

# Table of contents

- [Background](#)
- [Technology Development Overview](#)
  - [The Supervisor's Overview](#)
  - [Security](#)
  - [Software Overview](#)
- [Data Types](#)
  - [In-situ Observations](#)
    - [Format discussion](#)
    - [Data locations](#)
    - [Quick links](#)
  - [Model Output](#)
    - [Format discussion](#)
    - [Data locations](#)
    - [Quick links](#)
  - [Remote Sensing](#)
    - [Quick links](#)
  - [Equipment and Variable Inventory](#)
- [Aggregation Process](#)
  - [In-situ Observations and Model Output](#)
  - [Remote Sensing](#)
- [Database Specifications](#)
  - [PostgreSQL database with PostGIS extension](#)
  - [Data structures / canonical forms](#)
    - [point form](#)
  - [Visualization fields](#)
  - [Remote Sensing Particulars](#)
  - [Software Installation and configuration](#)
- [Application Performance](#)
- [Visualization Process](#)
  - [Web mapping with MapServer](#)
    - [Mapserv CGI](#)
    - [PHP/MapScript](#)
    - [Mapfile Details](#)
      - [Mapfile Example](#)
    - [Software Installation and configuration](#)
  - [OGC web services](#)
    - [WMS and WFS instances](#)
  - [Support Visualization Applications](#)
    - [GMT \(Generic Mapping Tools\)](#)
  - [Data exploration applications](#)
  - [Appendix I - Installation Tips](#)
  - [Appendix II - Troubleshooting \(Gotcha's and Gremlins\)](#)
    - [PostgreSQL](#)

- [Postgres operations \(update, fetch, etc\) seem to hang](#)
- [PostGIS](#)
  - [WFS queries against a table have disappeared](#)
  - [mapfile FILTER query doesn't return map - get sql error](#)
  - [can't get COPY to run](#)

## Background

The Southeast Atlantic Coastal Ocean Observing System (SEACOOS) is a distributed real-time ocean observation and modeling program that is being developed for a four-state region of the SE US, encompassing the coastal ocean from the eastern Gulf of Mexico to beyond Cape Hatteras.

SEACOOS was presented with the chance to define data standards and integrate in-situ observations, model output, and remote sensing data products from several institutions and programs. The development of a near-realtime data stream and subsequent GIS visualization provides immediate feedback and validation as to the effectiveness of this regional observation effort. Distribution of these aggregated datasets raises further questions about data QA/QC procedures and leveraging SEACOOS technology solutions to larger spatial scales.

## Technology Development Overview

This document covers the major steps SEACOOS followed to create a near real-time data stream of in-situ observations, model output, and remotely sensed imagery. Our hope is that an understanding of this process will help other ocean observing efforts as they formulate their initial technology strategies. This process is outlined in a linear fashion below, recognizing however, that iteration between several steps at once is likely, and that you should keep it simple. Perfect visualization of one layer before you try ten.

1. Conceptualize available data types and consider possible storage schemas: SEACOOS collects data on ocean state variables and biogeochemistry in the form of in-situ observations, circulation models, and satellite remote sensing. The spatial and temporal frequency of these data is highly variable and required considerable forethought to address all possible data configurations.
2. Develop and standardize data ontologies, file formats, and transport protocols: Developing a dictionary of common language to refer to our disparate data was a significant achievement. This was crucial in the development of SEACOOS data-specific file formats using netCDF. Although DODS/!OPeNDAP servers are available at partner institutions, currently the raw netCDF files are uploaded directly, bypassing these servers.
3. Determine desired applications and requisite software packages: SEACOOS visualizes data spatially and graphically, providing researchers and an external audience with access

to this information in near real-time. Open source GIS (MapServer) and graphics packages (GnuPlot) are used to drive these applications.

4. Determine database schemas for observations, model output, satellite imagery, and equipment metadata: With both the data and application ends of the data stream conceptualized, a database schema to enable their connection was developed. The open source PostgreSQL database, with PostGIS extension for mapping, is used by SEACOOS.

5. Address hardware ramifications of implementing the database and applications under consideration: [We have not talked about hardware anywhere in this doc so far, so someone make sure I've stated this correctly?] SEACOOS utilizes separate servers to house the database, web mapping application, and project website.

