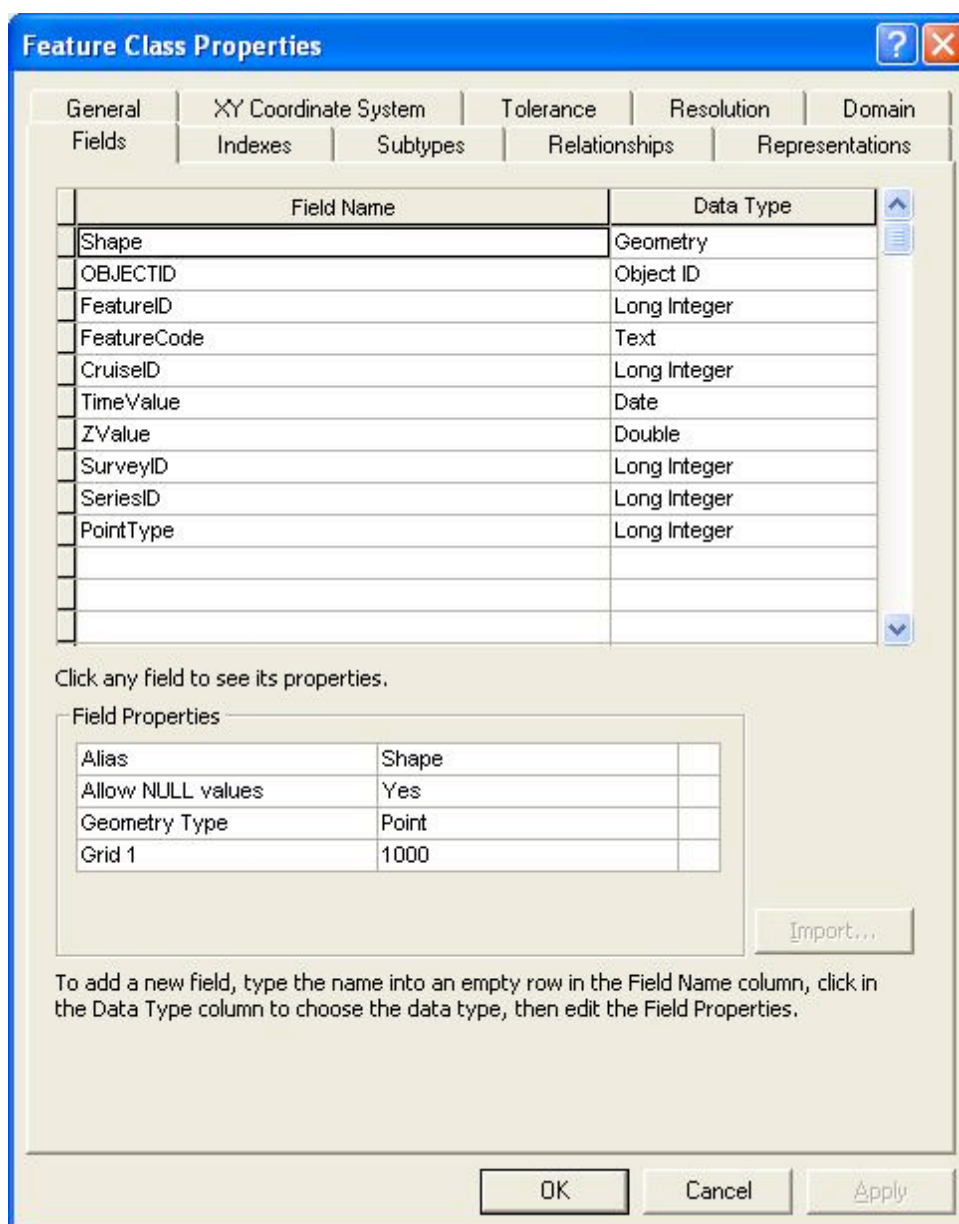


Figure A1. Structure of InstantaneousPoint

2) Shape geometry cannot be created directly from an Excel spreadsheet into an existing table. That Excel sheet, though, can be used to generate feature geometry. With LocationSeries, this geometry should always be created in a feature class and not a shapefile. Shapefiles

truncate time information from a date/time stamp and will result in only dates being listed in the TimeValue field.

To create the new feature class, expand the Excel file in ArcCatalog and right-click the sheet containing the feature information. Select the option Create Feature Class > From XY Table... (Figure A2).

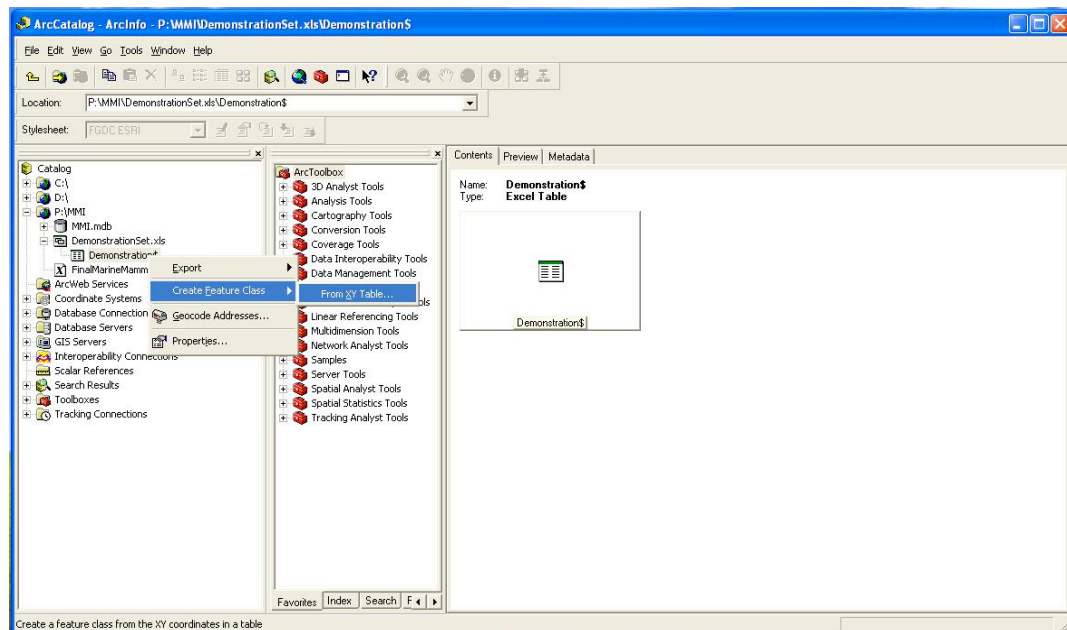


Figure A2. Creating a feature class from an Excel table.

3) Fill out the dialog as follows. Be certain to select an appropriate coordinate system, as ArcGIS will not select an appropriate one for you. X and Y may also be, respectively, longitude and latitude depending on how your data sheet is designed. Notice output is directed to a feature class inside the Arc Marine geodatabase (MMI.gdb) (Figure A3).

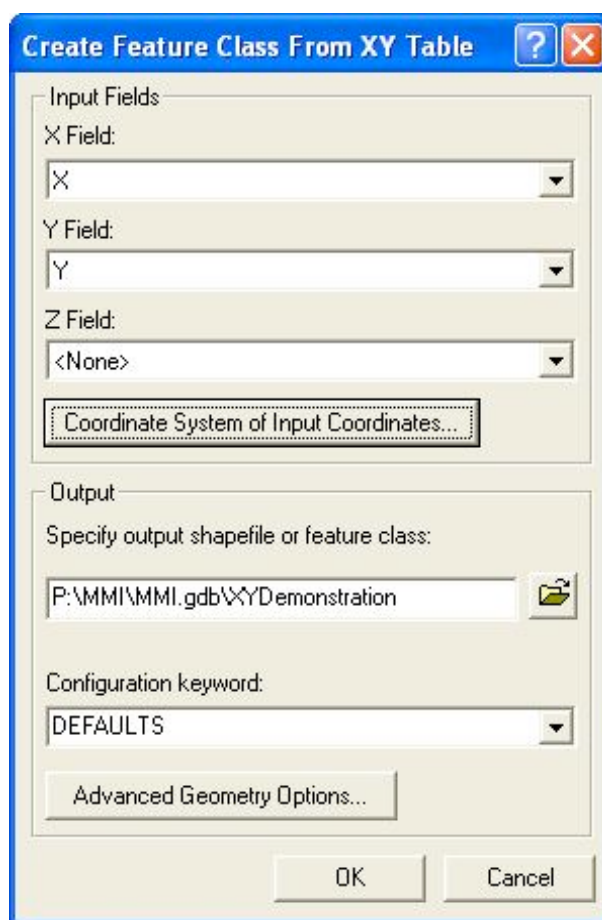


Figure A3. Arguments for Create Feature Class From XY Table

4) Once the new feature class is created (here called XYDemonstration), browse to it, right-click, and select properties. These fields will have to be modified to match the schema of InstantaneousPoint (Figure A4). The names of the fields are not important; rather it is the field types that must match. (Note that CruiseID in the example is a double, unlike the short integer field of CruiseID in InstantaneousPoint).

Feature Class Properties

General | XY Coordinate System | Tolerance | Resolution | Domain
Fields | Indexes | Subtypes | Relationships | Representations

Field Name	Data Type
OBJECTID	Object ID
CruiseID	Double
PTT	Text
Y	Double
X	Double
TimeValue	Date
PointType	Double
LocationClass	Text
Flagged	Text
Shape	Geometry

Click any field to see its properties.

Field Properties

Alias	OBJECTID
-------	----------

Import...

To add a new field, type the name into an empty row in the Field Name column, click in the Data Type column to choose the data type, then edit the Field Properties.

OK Cancel Apply

Figure A4. Attributes of XYDemonstration

5) Here is the same table with additional fields added to match the InstantaneousPoint schema (Figure A5). Fields without data can be omitted (for example, ZValue was blank for all records in this dataset). Editing is up to the user. Entire fields can be calculated using the Field

Calculator in ArcMap. For large datasets, it is recommended to use first load the feature class into a personal geodatabase and utilize Update queries within Microsoft Access. Then the edited feature class can be transferred to a file geodatabase.

Feature Class Properties

General | XY Coordinate System | Tolerance | Resolution | Domain
Fields | Indexes | Subtypes | Relationships | Representations

Field Name	Data Type
PTT	Text
Y	Double
X	Double
TimeValue	Date
PointType	Double
LocationClass	Text
Flagged	Text
Shape	Geometry
FeatureID	Long Integer
FeatureCode	Text
ZValue	Double
SurveyID	Long Integer
SeriesID	Long Integer

Click any field to see its properties.

Field Properties

Alias	
Allow NULL values	Yes
Default Value	
Domain	

Import...

To add a new field, type the name into an empty row in the Field Name column, click in the Data Type column to choose the data type, then edit the Field Properties.

OK Cancel Apply

Figure A5. XYDemonstration with InstantaneousPoint schema

6) Once the schema of the new feature class is edited and fields transformed to the appropriate data type, InstantaneousPoint can be loaded. Right-click InstantaneousPoint in ArcCatalog and select Load > Load Data... (Figure A6).

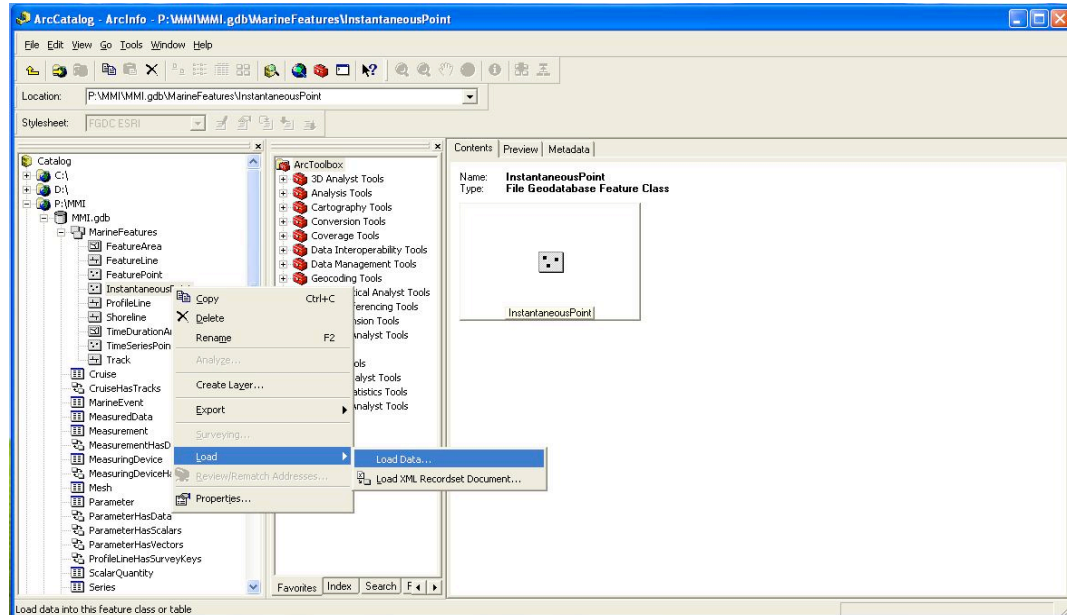
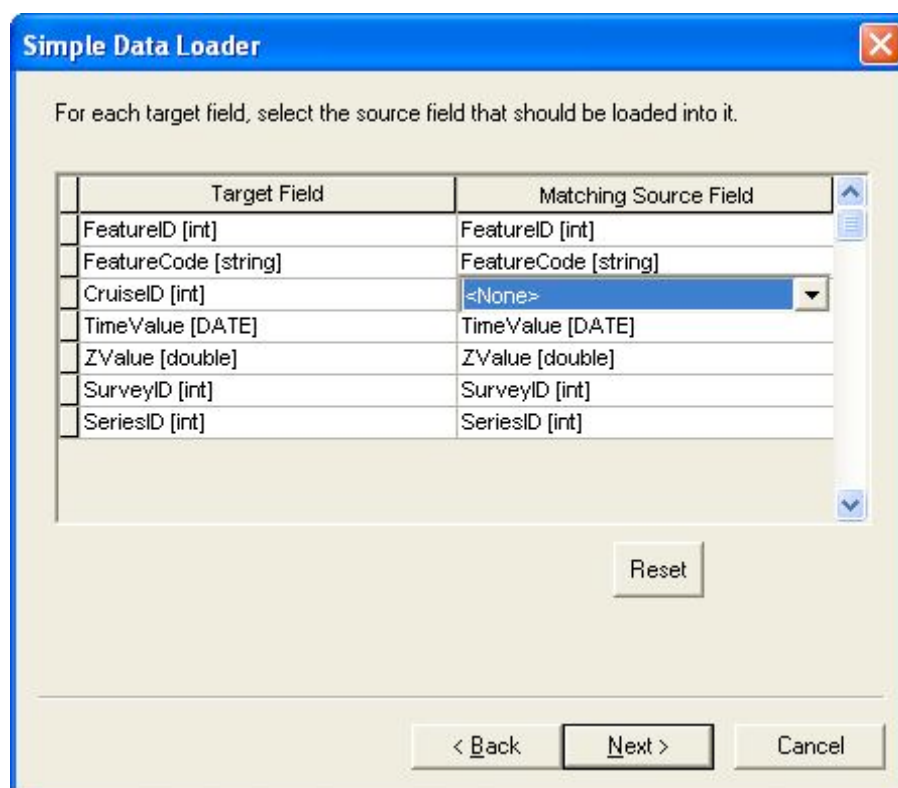


Figure A6. Executing the loading of InstantaneousPoint

7) Fields are automatically matched by name and type (Figure A7).

A source field can also be set to the correct field when there is a not a name match. CruiseID as a double cannot be downconverted automatically to an integer, so this field is unmatched. Loading is still possible, but this field would then be null.



The Simple Data Loader dialog box contains a table for mapping target fields to source fields. The table has two columns: 'Target Field' and 'Matching Source Field'. The rows are as follows:

Target Field	Matching Source Field
FeatureID [int]	FeatureID [int]
FeatureCode [string]	FeatureCode [string]
CruiseID [int]	<None>
TimeValue [DATE]	TimeValue [DATE]
ZValue [double]	ZValue [double]
SurveyID [int]	SurveyID [int]
SeriesID [int]	SeriesID [int]

Below the table is a 'Reset' button. At the bottom of the dialog are three buttons: '< Back', 'Next >', and 'Cancel'.

Figure A7. Field matching in the Simple Data Loader

8) After a quick re-editing of CruiseID, the Simple Data Loader process is repeated with default values, resulting in a populated InstantaneousPoint table (Figure A8).

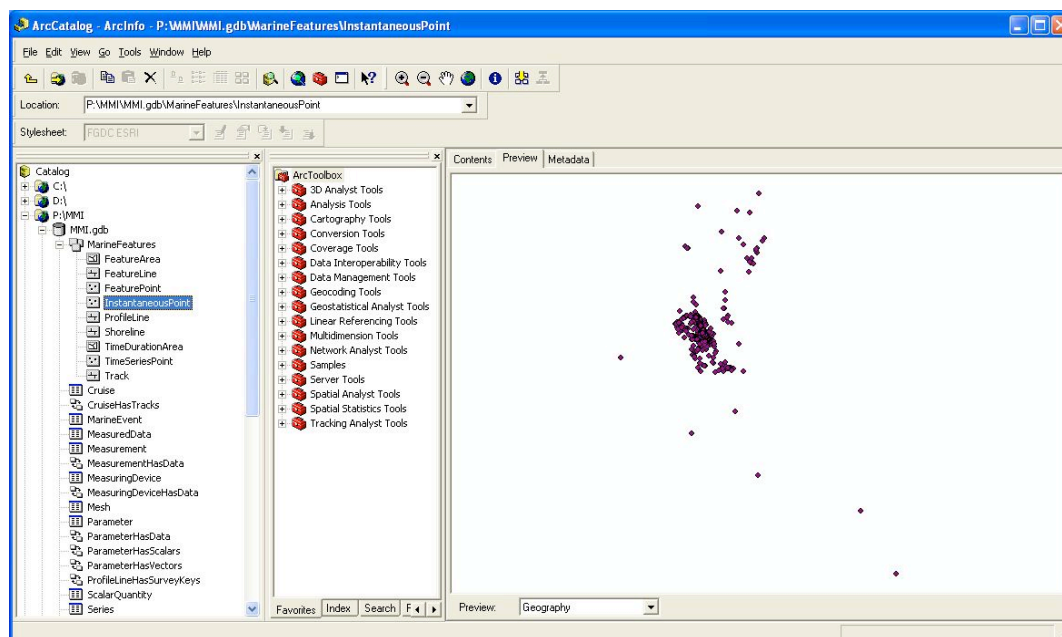


Figure A8. InstantaneousPoint populated from DemonstrationSet.xls

Loading *InstantaneousPoint* from OBIS-SEAMAP

1) To begin with, point your web browser to

<http://seamap.env.duke.edu/datasets> (Read et al., 2006) to select

downloadable datasets from the OBIS-SEAMAP Data Distribution server

(Figure A9).

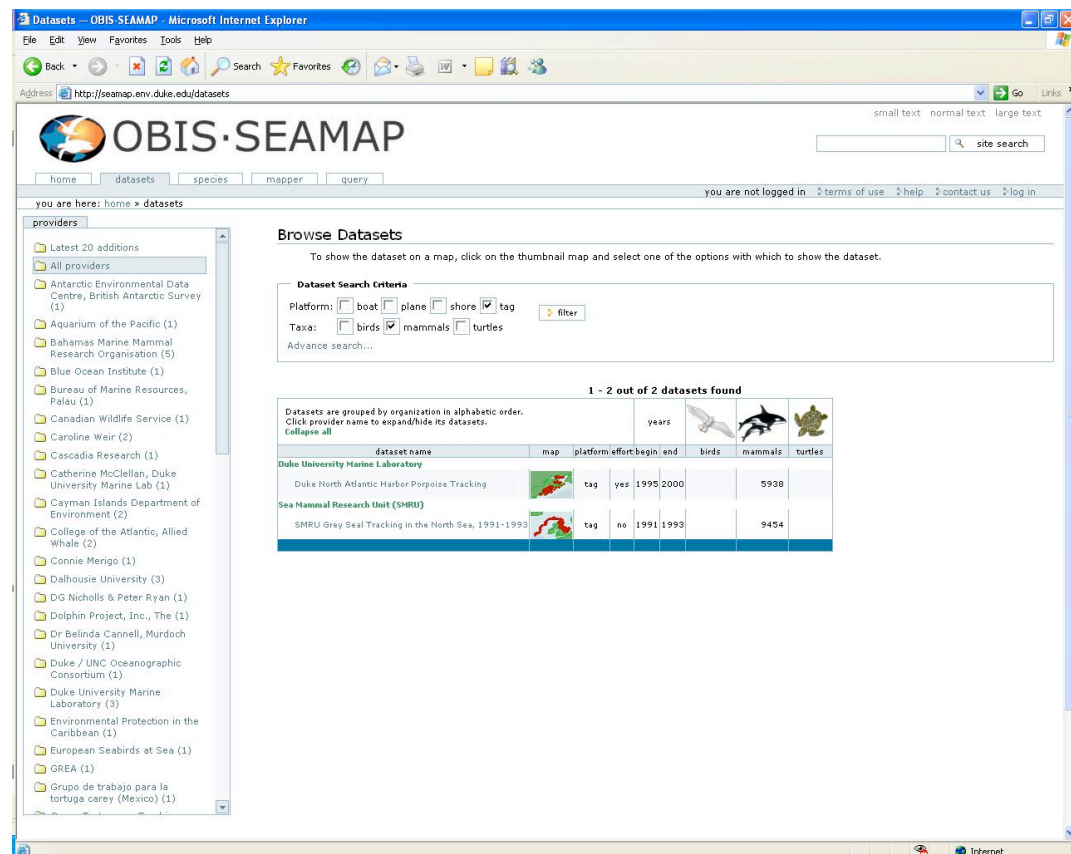


Figure A9. OBIS-SEAMAP dataset browsing screen.

2) When selecting a dataset to download, choose the CSV form.

The shapefile form will truncate time from date/time stamps in the dataset.

The full date/time stamp is preserved in CSV format (Figure A10).

Datasets - OBIS-SEAMAP - Microsoft Internet Explorer

Address: http://seamap.env.duke.edu/datasets/detail/83

OBIS-SEAMAP

small text normal text large text

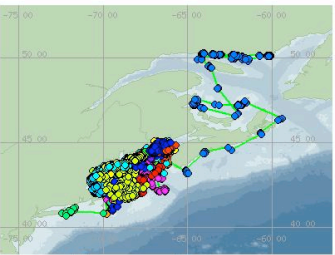
home datasets species mapper query

you are here: home > datasets > detail


you are not logged in | terms of use | help | contact us | log in

Duke North Atlantic Harbor Porpoise Tracking

ID	83
# of Records	5938
Date, Begin	1995-08-13
Date, End	2000-11-22
Latitude, Min	40.71
Latitude, Max	50.29
Longitude, Min	-72.44
Longitude, Max	-59.41
Platform	tag
Trackline	Yes (ID: 84)
Last updated	2006-10-20 10:12:55



larger image legend image

Data provider  Duke University Marine Laboratory

Abstract

These data were obtained from harbour porpoises (*Phocoena phocoena*) equipped with satellite-linked transmitters in the Gulf of Maine. All porpoises were obtained from herring weirs on the island of Grand Manan, New Brunswick Canada (44°45' N 67°45' W) during the summer months (July to September). Details on each porpoise are given in the summary table. Positional data are obtained via Service ARGOS (<http://www.argosinc.com/>). See Read and Westgate (1997) for complete details of tagging methods.

References

Read, A.J. & A.J. Westgate. 1997. Monitoring the movements of harbour porpoises (*Phocoena phocoena*) with satellite telemetry. *Marine Biology*, 130: 315-322

Contacts

According to OBIS-SEAMAP [Terms of Use](#), you are requested to contact the data provider(s) listed below as "Primary contact" for use of the dataset in any publication, product, or commercial application.

name	role
Andy Read	Primary contact
Benjamin Best	Data Entry

Copyright © 2002-2007. Please see our [Terms of Use](#). We're happy to field your questions, comments or suggestions.

javascript: void(0); Internet

Figure A10. OBIS-SEAMAP dataset download screen.

3) As with the Excel spreadsheet, browse to the CSV file in ArcCatalog and select Create Feature Class > From XY Table... (Figure A11). Direct output to a feature class (once again, do not use a shapefile) and set the X field to _lon and the Y field to _lat. Set your coordinate system to WGS84 (Figure A12). Each of these values is a standardized format for OBIS-SEAMAP.

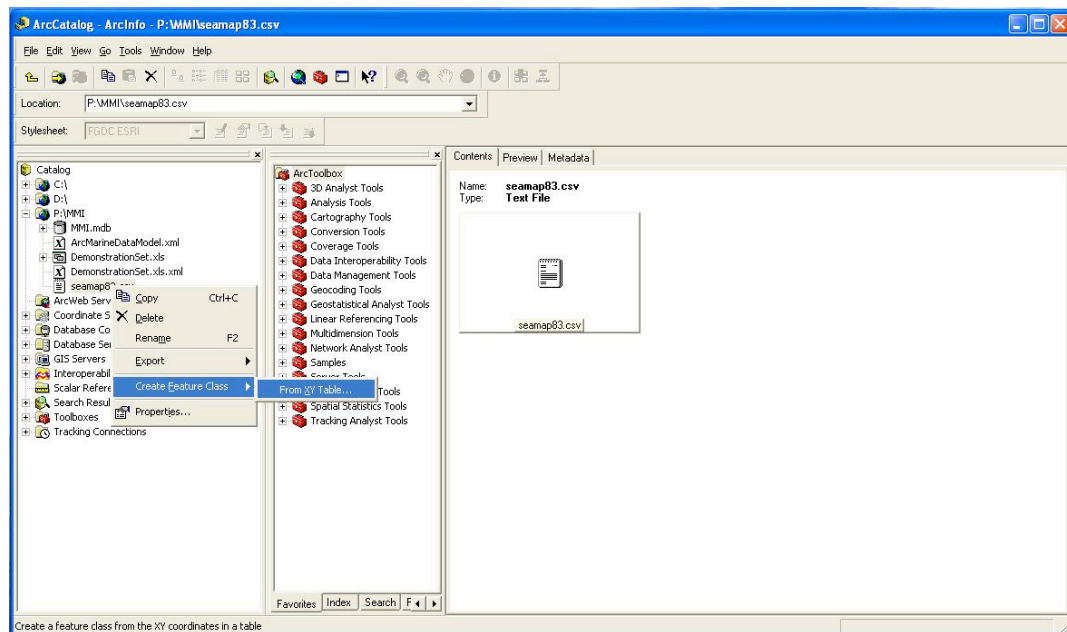


Figure A11. Creating a feature class from the CSV file.