6. Implement schemas and applications: Intermediary code development is crucial in automating and connecting these disparate technologies to handle a near real-time data stream. SEACOOS uses perl as the primary scripting language to obtain, parse, and store incoming data in the PostgreSQL database. PHP/!MapScript is used to create interactive mapping applications, embedding MapServer controls within HTML pages.

7. Leverage solutions “outward” to external audiences: As part of the IOOS push, SEACOOS cascades much of its data into this larger data aggregation effort. The OGC services are utilized to transfer map images and raw data to external GIS applications.

## **The Supervisor's Overview**

If your organization is going to implement the same or a similar visualization methodology, somewhere there is a supervisor who is accountable for allocating the resources (money, people, and equipment) to this effort. They may or may not have an IT background to fully understand the myriad nuances inherent to a project like this. They do, however, have a better understanding of all the other projects that are underway or anticipated, and how those projects might be affected by this specific effort. They need a realistic picture of what's involved, and it is in everyone's best interests to attempt a realistic assessment before work begins.

The following is a very simplified 30,000 foot view of the visualization implementation, broken down to help the techs help the administrators get a handle on what is involved. The idea is to move through the list, determine what steps need to happen first, and assess each step along the following guidelines:

What do we already have?

What do we need to acquire (and what will it cost)?

In what order do the steps needs to proceed (e.g., does it make sense to buy the software before buying the hardware?)

How long should each step take (in a perfect, works-the-way-it's-supposed-to world)?

How long until we are up and running? i.e, Other people are watching and waiting - how do I determine a reasonable delivery date that won't set us up to fail?

Most of these answers will depend entirely on the organization, and the organization's existing infrastructure, experience, and skillsets. As any experienced technician knows, it is impossible to foresee every complication, and things rarely work exactly as promised in the manual. Still, one can make some basic estimates. For example, it should take no more than an hour to download and install the needed Perl libraries. This assumes, however, that you did your homework and know the versions and compatibility of your software, hardware, etc., and that you know what you're doing. If you really don't know what you're doing, there's no way we can write everything into this cookbook to save you.

In this list, we have given the software versions that we have used because we know the combination(s) work. This does not mean other versions will not work, you'll just have to do the background work or experiment.

-Step-----Time-----  
Have/Need?--

<b>Review needed skills</b>	15 mins
* UNIX command line	
* knowledge of RDBMS	
* Perl/PHP	
* system administration	
* web server admin	
* image manipulation	

<b>Review software needed</b>	30 mins
* Red Hat Enterprise Linux (v 7.3)	
* Apache web server (v 1.3)	
* PostgreSQL (v 7.4.1)	
- PostGIS extension (v 0.8.1)	
* Perl (v 5.8.0)	
- netcdf-perl module (v 1.2.2)	
- udunits library (v 1.12.1)	
* ImageMagick (v 5.5.7)	
* GnuPlot (v )	
* GMT (v 3.4.3)	
* PHP (v 4.3.2)	
* MapServer (v 4.0.1)	
- Proj4 (v 4.4.7)	
- Gdal (v 1.2.4)	
- GD (v 2.0.28)	
* MapLab (v )	
* LibWWW (v 5.3.2)	
* GIFsicle (v 1.40)	
* Anis (v )	

**Review hardware to be used**

1 day

This is very specific to your project, and naturally depends on the breadth and depth of your data and audience.

SEACOOS is now running 4 machines and total hardware cost was \*roughly\* \$20,000. We have approximately 1000 in situ obs sites (including feds) plus model output and remote sensing. We have two database servers, and one application server, plus one desktop PC for the data aggregation scout. We started out with just one box, but it quickly became overloaded. You may start out small and find out you need to grow. Be sure to think about your archive target - how much data are you keeping and for how long.

**Define variables and metadata to be collected** 4-16 weeks

Assess who are your data providers, what are they collecting, how often are they reporting, etc. The time also depends on whether or not you can/want to use the existing SEACOOS NetCDF.