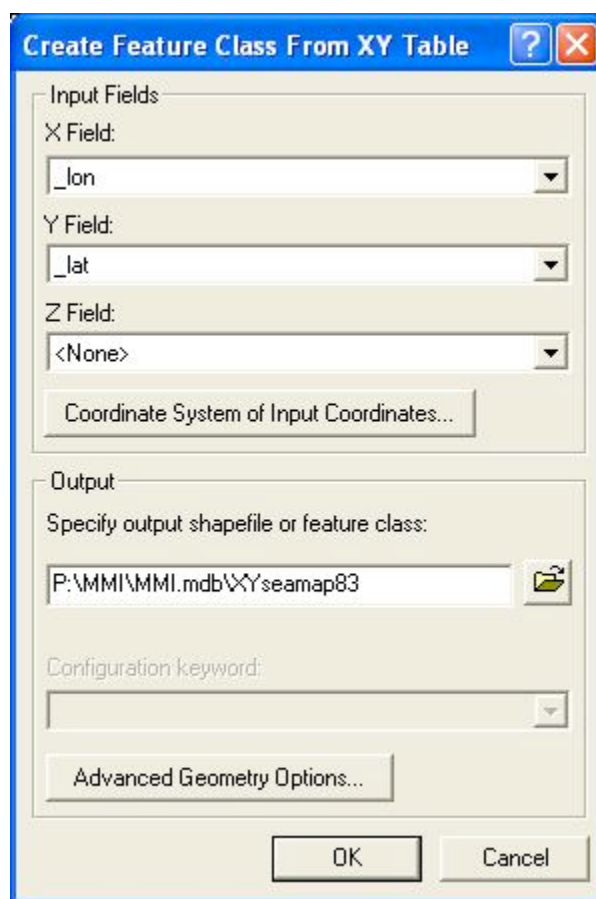


Figure A12. Arguments for Create Feature Class From XY Table for any OBIS-SEAMAP CSV file

4) There are two main forms for OBIS-SEAMAP datasets at the point of matching source fields to target fields. For data loading, the only significant difference is that one form (Figure A13) has an integer “dataset” field which should be loaded into CruiseID. The other form (Figure A14) has a string “owner” field which can be loaded into FeatureCode. While other forms are possible as well, all OBIS-SEAMAP datasets are required to have the _lon, _lat, and obs_datetime fields, and Seamap will add at least one identifier field from dataset or owner.

Simple Data Loader

For each target field, select the source field that should be loaded into it.

Target Field	Matching Source Field
FeatureID [int]	id [int]
FeatureCode [string]	<None>
CruiseID [int]	dataset [int]
TimeValue [DATE]	obs_date [string]
ZValue [double]	<None>
SurveyID [int]	<None>
SeriesID [int]	series [string]

Reset

< Back Next > Cancel

Figure A13. "Dataset" OBIS-SEAMAP schema

Simple Data Loader

For each target field, select the source field that should be loaded into it.

Target Field	Matching Source Field
FeatureID [int]	id [int]
FeatureCode [string]	owner [string]
CruiseID [int]	<None>
TimeValue [DATE]	obs_datetimez [string]
ZValue [double]	<None>
SurveyID [int]	<None>
SeriesID [int]	series [string]

Reset

< Back Next > Cancel

Figure A14. "Owner" OBIS-SEAMAP schema

5) Figure A15 shows several datasets loaded simultaneously into InstantaneousPoint and displayed according to the FeatureCode field. Multiple datasets can be loaded at the same time if they have matching schemas (see step 4 above).

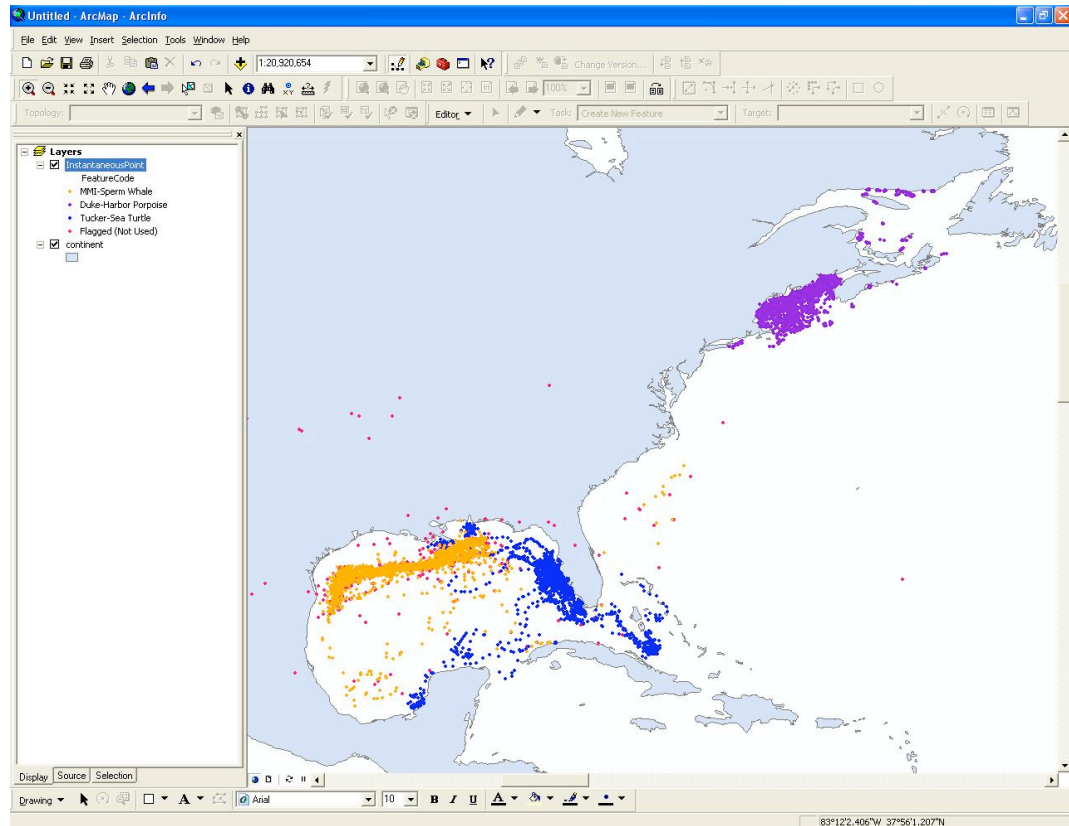


Figure A15. MMI sperm whale data with datasets from OBIS-SEAMAP. MMI sperm whale data is from DemonstrationSet.xls with excluded points flagged (for example, points on land; Read and Westgate, 1997; Duke University Marine Laboratory, 2004; Mote Marine Laboratory, 2007; Read et al., 2007).

Point to Path in Third-Party Extensions

This section demonstrates how to use the Flag table with LocationSeries point to build linear interpolated tracking paths using two third-party ArcGIS extensions commonly used in analyzing animal movement: Hawth's Tools and XTools. Be aware that Hawth's Tools is not being regularly updated for new versions of ArcGIS.

1) Generally, operations with third-party applications will be best handled with snapshot record sets (i.e. new feature classes pre-built with joined attributes). The filtering functions of the Marine Mammal Institution customization will eventually be built to automatically generate such snapshots. Instead, here Flag was joined to InstantaneousPoint. Note that the join cannot be a relationship join because of the many to one relationship from Flag to InstantaneousPoint.

2) With the joined table, the dataset is narrowed to only first priority Argos points with a layer definition query (Figure A16). This eliminates points that are flagged as "bad" (see Figure A15). The feature class is now ready for use in third-party extensions.

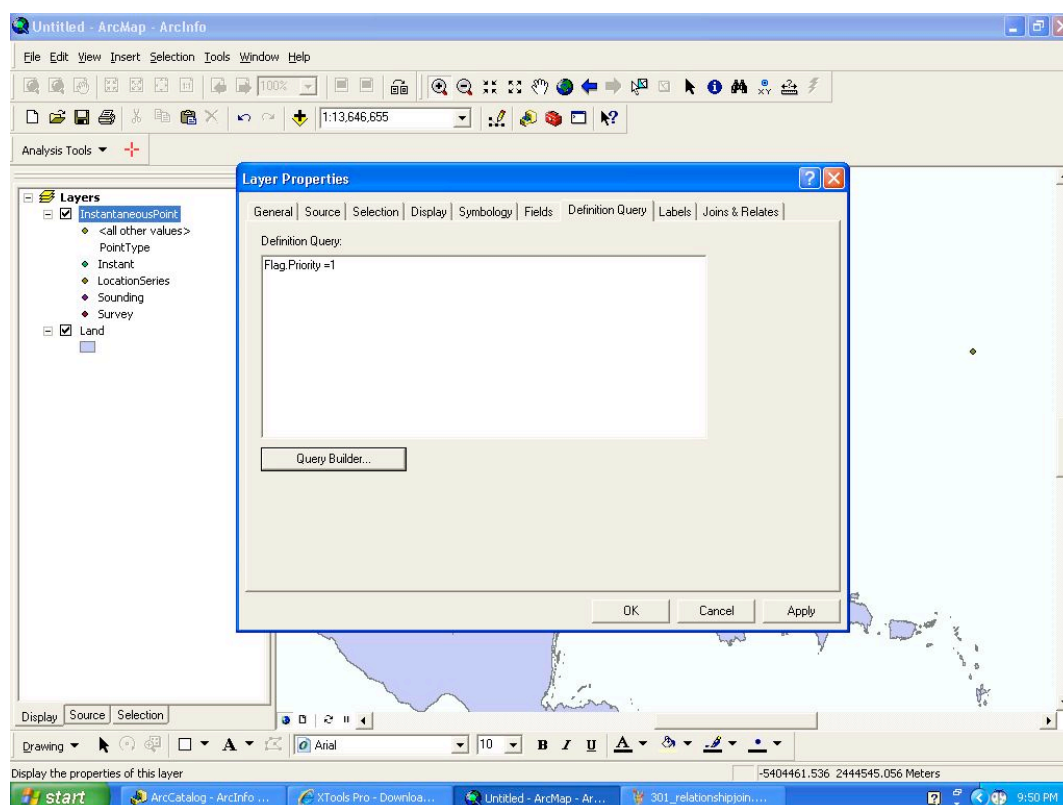


Figure A16. Setting the Definition Query to exclude “bad” points.

3) Hawth’s tools requires output to a shapefile. The output path can vary, but all other arguments are standard for any use of InstantaneousPoint with Convert Locations to Path in Hawth’s tools (Figure A17). “Make each segment a separate line” is an optional argument depending on user preference.

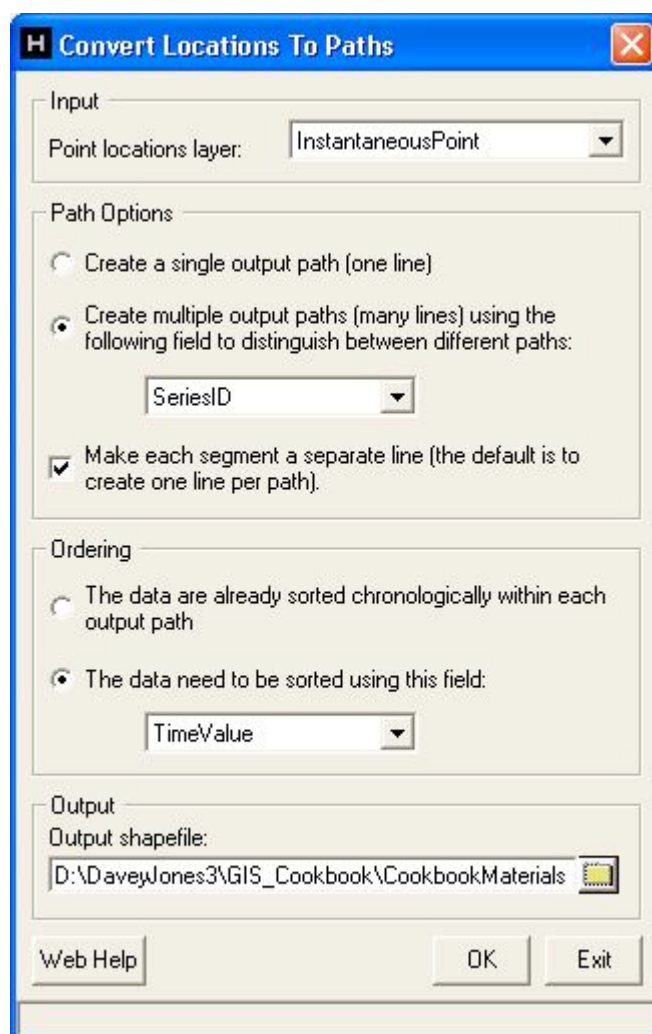


Figure A17. Using LocationSeries with Hawth's Tools

4) Using the “Make One Polyline from Points” tool in XTools is similarly simple. Output storage should be directed to a feature class for loading into the Tracks class in Arc Marine. Again, all arguments other than Output Storage will be identical for any use of Arc Marine with this tool (Figure A18). The final output is displayed in Figure A19.

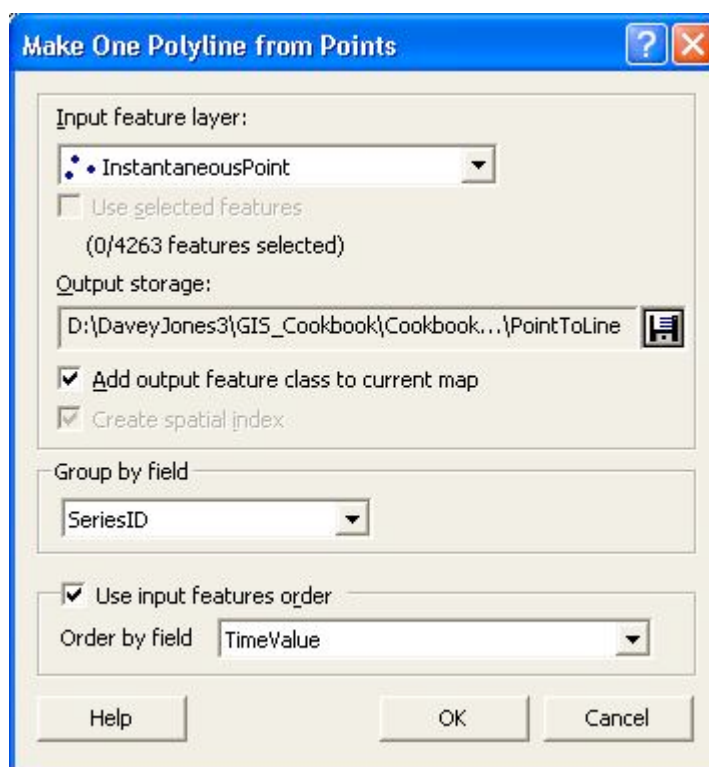


Figure A18. Using LocationSeries with XTools

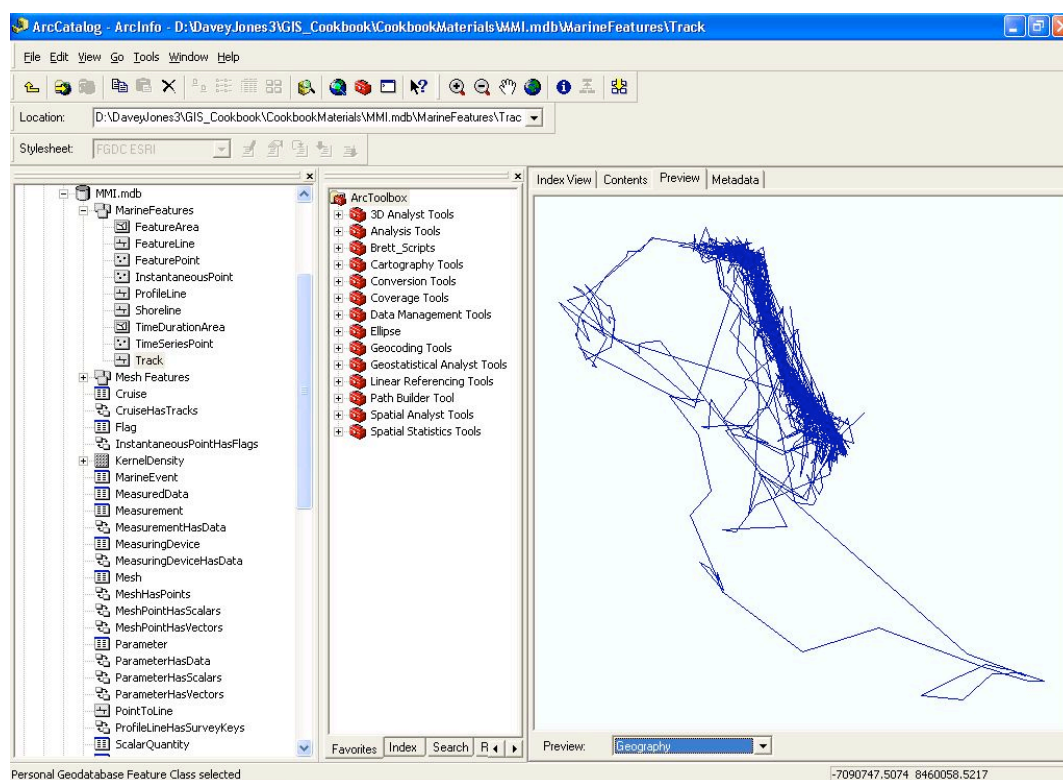


Figure A19. Animal paths loaded from XTools output into Track. Note that the SeriesID must be manually linked to Animal (Vehicle child class) to provide a relationship between LocationSeries points and the associated Track.

Using LocationSeries in Model Builder

Below are several samples demonstrating the use of InstantaneousPoint in Model Builder to create Arc Marine geoprocessing tools common to animal tracking.

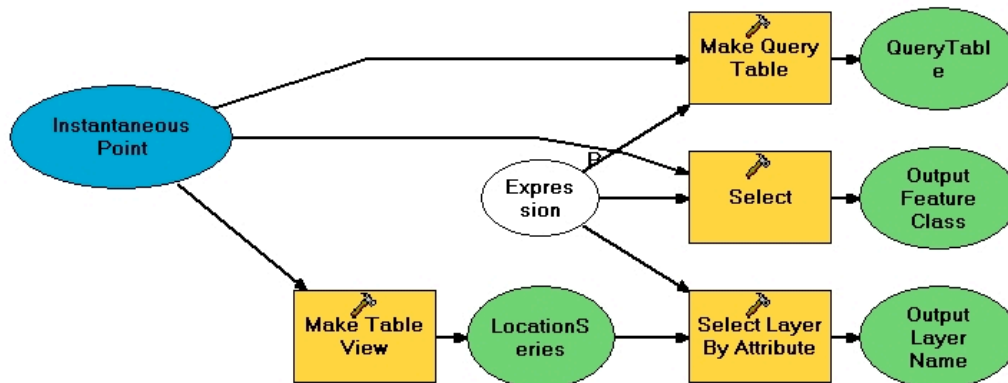


Figure A20. Three selection paths.

This model shows three examples of how to create model input feature layers based on an SQL expression as a model parameter. When used in this manner, SQL Expression Builder is available at run time.

Figure A20 shows three possible methods for delivering LocationSeries points to Model Builder processes. The SQL Expression should include the criteria "[PointType]=4" to designate the LocationSeries subtype of InstantaneousPoint. The top path, Make Query Table, allows for the use of complex queries that can take advantage of relationship joins (such as InstantaneousPoint to Series to Animal to Species) built into the MMI customization or core Arc Marine. This path generates an output table with feature geometry. The middle path, Select, is the most commonly used method to deliver LocationSeries to a model process tool. The last path is used for tools that specifically call for a selection layer or can use a selection layer. This is identical to use a loaded ArcMap layer as tool input.

Figures A21 and A22 display models utilizing the Select selection path to generate common analytic outputs

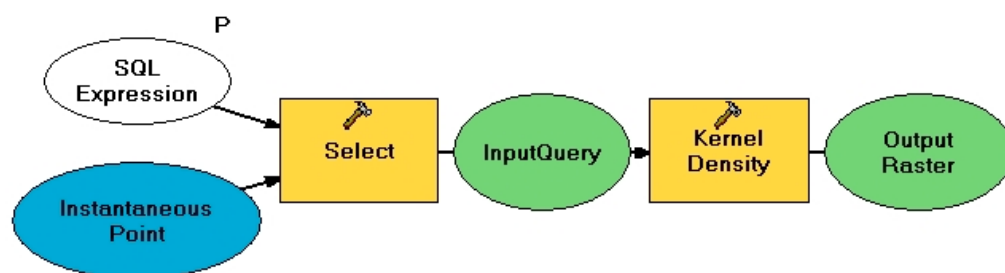


Figure A21. InstantaneousPoint input to the Kernel Density tool. Raster output is redirected back into the Arc Marine geodatabase.

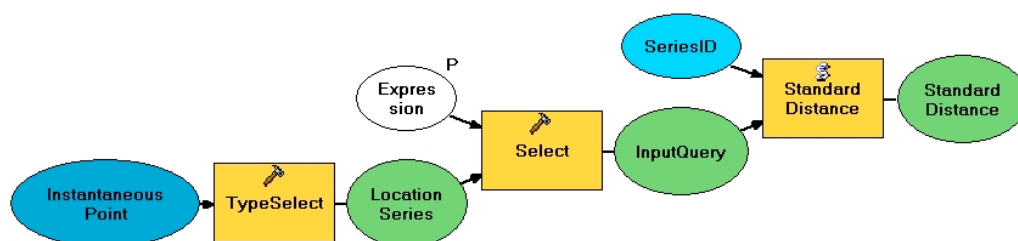


Figure A22. LocationSeries input to Standard Distance geoprocessing script.

In the slightly more complex model depicted in Figure A22, LocationSeries is preselected through use of the Select tool (here labeled TypeSelect) with the SQL string “[PointType]=4”. SeriesID is a defined grouping parameter for the Standard Distance script while an additional SQL expression is available to further define the input LocationSeries points. By substituting for the Standard Distance script (and with appropriate parameters in place of SeriesID), this model can be used to drive any Python script written to use a point feature or written to specifically use an InstantaneousPoint feature class.

Appendix D. Python Codebase

The following is the Python code base so far with associated programmer's notes. At least Python 2. 3 is required (for the datetime module) and Python 2. 4 is recommended for compatibility with ArcGIS 9. The only required additional module is adodb (or mxodbc) for use with the DBTESTING database abstraction layer code. The Marine Geospatial Ecology Tools (MGET), or GeoEco module, from the Duke Marine Geospatial Ecology Laboratory (<http://code.env.duke.edu/projects/mget/>) is also highly recommended. While MGET metadata references are integrated into the code base, it is not a required module.

DBTESTING.PY

This is the module in development. At the top are two lines that must be edited. They store the DSN connection string for the testing (or production later) database and the default module type. The DSN must be changed to correct testing database DSN. This is built on the adodb abstraction layer and conforms to DBAPI 2. 0, so once you change these entries the rest of the code will need no change. The function `change_db(dsn,module)` also allows interactive or programmatic changing of these defaults (though the change is not stored at this time). `dbhelp()` provides command line help on using the connection commands. `viewfield()`, `insertrow()`, and `deleterow()` are abstraction versions of SQL (so

that the underlying database syntax does not matter to the rest of the code). `insertrow()` will need more switches for other data types, in particular binary large objects (BLOBs) . `viewfield()` and `deleterow()` work fine for all data types. Important note: the date/time data type does not translate correctly for ODBC. You need to put in program lines to generate the correct strings instead of using these functions directly in a query. SELECT queries do work without additional code. As a result, use Julian values in the MValue field for more consistent date/time comparisons.

ARGOS.PY

There is a lot in this module. If you run it directly, it does an automated download of Argos using command line arguments. Notice the `-h` flag provides help on command line arguments, which are in Unix style. The one skeleton function right now is `InsertDatabase`, which should make a call to the functions in `DBTESTING` (after its name is changed) to insert PASS and DATA information into the database. The Call List towards the top of the file should be very helpful. Error catching is implemented throughout with lots of commenting.

ARGOSEXPRESSIONS.PY

The regexp objects used by `argos.py`. Read this one over, especially the part about how to add new dataline formats. I used the list system so

that argos.py would not have to be recoded for new dataline types. The exception is if there are more than 4 data fields on one line. That would require quite a bit of restructuring to argospass.py, argosdata.py, and argos.py. Note though that this all handles multiline raw data perfectly fine as long as there are only 4 fields per line.

ARGOSPASS.PY

Represents the PASS object. Has several important object translating functions. If you want to derive values from information stored in the pass object, use functions in here.

ARGOSDATA.PY

Same concept as ARGOSPASS, except for the DATA object.

TESTENV.PY

This is a lengthy script that tests new code without having to telnet into Argos. I used a flash drive as the main directory for running it. It also shows how the different functions should be used together. Edit line 4 and 5 to reference the location where you have put the test files. These test files are 070716dg. txt (but you can replace with any downloaded raw diag file) and 070716. txt (but you can replace with any download raw prv file). Edit line 6 to reference the directory to save output.

AUTOMATION UTILITIES

- `autorun.py`: Generic script to automatically run another script in the background at a set time interval.
- `autoargos.py`: Uses `Autorun.py` to run `argos.py` every day (downloading the daily Argos data and archiving it). This must be in the same directory as `argos.py` to work correctly, or `argos.py` must be in the system search path. Note that this uses a 86400 second timing interval, not a time of day trigger, so it will always execute a download as soon as it is started. You can run multiple threads with multiple program numbers
- `addstartup.py`: This is the "ON" button. Adds a line to the registry to automatically start `autoargos.py` when the computer is started up. Note that this does not use a program number. You can edit it to do this.
- `delstartup.py`: This is the "OFF" button. Removes the registry line, so that `autoargos.py` is not started on reboot. Note that this does not turn off a running thread of `autoargos.py`. You have to break from the running program to do this.

```

DBTESTING.PY
"""This module provides functions to perform some basic SQL commands"""
import adodb
import sys, datetime
import argospass

#Change this string to change the default target database and database type
#See the adodb documentation for the correct module
#pyodbc is implemented as well
dsnstring = "DSN=mrmttest"
module='odbc'
try:
    conn = adodb.NewADOConnection(module)
    conn.Connect(dsnstring)
except:
    print "Unable to establish a connection with DSN string:",dsnstring
    print sys.exc_info()[2]
    print "SQL statement will not Execute."
else:
    print "Connected to database '%s' with module '%s'." % (dsnstring,module)
    print "See dbhelp() for information on changing this connection."

def dbhelp():
    print "Use change_db(dsn, newmodule) to change to a different database and database type"
    print "or use change_db(dsn) to change to a different database of the same type."
    print "See the adodb documentation for module types and required extensions."

def change_db(dsn,newmodule = module):
    if conn and conn.IsConnected():
        conn.Close()
    if module == newmod:
        conn.Connect(dsn)
    else:
        conn = adodb.NewADOConnection

def viewfield(field,table,where=None):
    """Returns the results from a SELECT query for FIELD from TABLE using WHERE
    criteria"""
    cursor = None
    if conn and conn.IsConnected():
        try:
            if where <> None:
                cursor = conn.Execute('SELECT %s FROM %s WHERE %s' % (field, table, where),)
            else:
                cursor = conn.Execute('SELECT %s FROM %s' % (field,table),)
        except:
            print "viewfield() encountered an error:"
            print sys.exc_info()[1]
            print "SQL Statement:"
            print "SELECT %s FROM %s WHERE %s" % (field, table, where)
    else:
        return cursor

def insertrow(row,table):

```

```

"""Inserts one row into a table. row must be a sequence of values."""
valuelist = []
if conn and conn.IsConnected():
    for entry in row:
        if isinstance(entry, datetime.datetime):
            valuelist.append(conn.DBTimeStamp(entry))
        elif entry == None:
            valuelist.append("")
        else:
            valuelist.append("'" + str(entry) + "'")
    values = ",".join(valuelist) #Comma-delimited values list
    try:
        cursor = conn.Execute('INSERT INTO %s VALUES (%s)' % (table,values),)
    except:
        print "insertrow(%s,%s) not executed." % (row,table)
        print "Statement:"
        print "INSERT INTO %s VALUES (%s)" % (table,values)
        print sys.exc_info()[1]
    else:
        conn.CommitTrans()
else:
    print "Database connection not found."

def deleterow(criteria,table):
    if conn and conn.IsConnected():
        try:
            conn.Execute('DELETE FROM %s WHERE %s' % (table,criteria),)
        except:
            print "deleterow(%s,%s) not executed." % (criteria,table)
            print sys.exc_info()[1]
        else:
            conn.CommitTrans()
    else:
        print "Database connection not found."

class accessdb():
    """This object holds a series of methods for assessing the MMI Arc Marine
    based data repository."""
    def __init__(self):
        self.pttlist=[] #This is the list of active measuring devices
        #Stores tuples in format: (PTT#, program number,DeviceID,VehicleID)
        self.lastupdate = datetime.datetime(1900,1,1)
        self.loadptts()

    def loadptts(self):
        """Loads active PTTs for use by other functions."""
        #Tested
        self.pttlist = []
        p = viewfield('PTT,StartDate,StopDate,ProgramID,DeviceID,VehicleID','MeasuringDevice')
        curr = datetime.datetime.today() - datetime.timedelta(1)
        yest = curr - datetime.timedelta(1)
        for row in p:
            start = conn.TimeStamp(row[1])
            end = conn.TimeStamp(row[2])
            if (start and start <= curr) and (not end or end >= yest):