**Purchase, set up, and install software and hardware** 4-8 weeks

Build database according to equipment, variable, and model info to be stored

**Using PostgreSQL**

3 days

**Aggregate data**

1 day

This will involve adjusting the data scout and database schemas to match your datasets

**Establish and test data to visualization connection**

**Command line testing to determine map file is working correctly** 1-5 days

**Configure map file**  
on complexity of visualization

1-10 days depending

(try it out on MapLab first)

**Configure PHP files to talk to map files**

4-10 weeks

**enable OGC WMS functionality**

2-6 weeks

## Security

We're relying on our respective university systems for security. If you have sensitive data, you'll need to take measures that are not covered anywhere in this cookbook.

# Software Overview

Presented below is the software mix currently in production within SEACOOS. Open source software is used whenever possible in an effort to enable similar instantiation of this mix by other ocean observing systems. Many of the package versions in use by SEACOOS can be downloaded from [here](#) or from each package's website, linked below. More specific installation tips for the application packages are available in Appendix I.

## General packages:

- [Apache web server](#)
- [PostgreSQL database](#)
  - [PostGIS extension](#)
- [Perl](#)
  - [netcdf-perl module](#)
  - [udunits library](#)

## Application packages:

- [PHP4 - web based templates](#)
- [MapServer - web mapping application](#)
  - [Proj4 - cartographic projection library](#)
  - [Gdal - geospatial data abstraction library](#)
  - [GD - graphic library](#)
  - [LibWWW - w3c protocol library](#)
- [MapLab - MapServer application builder](#)

## Visualization tools (time series, maps, compositing)

- Graphs - time series, 3D, etc
  - [GnuPlot](#)
- Maps (color ramps, contours, etc)
  - - [GMT - Generic Mapping Tools](#)
- Animations
  - - [GIFsicle - produces animated gifs](#)
    - [Anis - Java-based applet for animating images](#)
- Compositing (image overlays)
  - - [ImageMagick](#)

# Data Types

SEACOOS partner institutions each create and collect a variety of ocean observations across the region. These observations are made under a variety of data transport and storage schemas. An initial challenge was how to maintain this diversity while also encouraging aggregation of this disparate data for the entire SEACOOS domain. This began with a look at the data types collected by partner institutions and adapting transport formats to these types.

One key consideration was to develop solutions for the most complex data model and let everything else fall out as a subset of that case. With this in mind, SEACOOS chose to model the data by making all variables (including latitude, longitude and depth) a function of time. Other data forms allowed for programmatic 'shortcuts' based on the types of dimensions presented in the file - for instance, if latitude and longitude were each a dimension of 1 point, then the file was processed as a fixed station. Most of the debate centered around whether description should be carried in the variable attributes or in the dimension or variable naming convention. COARDS, CF conventions were adopted where possible. SEACOOS uses the netCDF format to transport in-situ observations and model output.

It's important to note that, out of the box, MapServer only provides visualization for x and y (lat and longitude). One of the real strengths of our SEACOOS visualization is the inclusion of time and depth. Unfortunately, this also makes things more complicated, and this is the reason for all of our supporting elements.

## **In-situ Observations**

### **Format discussion**

After much discussion between SEACOOS institutions, data standard proposals for a SEACOOS in-situ netCDF format were developed. The resultant SEACOOS netCDF convention serves as a specific syntax of variable dimensions and attributes. The format currently focuses on a few different cases: fixed-point, fixed-profiler, fixed-map, moving-point-2D, moving-point-3D and moving-profiler. This format can be programmatically parsed by USC's "data scout" ([perl code here](#)) which downloads the netCDF files via HTTP from data providers and populates the aggregate database or alerts the provider when there is a problem with the data being input.

- All the [canonical forms](#) which were under consideration.
- [Documentation](#) on the current SEACOOS CDL v2.0 format SEACOOSv2.0.