```

```

        self.pttlist.append((row[0],row[3],row[4],row[5]))
        self.lastupdate = datetime.datetime.today()

def findptt(self,ptt,program = None):
    """Checks if a given PTT and program are active.
    Returns VehicleID for the PTT if active.
    Without program argument, returns program numbers for ptt if active."""
    #Tested
    #If more than one day since last update, loadptts()
    if self.lastupdate < (datetime.datetime.today() - datetime.timedelta(1)):
        self.loadptts()
    if program:
        for p in self.pttlist:
            if ptt == p[0] and program == p[1]:
                return p[3]
    else:
        progs = []
        for p in self.pttlist:
            if ptt == p[0]:
                print "PTT %s found with program %s" % (ptt, p[1])
                progs.append(p[1])
        if len(progs) > 0:
            return progs
    return False

def createorphan(self,passobj):
    #Inserts an orphan tag based on the pass object
    #All vehicleIDs are set to the orphan ID of 0 for now
    #But when written, the orphanID must instead by the vehicleID
    #Created when the orphan is inserted in the db
    orphanID = 0
    return orphanID

def findpass(self,passobj):
    """Checks if a pass object already has been loaded into argosinfo.
    Find records that are the same satellite and device and within one hour"""
    #Tested
    sat = "%s" % (passobj.satellite) #Satellite that received the transmission, lf1 criteria
    #MValues work more consistently with underlying databases

    t1 = (passobj.mvalue-(1.0/24)) #Start of 2 hour window as MValue
    t2 = (passobj.mvalue+(1.0/24)) #End of 2 hour window as MValue
    veh = self.findptt(passobj.PTT,passobj.program) #VehicleID, rf2 criteria
    if not veh:
        veh = self.createorphan(passobj)

    #Build query
    lt = "ArgosInfo" #Left table
    rt = "animalevent" #Right table
    lj = "TelemetryId" #Left join field
    rj = lj #Right join field
    lf1 = "satellite" #Left table criteria field 1
    rf1 = "mvalue" #Right table criteria field 1
    rf2 = "vehicleid" #Right table criteria field 2
    joincrit = "%s.%s = %s.%s" % (lt,lj,rt,rj)

```



```

fields = "*"
tables = "%s INNER JOIN %s ON %s" % (lt,rt,joincrit)
where = "%s.%s = %s AND %s.%s > %s AND %s.%s < %s AND %s.%s=%s" % (lt,lf1,sat,
rt,rf1,t1, rt,rf1,t2, rt,rf2,veh)

print fields
print tables
print where
#Send off the query and get results back as a cursor
cursor = viewfield(fields,tables,where)

#Check if there are any results
if cursor:
    records = []
    for row in cursor:
        records.append(row)
    if len(records) < 1:
        self.insertpass(passobj)
    print records
else:
    print "No cursor"
    #if there are results,
    #if no results, return None
    #If no cursor, there was an error somewhere, report this

#This part might be complex
#Three match types (first on data):
#All old and some new: Add new sensor events and check timedate on old
#All old and all new: Check timedatas on old
#Some unmatched old: Fail, a new argosinfo will be built and all new sensor
# events, old data is untouched
def finddata():
    #Based on found argosinfo, find matching data objects
    pass
def updatedata():
    #Update old sensor events. Very complex and needs to return result
    pass
def insertdata():
    #insert a new sensorevent based on based data
    pass

def insertpass(self,passobj):
    veh = self.findptt(passobj.PTT,passobj.program)    #VehicleID
    if not veh:
        veh = self.createorphan(passobj)
    aerow = [passobj.timevalue,passobj.mvalue,1,veh]
    insertrow(aerow,'AnimalEvent (timevalue,mvalue,eventtype,vehicleid)')
    conn.CommitTrans()
    where = "MValue = %s AND EventType = %s AND VehicleId = %s" %
(aerow[1],aerow[2],aerow[3])
    row = viewfield('*', 'animalevent',where).FetchRow()
    telemetryid = row[0]
    airow = passobj.insertSQL()
    airow.append(telemetryid)

```

```

        fields =
"satellite,lc,iqa,iqb,lat1,lon1,lat2,lon2,Nb_mes,Nb_120dB,Best_level,pass_dur,nopc,freq,telemetry
id"
        insertrow(airow,'ArgosInfo (%s)' % fields)
        conn.CommitTrans()
        #Insert the AnimalEvent and ArgosInfo
        #Make calls to insert the SensorEvents without any checks

try:
    acc = accessdb()
except:
    print "Unable to create database access object."
    print sys.exc_info()[2]
    print "Arc Marine database access functions will not be enabled."

```

ARGOS.PY

```

"""
#-----
# Tool Name: Download ARGOS data
# Source Name: argos.py
# Version: ArcGIS 9.2
# Author: Brett Lord-Castillo, lordcasb@onid.orst.edu
#
# This tool downloads ARGOS tracking data and loads it into a geodatabase
# constructed with the ArcMarineDataModel with MMI tracking customization
#
# Current status: Not integrated with GeoEco metadata checking
#                 Download: Complete, executes from main
#                 Archiving: Complete, executes from main
#                 Parsing: Complete, does not execute in main
#                 Create Objects: Complete, does not execute in main
#                 DB Insert: Partially written, see DBTESTING
#-----
"""

#-----
#Imports
#-----

try:
    from GeoEco.ArcGIS import GeoprocessorManager
    from GeoEco.DynamicDocString import DynamicDocString
    from GeoEco.Internationalization import _
    from GeoEco.Logging import Logger
except ImportError:
    print "Failed to import GeoEco modules."
    print sys.exc_info()[2]
    print "Proceeding, but may cause errors."
from telnetlib import Telnet
from time import strptime
import os, csv, datetime, sys
expressions_loaded = 1
try:
    from argosexpressions import *
except:

```

```

    print "Module argosexpressions is not available."
    print "Not able to load regexp definitions for parsing."
    print "Downloading functions still available."
    expressions_loaded = 0
argosobjects_loaded = 1
try:
    from argospass import PASS
except:
    print "Module argospass is not available."
    print "Cannot load to database."
    print "Downloading functions still available."
    argosobjects_loaded = 0
try:
    from argosdata import DATA
except:
    print "Module argosdata is not available."
    print "Cannot load to database."
    print "Downloading functions still available."
    argosobjects_loaded = 0

#-----
# Call List:
#   Call InitDownload
#       Calls ConnectTelnet or ConnectSSH
#       Returns conn object
#   Call Download with conn object and date object for start date
#       Downloads the text files
#       Returns directory to file
#   Call Cleanfile with path
#       Returns path to cleaned file (same file)
#   Call Parse with filepath to send the file off to be parsed
#       Returns a list
#       Three elements: DS headers, DS datalines, and garbage
#       Two elements: DG blocks, garbage
#   Call PairedParse with DB blocks, DS Headers, and DS datalines
#       Return paired blocks as [dg,ds,[dsdata]], unmatched dg, and unmatched ds
#   Call GenerateObjects with paired blocks from PairedParse
#       Returns an array of dB ready objects
#   Call InsertObjects
#       Insert the data into the dB, doing redundancy checks
#

class DownloadARGOS(object):
    """Method holding object. Create a DownloadARGOS object to access class methods.
    for the download and parsing of Argos data. Raw data is only stored, not decoded."""
    try:
        __doc__ = DynamicDocString()
    except:
        print "Unable to use GeoEco DynamicDocString."

    @classmethod
    def InitDownload(cls, user, password, host = "datadist.argosinc.com", port = 23, method =
"telnet"):
        #Status: Add Metadata
        """Allows switching between download connection methods. This is most useful to

```

```

allow the implementation of SSH or WWW protocols as those become available."""
if method == "telnet":
    conn = cls.ConnectTelnet(user, password, host, port)
elif method == "SSH":
    #SSH method is not implemented yet
    conn = cls.ConnectSSH(user, password, host, port)
#Add more methods here
else:
    print "Undefined connection method request passed to InitDownload()"
    print "Undefined method:", method
    print "Continuing with no connection..."
    return None
return conn

@classmethod
def ConnectTelnet(cls, sUser, sPassword, sHost, dPort):
#Status: Add Metadata
    """Uses a telnet object to connect to an ARGOS host using passed arguments
    Returns a telnet connection object."""
    try:
        #Open telnet connection and send username and password
        tn = Telnet(sHost, dPort)
        tn.read_until("Username:", 10)
        tn.write('%s\r' % (sUser))
        tn.read_until("Password:", 10)
        tn.write('%s\r' % (sPassword))
        #Wait for Argos to respond
        tn.expect(["ARGOS READY",],10)
    except:
        print "Error in opening telnet connection"
        print "User: ", sUser
        print "Host: ", sHost
        print "Port: ", dPort
        print sys.exc_info()[2]
        sys.exit()
    return tn

@classmethod
def ConnectSSH(cls, args):
    """#Status: Unwritten, not in use and should not be called"""
    print "SSH method not defined"
    print "Cannot use SSH connections at this time"
    sys.exit("Illegal function call")
    #SSH connection method will go here

@classmethod
def Download(cls, connection, directory, program = "",
             start=(datetime.datetime.today()-datetime.timedelta(1))):
    """Uses a connection to an Argos host to download results of PRV and DIAG commands.
    This should be accessed programmatically."""
    #Status: Metadata
    #Start must be a date object, defaults to yesterday"""

    #Build command strings
    calldate = datetime.datetime.today()          #Date command is being called

```

```

directory = os.path.abspath(directory)      #Archive directory
if not os.path.exists(directory):
    sys.exit("Output directory does not exist")
try:
    dayofyear = str(start.timetuple()[7])      #Day number for command string
except AttributeError:
    raise TypeError, "Argument 'start' must be <type 'datetime.datetime'>, not '%s'" %
type(start)

#ARGOS command strings. Could add more types of commands here or add switching
sendPRV = 'PRV/A,%s,DS,%s,\r' % (program, dayofyear)
sendDIAG = 'DIAG,%s,%s,\r' % (program, dayofyear)

#Define output files and check if they exist already
if abs(calldate - start) < datetime.timedelta(2,1):
    #If start date is yesterday, base name on today's date
    #e.g. '070131' for a call on 01/31/2007 for data on 01/30/2007
    outname = calldate.strftime("%y%m%d")
else:
    #Otherwise, used combined date format
    #e.g. '070131_070128' for a call on 01/31/2007 for data on 01/28/2007
    outname = '%s_%s' % (calldate.strftime("%y%m%d"),start.strftime("%y%m%d"))
pathprv = os.path.abspath('%s\\%s.txt' % (directory,outname))
pathdiag = os.path.abspath('%s\\%sdbg.txt' % (directory,outname))

#Download the DS file
try:
    if os.path.exists(pathprv):
        print "File %s already exists.\nOutput will be appended to this file." % pathprv
        outprv = open(pathprv,'a')
    else:
        print "File %s does not exist.\nCreating file." % pathprv
        outprv = open(pathprv,'w')
        outprv.write("DS\n")
except IOError:
    sys.exit("Unable to open output file: ", pathprv)
try:
    connection.write(sendPRV)
    outprv.write( connection.read_until("ARGOS READY", 30) )
except EOFError:
    print "Connection terminated before PRV/A command finished execution."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
finally:
    outprv.close()

#Download the DIAG file
try:
    if os.path.exists(pathdiag):
        print "File %s already exists.\nOutput will be appended to this file." % pathdiag
        outdiag = open(pathdiag,'a')
    else:
        print "File %s does not exist.\nCreating file." % pathdiag

```

```

        outdiag = open(pathdiag,'w')
        outdiag.write("DIAG\n")
    except IOError:
        sys.exit("Unable to open output file: ", pathdiag)
    try:
        connection.write(sendDIAG)
        outdiag.write( connection.read_until("ARGOS READY", 30) )
    except EOFError:
        print "Connection terminated before DIAG command finished execution."
    except:
        print "Unexpected error:", sys.exc_info()[0]
        raise
    finally:
        outdiag.close()

#Clean up
try:
    connection.write("LOGOUT\r")
    connection.close()
except:
    print "Unexpected error in closing connection:", sys.exc_info()
    print "Attempting to continue..."
return [pathprv,pathdiag]

@classmethod
def CleanFile(cls, infile):
    """Removes bad characters and extra line feeds from downloaded Argos output."""
    #Status: Metadata
    outdir = os.path.dirname(os.path.abspath(infile))
    outfile = os.path.basename(os.path.abspath(infile))
    #Open up output file to be cleaned
    try:
        editfile = open(infile, 'r')
    except IOError:
        print "Unable to open input file: %s" % infile
        raise

    #Open temporary ~ file to hold cleaned output
    try:
        cleanfile = open('%s/~%s' % (outdir,outfile),'w')
    except IOError:
        print "Unable to open temporary file: %s/~%s" % (outdir,outfile)
        editfile.close()
        raise
    #-----
    #Write cleaned text to ~ file
    try:
        text = editfile.read()
    except:
        print "Error reading input from %s." % infile
        editfile.close()
        cleanfile.close()
        raise

    llen = 0          #Line length

```

```

white = 1          #All characters in line are whitespace flag
newline = ""       #Stores line to be written

try:
    #Read character by character and write line by line
    for char in text:
        if ord(char)==10 or ord(char)==13:    #End of line encountered
            #Only write lines of more than 1 character and with non-whitespace characters
            if llen > 1 and white==0:
                newline = '%s\n' % newline
                if newline[0] == '/':
                    newline = newline [1:]
                cleanfile.write(newline)
                white = 1
            llen = 0
            newline = ""
        else:                                #End of line not encountered
            newline = '%s%s' % (newline,char) # Add character to line
            if ord(char)<>32: white = 0        # If character is not whitespace, unset white flag
            llen = llen + 1                  # Increment line length

    #Write last line if not blank
    if newline <> "":
        newline = '%s\n' % newline
        cleanfile.write(newline)
except:
    print "Problem encountered during file cleaning."
    raise
finally:
    #Close files in use
    editfile.close()    #Reading from
    cleanfile.close()   #Temporary write to
#-----

#-----
#Copy from temporary file to input file
try:
    copyfrom = open('%s//~%s' % (outdir, outfile),'r')
    copyto = open(infile,'w')
except:
    print "Unable to copy from temporary file %s//~%s to %s." % (outdir,outfile, infile)
    raise

try:
    copyto.write(copyfrom.read())
except:
    print "Unable to copy from temporary file %s//~%s to %s." % (outdir,outfile,infile)
    print "Attempting to continue with temporary file"
    infile = ""

copyfrom.close()
copyto.close()
#-----

#Cleanup and return path to cleaned file

```

```

if infile:
    try:
        os.remove('%s/~%s' % (outdir, outfile))
    except OSError:
        print "Failed to remove temporary file %s/~%s." % (outdir,outfile)
    return infile
else:
    #Copy from temporary file failed, so return temporary file.
    return '%s/~%s' % (outdir, outfile)

@classmethod
def Parse(cls, infile):
    """Switches the input off to the correct parsing functions.
    Takes a returned array which is outputted to the correct CSV types
    and then puts that data into the correct location."""
    #Status: Metadata

    #Check to make sure argosexpressions.py has been loaded.
    if not expressions_loaded:
        raise ImportError, "Module argosexpressions not loaded. Cannot parse downloaded text."

    #Open the file to be parsed
    try:
        file_type = open(infile, 'r')
    except IOError:
        print "Parse() could not open input file:", infile
        raise

    try:
        #Read in the first line which defines the program output type (DS or DIAG)
        test_type = file_type.readline()

        #parselist stores the lines to be sent to the parsing functions
        parselist = []

        #Read in the file
        for line in file_type.readlines():
            parselist.append(line)          #Append any non-termination line
    except IOError:
        print "IOError encountered while reading file %s." % infile
        if len(parselist) > 0:
            #If any lines were received, attempt to parse them
            print " Attempting to continue..."
        else:
            print " Exiting..."
            raise
    finally:
        #Always close the file
        file_type.close()

    #Switch to send file to the correct parser
    if test_type == "DS\n":
        return cls.ParseDS(parselist)
    elif test_type == "DIAG\n":
        return cls.ParseDIAG(parselist)

```



```

#If the right type is not found, cannot parse this file
print 'First line of file must be "DS" or "DIAG".'
print 'Unable to read command type. Cannot parse input file.'
return []

@classmethod
def ParseDS(cls, inlist):
    """Should only be called from Parse(). Used to parse PRV command output."""
    #Status: comment, metadata, trap errors

    #Variable List
    hf = 0          # Header Flag
    ff = 0          # Footer Flag
    hcnt = 0        # Header count variable
    #Variables for header/footer information
    headers = ['header','program','ptt','numlines','satellite',
               'LC','date','yr','mon','day','time','hr','min','sec',
               'lat','lon','Z','freq','NumMsg','msgs>-120dB',
               'Best','Freq','IQ',
               'lat1','lon1','lat2','lon2','dateobj']
    #lines start with fields: 0,5,14,20,23

    #Variables for data lines
    dataheaders = ['header','date','year','month','day',
                  'time','hour','minute','second',
                  'passes','data1','data2','data3','data4','dateobj']
    #lines start with fields: 0,5,9

    #First line of each array is variable names
    wDS = [headers]
    wDSData = [dataheaders]
    garbage = []
    dsline = []
    dataline = []
    lastatts = [",",",",",",",",",",","]

    if len(inlist) < 1:
        #If empty, only return headers
        return [wDS,wDSData,garbage]
    timestamp = ""
    date_first = ""
    date_last = ""

    for line in inlist:
        if ff == 1:          #If in a footer block
            footer2 = fPRVb.match(line)
            if footer2:
                dsline.extend([footer2.group('lat1'),footer2.group('lon1'),
                              footer2.group('lat2'),footer2.group('lon2')])
            else:
                #No second line of footer
                dsline.extend(["",",",","])
            ff = 0
            if not timestamp:

```

```

        timestamp = date_first
        if date_last:
            date_mid = abs(date_last-date_first)/2
            date_avg = datetime.timedelta(date_mid.days,date_mid.seconds)
            timestamp = date_first + date_avg
        dsline.extend([timestamp])
        wDS.append(dsline)
    else:
        header1 = hPRV.match(line)
        footer1 = fPRVa.match(line)
        if header1:
            if hf == 1:
                #Already in an unterminated block
                #Most likely a one message block
                #Pad out dsline to 28 columns and append
                for n in range(len(dsline),28):
                    dsline.extend([""])
                wDS.append(dsline)
            #Encountered start of block
            hcnt = hcnt + 1
            hf = 1
            date_first = ""
            date_last = ""
            header2 = hPRVf.match(line)
            #Start new ds entry
            #All headers have this information
            dsline = [hcnt,header1.group('program'),header1.group('PTT'),
                    header1.group('lines'),header1.group('satellite')]
            if header2:
                #Only full headers have this information
                dsline.extend([header2.group('locclass'),header2.group('date'),
                            header2.group('yr'),header2.group('mon'),header2.group('day'),
                            header2.group('time'),header2.group('hr'),header2.group('min'),
                            header2.group('sec'),header2.group('lat'),header2.group('lon'),
                            header2.group('Z'),header2.group('freq')])
                timestamp = datetime.datetime(int(header2.group('yr')),int(header2.group('mon')),
                                            int(header2.group('day')),int(header2.group('hr')),
                                            int(header2.group('min')),int(header2.group('sec')))
            else:
                #Pad with blanks if not a full header
                dsline.extend(["","","","","","","",""])
        elif footer1:
            #Encountered end of block
            #Reset data line attributes
            lastatts = ["","","","","",""]
            ff = 1
            hf = 0
            dsline.extend([footer1.group('msgs'),footer1.group('dB'),footer1.group('best')])
            dsline.extend([footer1.group('freq'),footer1.group('iqx')+footer1.group('iqy')])
        elif hf == 1:
            #Begin dataline matching
            dataline = [hcnt]
            for datatest in dataopts:
                result = datatest[0].match(line)
                has_attributes = datatest[1]

```

```

numgroups = datatest[2]
if result:
    if has_attributes:
        #If there are attributes, use those values
        lastatts = [result.group('date'),result.group('yr'),
                    result.group('mon'),result.group('day'),
                    result.group('time'),result.group('hr'),
                    result.group('min'),result.group('sec'),
                    result.group('passes')]
        date_temp = datetime.datetime(int(result.group('yr')),
                                      int(result.group('mon')),
                                      int(result.group('day')),
                                      int(result.group('hr')),
                                      int(result.group('min')),
                                      int(result.group('sec')))

        if not date_first:
            if int(result.group('passes')) == 1:
                date_first = date_temp
            else:
                date_first = date_temp -
datetime.timedelta(0,10*int(result.group('passes')))
        else:
            date_last = date_temp
    else:
        lastatts = [",",",",",",",",",",","] #Comment out this line to use attributes of last
data line
        dataline.extend(lastatts)
        n = 0
        while n < numgroups:
            dgrp = 'data%s' % str(n+1)
            dataline.extend([result.group(dgrp)])
            n = n + 1
        while n < 4:
            dataline.extend([""])
            n = n + 1
        #If we find a result, stop testing data formats
        break
    if len(dataline):
        dataline.extend([date_temp])
        wDSData.append(dataline)
    else:
        #If there was no data line match, then line is garbage
        garbage.append(hcnt,line)
    else:
        #Encountered garbage outside a header block, so ref to last header
        garbage.append([hcnt,line])
        #Increment and move to next line
return [wDS,wDSData,garbage]
#-----

@classmethod
def ParseDIAG(cls, inlist):
    """Should only be called from Parse(). Used to parse DIAG command output.
    #Status: Comments, Metadata, Error trapping"""
    #Variable List

```

```

lc = 0          # Line Counter
hf = 0          # Header Flag
hcnt = 0        # Header count variable
garbage = []

#Variable names for header, line 1, line 2, line 3, line 4, and data section
columnheaders = ['header','ptt','date','yr','mon','day',
                  'time','hr','min','sec','lc','iq',
                  'lat1','lon1','lat2','lon2',
                  'nbmsg','dB','best','passdur','nope',
                  'fq','altitude','data1','data2','data3','data4','dateobj']
#Lines start with fields: 0,6,12,16,21

wDiag = [columnheaders]          #First line of array is variable names

#Trap for zero length lists
if len(inlist) < 1:
    return wDiag                #Only return header list

#Parse and create array
curline = 0                     #Current line (1-5) in DIAG block
err = 0                         #Error counter
timestamp = ""
for line in inlist:
    header = hDIAG.match(line)
    #If header line, move to line 1
    if header:
        hcnt = hcnt + 1 #Increment header count
        hf = 1         #Inside a DIAG block
        curline = 1     #Move to next line
        err = 0         #Init error count
        parms = [hcnt,header.group('ptt'),header.group('date'),
                  str(2000+int(header.group('yr'))),header.group('mon'),
                  header.group('day'),header.group('time'),header.group('hr'),
                  header.group('min'),header.group('sec'),header.group('lc'),
                  header.group('iq')]
        timestamp = datetime.datetime(int(parms[3]),int(parms[4]),int(parms[5]),
                                       int(parms[7]),int(parms[8]),int(parms[9]))

    #If header done, add line 1 data to parms list if present and move to next line
    elif curline == 1:
        line1 = lDIAG1.match(line)
        line1a = lDIAG1a.match(line)
        if line1:
            parms.extend([line1.group('lat1'),line1.group('lon1'),
                          line1.group('lat2'),line1.group('lon2')])
        elif line1a:
            parms.extend([line1a.group('lat1'),line1a.group('lon1'),
                          line1a.group('lat2'),line1a.group('lon2')])
        else:
            #Line 1 not found, add blanks, increment error count
            print "Line 1 Missing!"
            parms.extend(["", "", ""])
            err = err + 1
        curline = 2     #Move to next line

```

```

#If line 1 done, add line 2 data to parms list if present and move to next line
elif curline == 2:
    line2 = IDIAG2.match(line)
    if line2:
        parms.extend([line2.group('nbmsg'),line2.group('dB'),line2.group('best')])
    else:
        #Line 2 not found, add blanks, increment error count
        print "Line 2 Missing!"
        parms.extend(["","])
        err = err + 1
    curline = 3    #Move to next line

#If line 2 done, add line 3 data to parms list if present and move to next line
elif curline == 3:
    line3 = IDIAG3.match(line)
    line3a = IDIAG3a.match(line)
    if line3:
        parms.extend([line3.group('passdur'), line3.group('nopc')])
    elif line3a:
        parms.extend([line3a.group('passdur'), line3a.group('nopc')])
    else:
        #Line 4 not found, add blanks, increment error count
        print "Line 3 Missing!"
        parms.extend(["","])
        err = err + 1
    curline = 4    #Move to next line

#If line 3 done, add line 4 data to parms list if present and move to next line
elif curline == 4:
    line4 = IDIAG4.match(line)
    if line4:
        parms.extend([line4.group('fq1')+line4.group('fq2'),line4.group('altitude')])
    else:
        #Line 4 not found, add blanks, increment error count
        print "Line 4 missing!"
        parms.extend(["","])
        err = err + 1
    curline = 5    #Move to next line

#If line 4 done, add the data line to parms list if present and end block
elif curline == 5:
    lineh = dDIAGh.match(line)
    lined = dDIAGd.match(line)
    if lineh:
        #Four fields found
        parms.extend([lineh.group('data1'),lineh.group('data2'),
                      lineh.group('data3'),lineh.group('data4')])
    elif lined:
        #Two fields found
        parms.extend([lined.group('data1'),lined.group('data2'),
                      "",])
    else:
        #No data found, add blanks, increment error count
        parms.extend(["",",","])
        print "No Data Line!"
        err = err + 1
    #Write the block to the output file and reset
    parms.extend([timestamp])
    wDiag.append(parms)
    hf = 0
    curline = 0

```

```

#If data encountered outside the DIAG block, skip it, we will only match first four fields
elif curline == 0:
    garbage.append([line])          #Extra lines captured as garbage
    if dDIAGh.match(line):
        pass                       #Extra dataline(4 fields), can capture here
    elif dDIAGd.match(line):
        pass                       #Extra dataline(2 fields), can capture here
    else:
        pass                       #Other line outside block

if err > 0:
    print "Total DIAG Errors: ", err
#Return array of parsed data
return [wDiag, garbage]

@classmethod
def PairedParse(cls, dglist, dslist, dtlist):
    """Matches output from ParseDS and ParseDIAG to produce paired array
    #Status: Tested, needs commenting"""

    paired = []
    #paired is structured: header number,dg block,ds block,data line list
    unmatcheddg = [dglist[0]]
    unmatchedds = [dslist[0]]
    if len(dslist) > 1:
        maxds = dslist[-1][0]
    else:
        maxds = 0

    #Load DS blocks
    for i in range(maxds+1):
        paired.append([i,"",dslist[i],[]])
    #Load headers
    paired[0][0]="Header Number"
    paired[0][1]=dglist[0]
    paired[0][2]=dslist[0]
    paired[0][3]=dtlist[0]

    #Load data lines
    dtread = dtlist[1:]
    for line in dtread:
        paired[line[0]][3].append(line)

    #Load DIAG blocks
    dgread = dglist[1:]
    for line in dgread:
        #line is the dgblock loaded
        #hdr is the header number of this DIAG block
        hdr = line[0]
        #dstest is the DS block loaded for this header
        dstest = paired[hdr][2]
        #Match on data line
        dgdata = ".join(line[23:27])          #data1+data2+data3+data4
        dsdata = []