### **Data locations**

Examples of SEACOOS in-situ observations in netCDF format can be seen at the following **data provider directories**:

- [http://nemo.isis.unc.edu/data/nos/proc\\_data/latest\\_v2.0/](http://nemo.isis.unc.edu/data/nos/proc_data/latest_v2.0/)
- [http://seacoos.skio.peachnet.edu/proc\\_data/latest\\_v2.0/](http://seacoos.skio.peachnet.edu/proc_data/latest_v2.0/)

- [http://nemo.isis.unc.edu/data/nws\\_metar/proc\\_data/latest\\_v2.0/](http://nemo.isis.unc.edu/data/nws_metar/proc_data/latest_v2.0/)
- [http://trident.baruch.sc.edu/usgs\\_data/](http://trident.baruch.sc.edu/usgs_data/)
- [http://trident.baruch.sc.edu/storm\\_surge\\_data/latest/](http://trident.baruch.sc.edu/storm_surge_data/latest/)
- [http://seacoos.marine.usf.edu/data/seacoos\\_rt\\_v2/](http://seacoos.marine.usf.edu/data/seacoos_rt_v2/)
- <http://oceanlab.rsmas.miami.edu/ELWX5/v2/>
- [http://trident.baruch.sc.edu/po\\_daac\\_data/OVW/](http://trident.baruch.sc.edu/po_daac_data/OVW/)
- [http://nemo.isis.unc.edu/data/nc-coos/latest\\_v2.0/](http://nemo.isis.unc.edu/data/nc-coos/latest_v2.0/)
- <http://oceanlab.rsmas.miami.edu/wera/>

## Quick links

- MapServer-based [Observations Interactive Map](#)
- DODS/OPeNDAP [data access](#)
- [CSV \(Comma Separated Value/Excel\) files](#) and perl generation programs

## Model Output

### Format discussion

SEACOOS modeling groups had a different set of integration issues to resolve. Model resolution and interpolation issues regarding time and space were discussed as they related to model output, region overlaps, and their display. Display of the results via the GIS helped articulate these various projection, alignment, and resolution issues. Since all the model data was homogenous area/field oriented data, deciding on a common netCDF representation was fairly straightforward.

### Data locations

- [http://nemo.isis.unc.edu/data/nc-coos/model\\_data/quoddy/forecast/](http://nemo.isis.unc.edu/data/nc-coos/model_data/quoddy/forecast/)
- [http://nemo.isis.unc.edu/data/nc-coos/model\\_data/met/awip12/](http://nemo.isis.unc.edu/data/nc-coos/model_data/met/awip12/)
- [http://nemo.isis.unc.edu/data/nc-coos/model\\_data/adcirc/](http://nemo.isis.unc.edu/data/nc-coos/model_data/adcirc/)
- <http://seacoos.marine.usf.edu/data/wfsmodel/>
- <http://efsis.rsmas.miami.edu/netcdf/>

## Quick links

- MapServer-based [Interactive Model Display](#)
- [Documentation](#) on the current SEACOOS CDL v2.0 format SEACOOSv2.0.

## Remote Sensing

To complement the in-situ observations being collected in this domain, SEACOOS has engaged the real-time satellite remote sensing capabilities at the University of South Florida and the University of Miami, including redundant ground station facilities. The integration of remotely sensed data into the SEACOOS program has provided a regional

context for in-situ time series data, as well as a historical baseline for the region's surface waters. SEACOOS partners are engaged in the collection of real-time satellite data (some tailored for the SEACOOS domain), the production of derived data products, and the rapid delivery of these products via the SEACOOS web portal. Formatting decisions are left to the data providers and image transport is handled by FTP of images as \*.png files. Currently SEACOOS is ingesting images from the MODIS, AVHRR, and QuikSCAT platforms.

## Quick links

- MapServer-based [Remote Sensing Interactive Map](#)
- Remote sensing [listserv](#)

## Equipment and Variable Inventory

This inventory is a database and queriable map housing detailed information about observation platforms, sensor equipment, and environmental variables measured across the SEACOOS region. It presents the spatial resolution of sensors and variable measurements while also serving to facilitate technical discussion amongst the SEACOOS observation community. Visit the [SEACOOS Equipment and Variable Inventory](#)

An additional goal is to link this information as metadata to specific observations and thus assess the quality of measurements made by specific equipment over time. This equipment assessment requires creating a database with an eye towards historical data, since QA/QC procedures may later determine a problem with particular data and thus an individual sensor. For example, that all of the measurements made by the thermosalinograph Z on platform X between March and April were off by three degrees. If you are storing observations, you need to store the equipment metadata for those observations as long as the observation data themselves.