```

```

for i in paired[hdr][3]:
    dsdata.append(" ".join(i[10:14]))    #data1+data2+data3+data4
dsdata.append(dgdata)
if dsdata.index(dgdata) == len(dsdata)-1:
    #No data line match, fail
    unmatcheddg.append(line)
else:
    #Data line match, now match on date
    if not dstest[27]:
        paired[hdr][1] = line
    elif abs(line[27]-dstest[27])<datetime.timedelta(0,30,1):
        #Match if time difference is 30 seconds or less
        paired[hdr][1] = line
    else:
        #Fail to match
        unmatcheddg.append(line)

#Find unmatched DS blocks
for block in paired:
    if block[1]:
        #DS block has a DG block
        pass
    else:
        #DS block has no match
        unmatchedds.append(block[2])
        paired.remove(block)
return [paired, unmatcheddg, unmatchedds]

@classmethod
def GenerateObjects(cls, paired):
    """Not written. Generates initial matched objects from PairedParse arrays."""

    #Paired is the paired list returned by PairedParse
    #-----
    #Here is where the information comes from:
    #paired[i][d][f]
    #i is the argos header number
    #t is the dataset 1=DIAG 2=DS 3=DS datalines
    #f is the field in the dataset. See the header lists in those parsers
    #entry[d][f] corresponds to the above for a specific header entry

    #Check to make sure argospass.py and argosdata.py have been loaded.
    if not argosobjects_loaded:
        raise ImportError, "Modules argospass and argosdata required for database loading. Cannot
continue."

    dbloader = []    #This list will eventually be loaded into the db
                    #Structure: [ [PASS1,[DATA,...]], [PASS2,[DATA,...]] ]
    paireddata = paired[1:]    #Strip off the headers
    for entry in paireddata:
        dbline = []    #Line to be added, structured as [PASS,[DATA,DATA...]]
        #Start by building two dictionaries based off the headers
        #This way header references can be used even if the paired list
        #structure changes later
        dg=dict(zip(paired[0][1],entry[1]))

```

```

ds=dict(zip(paired[0][2],entry[2]))
try:
    #LOAD FROM DS (unless blank, then from DG)
    #Load program, ptt, timevalue, and satellite
    #PASS object handles timevalue to timestamp transformation internally
    dbpass = [ds['program'],ds['ptt']]
    if ds['dateobj']:
        dbpass.append(ds['dateobj'])
    else:
        dbpass.append(dg['dateobj'])
    if ds['satellite']:
        dbpass.append(ds['satellite'])
    else:
        dbpass.append(dg['satellite'])

    #LOAD FROM DIAG
    #LC domain and IQ split handled by PASS object
    dbpass.extend([dg['lc'],dg['iq']])
    #Load lat/lon values
    dbpass.extend([dg['lat1'],dg['lon1'],dg['lat2'],dg['lon2']])
    #Load Nb messages, Nb>-120dB, and best level
    dbpass.extend([dg['nbmsg'],dg['dB'],dg['best']])
    #Load Pass duration, NOPC, and frequency
    dbpass.extend([dg['passdur'],dg['nopc'],dg['fq']])
except KeyError, missing:
    print "No entry %s in 'paired' list. Entry %s is expected for creation of PASS objects."
% (missing,missing)
    raise

try:
    #Make the PASS object
    passobj = PASS.initlist(dbpass)
except:
    print "Could not create PASS object with initlist()."
    raise
dbline.extend([passobj,[]])
try:
    #Make the DATA objects
    for dataline in entry[3]:
        #Create dictionary based on dataline headers
        dt=dict(zip(paired[0][3],dataline))
        rawdata='%s%s%s%s' % (dt['data1'],dt['data2'],dt['data3'],dt['data4'])
        if not dt['passes']:
            dbline[1][-1].raw = '%s%s' % (dbline[1][-1].raw,rawdata)
        else:
            dataobj=DATA(dt['dateobj'],dt['passes'],passobj.passid,rawdata)
            dbline[1].append(dataobj)
    dbloader.append(dbline)
except:
    print "Could not create DATA object."
    raise
return dbloader

@classmethod
def InsertDatabase(cls, sequence, connection):

```



```

        """Takes an sequence of objects and inserts into Arc Marine.
        Sequence must be in this form:
        [[PASS,[DATA,...]], [PASS,[DATA,...]]]
        connection is a DB-API 2.0 connection object.
        """
        pass

    @classmethod
    def WriteCSV(cls, datalist, filename):
        """Writes a sequence to a CSV file. Appends if the file already exists.
        datalist    sequence to be written
        filename    file to write to"""
        #Status: Comments, error trapping
        filename = os.path.abspath(filename)
        if os.path.exists(filename):
            wtr = csv.writer(open(filename,'ab'))
        else:
            wtr = csv.writer(open(filename,'wb'))
        wtr.writerows(datalist)
        del wtr
        return filename

#-----

#-----
#MAIN
import getopt
def main():
    """Allows argos.py to be called for automated download.
    Use argos.py -h for command line options.

    argos.py [-h] [-p program] username password directory [startdate]
    -h This text
    -p Specify a program number
    -d Specify a download start date, must be mm/dd/yyyy format
    username ARGOS system username
    password ARGOS system password
    directory Specifies the location of text downloads
    startdate Start date for download, must be mm/dd/yyyy format
           Defaults to yesterday (according to local time)"""

    try:
        opts, args = getopt.getopt(sys.argv[1:], 'hp:', ['help'] )
    except getopt.GetoptError:
        usage()
        sys.exit(2)
    program = "
    for o,a in opts:
        if o in ("-h", "--help"):
            usage()
            sys.exit()
        if o == '-p':

```

```

        program = str(a)
    print "Beginning execution."
    if len(args)<3:
        print "Test execution ended."
        sys.exit("Test.")
    user = args[0]
    password = args[1]
    directory = args[2]
    directory = os.path.abspath(directory)
    if not os.path.exists(directory):
        sys.exit("Output path does not exist")
    if len(args)<4:
        startdate = datetime.datetime.today() - datetime.timedelta(1)
    else:
        try:
            start =.strptime(args[3],"%m/%d/%Y")
        except ValueError:
            sys.exit("Date must be in mm/dd/yyyy format.")
        startdate = datetime.datetime(*start[0:6])
        limit = datetime.datetime.today() - datetime.timedelta(9)
        if startdate < limit:
            sys.exit("Data not available before " + limit.ctime())
    print "Downloading at:", startdate.ctime()
    da = DownloadARGOS()
    files = da.Download(da.InitDownload(user,password),directory,program,startdate)
    print "Outputting to:"
    print files
    for entry in files:
        parseout = da.Parse(da.CleanFile(entry))
        if len(parseout) == 2:
            dg,dggarb = parseout
        elif len(parseout) == 3:
            ds,dt,dsgarb = parseout
        else:
            print "%s lists returned by Parse(). Expected 2 (DIAG) or 3 (PRV)." % len(parseout)
            return 0
    print "Output to text archives complete."
    print "Dumping garbage and creating database loader."
    try:
        da.WriteCSV(dggarb,"%s\\garbage.csv" % directory)
        da.WriteCSV(dsgarb,"%s\\garbage.csv" % directory)
    except:
        print "Unable to write to garbage files"
    paired,unmatchdg,unmatchds = da.PairedParse(dg,ds,dt)
    try:
        da.WriteCSV(unmatchdg,"%s\\unmatcheddg.csv" % directory)
        da.WriteCSV(unmatchds,"%s\\unmatchedds.csv" % directory)
    except:
        print "Unable to output unmatched headers."
    finaloutput = da.GenerateObjects(paired)
    print finaloutput

def usage():
    """Command line help for argos.py"""

```

```

print "help ARGOS"
print "Downloads and text archives argos satellite telemetry results\n"
print "argos.py [-h] [-p program] username password directory [startdate]"
print "  -h This text"
print "  -p Specify a program number"
print "  -d Specify a download start date, must be mm/dd/yyyy format"
print "  username ARGOS system username"
print "  password ARGOS system password"
print "  directory Specifies the location of text downloads"
print "  startdate Start date for download, must be mm/dd/yyyy format"
print "           Defaults to yesterday (according to local time)"

if __name__ == "__main__":
    main()

#Used for automated download
#=====
=====

ARGOEXPRESSIONS.PY
#-----
#Regular expressions for the parsing of downloaded ARGOS output
#-----

#-----
#Regular Expression variables (in verbose descriptions)
#-----
import re
#-----
# PRV(DS) Expressions

""" New PRV data line formats must be added in the correct section below
and a new entry must be created in the dataopts array
matchobject: the variable name for the expression
has_attributes: whether or not the match object has date, time,
and passes attributes
number of data groups: how many data groups (up to four) need to be
parsed; groups less than 4 are padded with blanks
"""

hPRV = re.compile("""
^ (?P<program>\d{5})\s          #Any Header
  (?P<PTT>\d{5})\s            # Program Number
  (?P<lines>..\d)\s           # PTT
  (?P<num>..\d)\s             # Num Lines
  (?P<satellite>\w)\s         # num of bytes
                              # Satellite
""", re.VERBOSE)

hPRVf = re.compile("""
^ (?P<program>\d{5})\s          #Full Header
  (?P<PTT>\d{5})\s            # Program Number
  (?P<Lines>..\d)\s           # PTT
  ..\d\s                      # Num Lines
  (?P<Satellite>\w)\s         # num of bytes
                              # Satellite

```

```
(?P<locclass>.)\s                # Location Class
(?P<date>(P<yr>\d{4})-(P<mon>\d{2})-(P<day>\d{2}))\s  # Date
(?P<time>(P<hr>\d{2}):(P<min>\d{2}):(P<sec>\d{2})))\s*  # Time
(?P<lat>\d{2}.\d{3})\s*          # Lat in dec deg
(?P<lon>\d{2,3}.\d{3})\s*        # Lon in dec deg
(?P<Z>\d{1,2}.\d{3})\s*         # Z in m
(?P<freq>\d{9})                  # Frequency in Hz
""", re.VERBOSE)
```