Our basic inventory schema is as follows: [need to create a graphic to go in here]

**Institutions** manage **Platforms** (towers, buoys, etc. - right now this is restricted to in situ sensors)

- **Platforms** house **Equipment** (sensors and support equipment)
  - **Equipment** (sensors) measure **variable(s)**

**Institutions** also have **Contacts**

- **Contacts** are associated with a piece of **Equipment**

## Aggregation Process

Initializing the SEACOOS data stream required the development of data transport formats (netCDF, \*.png) as well as destination database schemas. This aggregation process is much more complex for in-situ and model output data than for remotely sensed data.

## In-situ Observations and Model Output

The process SEACOOS followed to prepare for in-situ and model output data streams formatted to netCDF files is as follows:

1. Database schema preparation: Pick a physical variable of interest (like wind speed & direction, sea surface temperature). Each variable is defined within a separate database table (one record for each measurement). One table would contain station id, time, latitude, longitude, depth, **wind\_speed**, **wind\_direction**, and associated metadata fields. Another table would contain station id, time, latitude, longitude, depth, **sea\_surface\_temperature**, and associated metadata fields. Table joins are possible using SQL, but are not currently used. Instead each separate table generates a GIS layer which can then be superimposed.
2. Determine how the measurements will be defined in time and space: SEACOOS uses the same base time frame of seconds elapsed since 1970-01-01 (this can be a floating point number for subsecond intervals). For spatial considerations SEACOOS has developed a framework for datatypes relating to the degrees of locational freedom of the measurement point(s). This provides guidelines on how these should be defined in a netCDF file via dimensions and attributes.
3. Additional considerations for display purposes: Metadata fields are added which take into consideration:
  - whether the data point should be shown in the display
  - whether the data point can be normalized given an agreed upon normalization formula
  - how the data point is orientated as it relates to a specific locational framework
  - how the data are interpolated and chosen for display

To add new physical in-situ variables, aside from addressing any new naming conventions, step 3 is the only step which should be required. Steps 1 & 2 are initial group discussion/decision processes which are subject to periodic consideration and revision if needed. Step 3 take product (GIS in this case) considerations into mind, whereas the work accomplished in steps 1 & 2 should be universally applicable for aggregation needs across a variety of products.

For in-situ observations and model data, each partner institution set up a DODS/OPeNDAP netCDF server to share a netCDF file representing the past 2 weeks worth of data. This data is still available via this interface, but since each transmission only involves a few kilobytes, a direct approach is currently used. So, for performance reasons, when aggregating the data at the central relational database, the netCDF files are uploaded directly (not utilizing the DODS/OPeNDAP API) and parsed with [perl netCDF](#)

[libraries](#). Data providers are alerted when there is a problem with the data made available to the data scout.

- USC's [perl data scout](#) which gather the latest data(...\_latest.nc) from providers on a periodic basis and converts these netCDF files to SQL INSERT statements to populate the relational database.
- [Documentation](#) on the current SEA-COOS CDL v2.0 format SEACOOSv2.0.

## Remote Sensing

The aggregation process for remotely sensed data differs from the temporally regular data mentioned above. Images are fetched periodically (how often/what prompts the fetch?) from SEACOOS partners based on available image timestamps and downloaded to USC (as \*.png files). ImageMagick is used to remove any background or watermark images. The cleaned image contains an embedded timestamp in the filename, an associated .wld file (georeferencing), and has a matching reference created in a remote sensing database lookup table. These lookup tables contain pointers to all remotely sensed images on the filesystem accessed by timestamp. The timestamp information is used to determine which image should be displayed for given temporal conditions in SEACOOS mapping applications.

MODIS RGB images are also part of a compositing process. After a new RGB image has been fetched, a script ([build\\_composite.php](#)) is called which creates a resultant image of the past n passes (in our case 4) that are no older than y days (in our case 4). New images are similarly included as part of the DB lookup process, along with non-composite images.

## Database Specifications

SEACOOS uses the open source PostgreSQL relational database to store in-situ observations, model output, and remotely sensed data aggregated project partners. PostgreSQL can be accessed by a number of front-end applications and expanded to include geospatial datatypes using the PostGIS extension.