#Begin data line formats here

#New formats must be added to dataopts list below

```
dPRVda = re.compile(r""           #Data Line decimal
^ \s*                             # Leading whitespace
(P<date>(P<yr>\d{4})-(P<mon>\d{2})-(P<day>\d{2}))\s  # Date
(P<time>(P<hr>\d{2}):(P<min>\d{2}):(P<sec>\d{2})))\s+  # Time
(P<passes>\d{1,2})\s+             # Passes
(P<data1>\d{2,5})\s+              # Decimal Data
(P<data2>\d{2,5})\s+              # Decimal Data
""", re.VERBOSE)
```

```
dPRVdb = re.compile(r""           #Data Line decimal
^ \s*                             # Leading whitespace
(P<date>(P<yr>\d{4})-(P<mon>\d{2})-(P<day>\d{2}))\s  # Date
(P<time>(P<hr>\d{2}):(P<min>\d{2}):(P<sec>\d{2})))\s+  # Time
(P<passes>\d{1,2})\s+             # Passes
(P<data1>\d{1,3})\s+              # Decimal Data
(P<data2>\d{1,3})\s+              # Decimal Data
(P<data3>\d{1,3})\s+              # Decimal Data
(P<data4>\d{1,3})                 # Decimal Data
""", re.VERBOSE)
```

```
dPRVdc = re.compile(r""           #Data Line decimal
^ \s*                             # Leading whitespace
(P<data1>\d{1,3})\s+              # Decimal Data
(P<data2>\d{1,3})\s+              # Decimal Data
(P<data3>\d{1,3})\s+              # Decimal Data
(P<data4>\d{1,3})                 # Decimal Data
""", re.VERBOSE)
```

```
dPRVha = re.compile(r""           #Data Line hex format 1
^ \s*                             # Leading whitespace
(P<date>(P<yr>\d{4})-(P<mon>\d{2})-(P<day>\d{2}))\s  # Date
(P<time>(P<hr>\d{2}):(P<min>\d{2}):(P<sec>\d{2})))\s+  # Time
(P<passes>\d{1,2})\s+             # Passes
(P<data1>[A-F0-9]{2})\s+          # Hexidecimal Data
(P<data2>[A-F0-9]{2})\s+          # Hexidecimal Data
(P<data3>[A-F0-9]{2})\s+          # Hexidecimal Data
(P<data4>[A-F0-9]{2})             # Hexidecimal Data
""", re.VERBOSE)
```

```
dPRVhb = re.compile(r""           #Data Line hex format 2
^ \s*                             # Leading whitespace
(P<date>(P<yr>\d{4})-(P<mon>\d{2})-(P<day>\d{2}))\s  # Date
```

```

(?P<time>(P<hr>\d{2}):(?P<min>\d{2}):(?P<sec>\d{2}))\s+ # Time
(?P<passes>\d{1,2})\s+ # Passes
(?P<data1>[A-F0-9]{4})\s+ # Hexidecimal Data
(?P<data2>[A-F0-9]{4}) # Hexidecimal Data
""", re.VERBOSE)

dPRVhc = re.compile("""
^\s* #Extra hex data
# Leading whitespace
(?P<data1>[A-F0-9]{2})\s+ # Hexidecimal Data
(?P<data2>[A-F0-9]{2})\s+ # Hexidecimal Data
(?P<data3>[A-F0-9]{2})\s+ # Hexidecimal Data
(?P<data4>[A-F0-9]{2}) # Hexidecimal Data
""", re.VERBOSE)

#Option format is: matchobject, has_attributes, number of data groups
#Groups without attributes must come at end of list
dataopts = [[dPRVdb, 1, 4],
             [dPRVha, 1, 4],
             [dPRVhb, 1, 2],
             [dPRVda, 1, 2],
             [dPRVhc, 0, 4],
             [dPRVdc, 0, 4]]
#End Data Line formats

fPRVa = re.compile("""
^\s* #Footer Line 1
# Leading whitespace
(?P<msgs>\d{3})\s msgs\s # Number of Messages
(?P<dB>\d{3})>-120dB\s* # msgs > -120 dB
Best:\s* (?P<best>\d{3})\s* # Best signal in dB
Freq:\s* (?P<freq>\d{6}.\d{1})\s* # Frequency in Hz
IQ\s :\s (?P<iqx>.) (?P<iqy>\d{1}) # IQ x,y
""", re.VERBOSE)

fPRVb = re.compile("""
^\s* #Footer Line 2
# Leading whitespace
Lat1:\s* (?P<lat1>\d{1,2}.\d{3}[NS])\s* # Lat1 in dec deg N/S
Lon1:\s* (?P<lon1>\d{1,3}.\d{3}[EW])\s* # Lon1 in dec deg E/W
Lat2:\s* (?P<lat2>\d{1,2}.\d{3}[NS])\s* # Lat2 in dec deg N/S
Lon2:\s* (?P<lon2>\d{1,3}.\d{3}[EW]) # Lon2 in dec deg E/W
""", re.VERBOSE)

#-----
#-----
# DIAG Expressions

prog = re.compile("""
^\s* #First Line of DIAG
# Leading Whitespace
Prog\s* (?P<program>\d{1,5}) # Program Number
""", re.VERBOSE)

hDIAG = re.compile("""
^\s* #Diag Header
# Leading whitespace
(?P<ptt>\d{5})\s{2} # PTT
Date\s :\s # "Date :"
```

```

(?P<date>(?(P<day>\d{2}).(?(P<mon>\d{2}).(?(P<yr>\d{2})))\s # Date
(?P<time>(?(P<hr>\d{2}).(?(P<min>\d{2}).(?(P<sec>\d{2})))\s* # Time
LC\s : \s (?(P<lc>.)\s* # Location Class
IQ\s : \s (?(P<iq>\d{2}) # IQ
"", re.VERBOSE)

IDIAG1 = re.compile(""" #Stats Line 1
^ \s* # Leading whitespace
Lat1\s : \s* (?(P<lat1>\d{2}.\d{3}[NS])\s* # Lat1 in dec deg N/S
Lon1\s : \s* (?(P<lon1>\d{2,3}.\d{3}[EW])\s* # Lon1 in dec deg E/W
Lat2\s : \s* (?(P<lat2>\d{2}.\d{3}[NS])\s* # Lat2 in dec deg N/S
Lon2\s : \s* (?(P<lon2>\d{2,3}.\d{3}[EW]) # Lon2 in dec deg E/W
"", re.VERBOSE)

IDIAG1a = re.compile(""" #Stats Line 1 as '?'s
^ \s* # Leading whitespace
Lat1\s : \s* (?(P<lat1>.....)\s* # Lat1 unknown
Lon1\s : \s* (?(P<lon1>.....)\s* # Lon1 unknown
Lat2\s : \s* (?(P<lat2>.....)\s* # Lat2 unknown
Lon2\s : \s* (?(P<lon2>.....) # Lon2 unknown
"", re.VERBOSE)

IDIAG2 = re.compile(""" #Stats Line 2
^ \s* # Leading whitespace
Nb\s mes\s : \s (?(P<nbmsg>\d{3})\s* # Num messages
Nb\s mes>-120dB\s : \s (?(P<dB>\d{3})\s* # Num > -120dB
Best\s level\s : \s (?(P<best>-\d{3})\s dB # Best level in dB
"", re.VERBOSE)

IDIAG3 = re.compile(""" #Stats Line 3
^ \s* # Leading whitespace
Pass\s duration\s : \s (?(P<passdur>\d{3})\s\s* # Pass duration in s
NOPC\s : \s (?(P<nopc>\d) # NOPC
"", re.VERBOSE)

IDIAG3a = re.compile(""" #Stats Line 3 with '?'s
^ \s* # Leading whitespace
Pass\s duration\s : \s (?(P<passdur>...)\s\s* # Pass duration unknown
NOPC\s : \s (?(P<nopc>\d?) # NOPC unknown
"", re.VERBOSE)

IDIAG4 = re.compile(""" #Stats Line 4
^ \s* # Leading whitespace
Calcul\s freq\s : \s # "Calcul freq :"
(?(P<fq1>\d{3})\s (?(P<fq2>\d{6}.\d)\s Hz\s* # Frequency in Hz
Altitude\s : \s* (?(P<altitude>\d{1,4})\s m # Altitude in m
"", re.VERBOSE)

dDIAGh = re.compile(""" #Hex data
^ \s* # Leading whitespace
(?(P<data1>[A-F0-9]+\s+ # Field 1
(?(P<data2>[A-F0-9]+\s+ # Field 2
(?(P<data3>[A-F0-9]+\s+ # Field 3
(?(P<data4>[A-F0-9]+\s+ # Field 4

```

```

""" , re.VERBOSE)

dDIAGd = re.compile("""          #Two data fields
    ^\s*                        # Leading whitespace
    (?P<data1>[A-F0-9]+\s+      # Field 1
    (?P<data2>[A-F0-9]+\s+      # Field 2
    """, re.VERBOSE)

ARGOSPASS.PY
import datetime
class PASS():
    """PASS Object"""
    idcounter = 0

    def julian(self):
        """Returns a julian timestamp based on basetime for mvalue."""
        #Basetime is 1/1/1900 12:00 am
        basetime = datetime.datetime(1900,1,1,0,0,0)
        diff = self.timevalue - basetime
        return diff.seconds / 86400.0 + diff.days
    mvalue = property(julian)

    def gettimestamp(self):
        """Returns date string instead of datetime object for timevalue"""
        return self.timevalue.ctime()
    timestamp = property(gettimestamp)

    def insertSQL(self):
        sql = [self.satellite, self.lc, self.iqa, self.iqb, self.lat1, self.lon1,
            self.lat2, self.lon2, self.nbmes, self.nb120db, self.bestlevel,
            self.passdur, self.nopc, self.freq]
        return sql
    #This will return a list to construct a VALUES statement for an INSERT into ArgosInfo

    def __str__(self):
        return "PASS object %s at %s on tag %s." % (self.passid, self.timestamp,self.PTT)

    def __init__(self, prog, device, timev, sat, locclass, iq, lt1, ln1, lt2, ln2, nbm,
        num120db, best, duration, no_pc, frequency):
        self.__class__.idcounter +=1
        self.passid = self.__class__.idcounter
        self.program = prog          #string
        self.PTT = device            #string
        self.timevalue = timev       #datetime
        self.satellite = self.satdomain(sat) #str convert to domain
        self.lc = self.locdomain(locclass) #str convert to domain
        self.iqa = int(iq[0])        #integer
        self.iqb = int(iq[1])        #integer
        self.lat1 = self.llconv(lt1) #float or blank
        self.lon1 = self.llconv(ln1) #float or blank
        self.lat2 = self.llconv(lt2) #float or blank
        self.lon2 = self.llconv(ln2) #float or blank
        self.nbmes = int(nbm)        #integer
        self.nb120db = int(num120db) #integer

```

```

self.bestlevel = int(best)    #integer
try:
    self.passdur = int(duration)
except ValueError:
    self.passdur = -1         #integer, or -1 if N/A
try:
    self.nopc = int(no_pc)
except ValueError:
    self.nopc = -1           #integer, or -1 if N/A
self.freq = float(frequency) #float

@classmethod
def initlist(cls,a):
    """Receives a list of 16 elements to generate a PASS object instead of using the default
    constructor"""

    if len(a) == 16:
        b = PASS(a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],
                a[13],a[14],a[15])
        return b
    else:
        print "List passed to initlist must have 16 elements."
        return None

@classmethod
def satdomain(cls, sat):
    return sat

@classmethod
def locdomain(cls, lc):
    return lc

@classmethod
def llconv(cls, coord):
    try:
        numcoord = float(coord[:-1])
    except ValueError:
        return -1
    except:
        raise
    dircoord = coord[-1]
    if dircoord in ('S','W','s','w'):
        numcoord = numcoord * -1.0
    elif dircoord in ('N','E','n','e'):
        pass
    else:
        print "Cannot convert coordinate:", coord
        return
    if dircoord in ('E','e','W','w'):
        if abs(numcoord) > 360.0:
            print "Coordinate value out of bounds:", coord
            return
        else:
            if numcoord > 180.0:
                numcoord = (numcoord - 360.0)
            elif numcoord <= -180.0:

```



```

        numcoord = (numcoord + 360.0)
    if dircoord in ('N','n','S','s'):
        if abs(numcoord) > 90.0:
            print "Coordinate value out of bounds:", coord
            return
        numcoord = round(numcoord, 5)
    return numcoord
#-----

ARGOSDATA.PY
import datetime
class DATA():
    """DATA Object"""
    idcounter = 0

    #Returns date string instead of datetime object for timevalue
    def gettimestamp(self):
        return self.timevalue.ctime()
    timestamp = property(gettimestamp)

    def __str__(self):
        return "DATA object %s at %s for PASS %s." % (self.dataid,self.timestamp,self.passid)

    def __init__(self, tv = datetime.datetime.today(), dup = 1, PID = -1, rawdata = "test"):
        self.__class__.idcounter +=1
        self.dataid = self.__class__.idcounter
        self.timevalue=tv          #datetime
        self.duplicates=int(dup)   #Integer: Duplicate messages
        self.passid=PID            #ID for associated PASS object
        self.raw=str(rawdata)      #Stored raw string

TESTENV.PY
import argos, dbtesting
import os, csv, sys
da = argos.DownloadARGOS()
pathstra = 'E:\\070716dg.txt'
pathstrb = 'E:\\070716.txt'
directory = 'E:\\'
files = [pathstra,pathstrb]
print da
print 'pathstra',files[0]
print 'pathstrb',files[1]
for entry in files:
    parseout = da.Parse(da.CleanFile(entry))
    if len(parseout) == 2:
        dg,dggarb = parseout
    elif len(parseout) == 3:
        ds,dt,dsgarb = parseout
    else:
        print "%s lists returned by Parse(). Expected 2 (DIAG) or 3 (PRV)." % len(parseout)
print "Output to text archives complete."
print "Dumping garbage and creating database loader."
try:

```

```

    da.WriteCSV(dggarb,'%s\\garbage.csv' % directory)
    da.WriteCSV(dsgarb,'%s\\garbage.csv' % directory)
except:
    print "Unable to write to garbage files"
    raise
paired,unmatchdg,unmatchds = da.PairedParse(dg,ds,dt)
try:
    da.WriteCSV(unmatchdg,'%s\\unmatcheddg.csv' % directory)
    da.WriteCSV(unmatchds,'%s\\unmatchedds.csv' % directory)
except:
    print "Unable to output unmatched headers."
    raise
finaloutput = da.GenerateObjects(paired)
# This section is for testing database inserting of the pass objects
for entry in finaloutput:
    print entry[0]
    dbtesting.acc.findpass(entry[0])
# This section below was to output a listing of every object created
## for line in entry[1]:
##     print '    %s: %s' % (line,line.raw)

```

AUTOMATION UTILITIES

AUTORUN.PY

```

import time, os, sys, string
def main(cmd, inc=60):
    try:
        while 1:
            os.system(cmd)
            time.sleep(inc)
    except KeyboardInterrupt:
        print "Keyboard interrupt detected in autorun.main()."
        pass
    except:
        raise

if __name__ == '__main__':
    if len(sys.argv) < 2 or len(sys.argv) > 3:
        print "usage: %s command [seconds_delay]" % sys.argv[0]
        sys.exit(1)

    cmd = sys.argv[1]
    try:
        if len(sys.argv) < 3:
            main(cmd)
        else:
            inc = int(sys.argv[2])
            main(cmd,inc)
    except KeyboardInterrupt:
        print "Keyboard interrupt detected in autorun.py."
        pass
    except:
        raise

```

```

AUTOARGOS.PY
from os import system
import sys
import autorun
def main(username,password,directory,program = None):
    try:
        if program:
            cmd = 'argos.py -p %s %s %s %s' % (program,username,password,directory)
        else:
            cmd = 'argos.py %s %s %s' % (username,password,directory)
        print "Interrupt with Control-C"
        autorun.main(cmd, 86400)
    except KeyboardInterrupt:
        print "Keyboard interrupt detected in autoargos.py."
        print "Exiting..."
        sys.exit()

if __name__ == '__main__':
    if len(sys.argv) < 4 or len(sys.argv) > 5:
        print "usage: autoargos.py username password directory [program]"
        sys.exit(1)
    username = sys.argv[1]
    password = sys.argv[2]
    directory = sys.argv[3]
    if len(sys.argv) < 5:
        main(username, password, directory)
    else:
        program = sys.argv[4]
        main(username, password, directory, program)

```

```

ADDSTARTUP.PY
from _winreg import *
import os,sys

aReg = ConnectRegistry(None, HKEY_LOCAL_MACHINE)

try:
    targ = r'SOFTWARE\Microsoft\Windows\CurrentVersion\Run'

    print "*** Writing to", targ, "***"
    aKey = OpenKey(aReg, targ, 0, KEY_WRITE)
    aPath = os.path.abspath(os.path.dirname(sys.argv[0]))
    aTool = 'autoargos.py'
    aToolpath = r'%s\%s' % (aPath,aTool)
    try:
        SetValueEx(aKey, "Autoargos",0,REG_SZ,aToolpath)
        print "Autoargos will now execute on startup."
    except EnvironmentError:
        print "Encountered problems writing into the Registry..."
        raise
finally:
    CloseKey(aKey)

```

```

print "*** Reading from", targ, "***"
aKey = OpenKey(aReg, targ)
try:
    for i in range(1024):
        try:
            n,v,t = EnumValue(aKey,i)
            print i,n,v,t
        except EnvironmentError:
            print "You have", i, "tasks starting at logon"
            break
    finally:
        CloseKey(aKey)

finally:
    CloseKey(aReg)

DELSTARTUP.PY
from _winreg import *
aReg = ConnectRegistry(None, HKEY_LOCAL_MACHINE)
try:
    targ = r'SOFTWARE\Microsoft\Windows\CurrentVersion\Run'
    aKey = OpenKey(aReg,targ,0,KEY_WRITE)
    try:
        DeleteValue(aKey, "Autoargos")
        print "Autoargos will no longer execute on startup."
    finally:
        CloseKey(aKey)
finally:
    CloseKey(aReg)

```