### PostgreSQL database with PostGIS extension

SEACOOS data is stored in two PostgreSQL database instances at the University of South Carolina (USC). One instance contains the in-situ observations and remotely sensed data. The other contains model output data and duplicate in-situ observations, used for “round-robin” updating. The databases are partitioned into separate tables for each in-situ observation variable, remotely sensed raster layer, and model variable layer per hour. The remotely sensed tables do not house the actual images but pointers to the image files and their ancillary boundary files. The RS tables are used to execute raster

queries, which require the image RGB values to be referenced against a look-up table of actual measured values.

The PostgreSQL database is “spatially enabled” using the PostGIS extension for PostgreSQL. PostGIS adds several geospatial objects to the supported datatypes in PostgreSQL. This functions as the spatial database engine for all subsequent GIS data visualization. This extension converts the raw locations of SEACOOS observation data and stores them in a GIS-accessible geometry column, enabling mapping and spatial query functionality. GIS mapping applications utilize these new columns to render the geometric topologies they contain. PostGIS fields can also be populated from other common GIS data formats such as ESRI shapefiles. For details, see above.

## Data structures / canonical forms

The structure of temporal, geospatial data as it is stored in various formats should ideally be capable of having its structural elements described in a handful of forms. Describing and labeling these forms (and what should be abstracted away) are the beginning steps before automated programmatic conventions, labels, and processing can be utilized in data transformation.

As an example, two predictable forms for storing buoy data are:

- 'by station' where the tablename is that of the station and each row corresponds to all the variable reading for a given time measurement
- 'by variable' where the tablename is that of the variable measured and each row corresponds to a measurement time, station id, and possibly lat, long, and depth describing the measurement point and the measurand value. Example below...

Currently the GIS favors a 'by variable' approach which corresponds to variable data layers. This format is concise, amenable to query, and resultset packaging (the ability to mix and match variables which have a similar reference scheme on each variable table). Issues of varying temporal sampling resolutions across multiple stations are also better handled in this form. SEACOOS is developing programs to convert other table formats to this format. See [here](#).

Click [here](#) for database descriptions of the wind and sst tables which USC currently utilizes. A full listing of the archival seacoos observation database schema is listed [here](#). We make efforts to keep from 'normalizing' the table into subtables preferring a single table approach with redundancy in certain fields. Since the storage needs are initially low, the database remains conceptually and operationally simple. Table performance can be further optimized by partitioning and use of VACUUM, COPY, CLUSTER commands and other indexing schemes applied similarly across these repeated table structures.

### point form

*(By variable form, used with point and moving point data)*

The following represents a basic table layout which might be implemented on a PostgreSQL database. Click [here](#) for generic table creation details.

```
CREATE TABLE <my_table> (  
  row_id SERIAL PRIMARY KEY,  
  row_entry_date TIMESTAMP with time zone,  
  row_update_date TIMESTAMP with time zone,  
  platform_id INT NOT NULL,  
  sensor_id INT,  
  measurement_date TIMESTAMP with time zone,  
  measurement_value_<my_var> FLOAT,  
  -- other associated measurement_value_<my_var> added here as well  
  latitude FLOAT,  
  longitude FLOAT,  
  z FLOAT,  
  z_desc VARCHAR(20),  
  qc_level INT,  
  qc_flag VARCHAR(32)  
);
```

## Multi\_Obs Form

Lately(January 2006), we've been rethinking some of the table structure issues. While on the one hand the one table per observation approach has been working fine, my temptation is to want to collapse these singular similar structured observation tables into one mega-table with an extra index of observation type. The advantage to this approach is hopefully easier code and database maintenance as there are less individual table references involved, but the disadvantage is also that a singular table reference can get into an all or nothing scenario when it comes to performance or problems at the database table level. See more notes at [MultiObsSchema](#) showing a sample schema and implementation.

## Xenia package

We've been trying to reduce and simplify what we're doing in terms of data collection and sharing for ocean observations down to a single set of relational database tables and scripts. See [XeniaPackage](#) .

## Visualization fields

The database described above contains additional fields to aid visualization efforts. Fields are included for:

- GIS display purposes (should this be displayed in the GIS or not) Certain in-situ data requires MapServer-specific fields to be populated in order to create maps. In-situ data must be collected into an hourly snapshot table. See function `set_wind_prod_show()` as an example of how in-situ winds are selected to appear in an hourly snapshot table. The flow of information is that raw in-situ data is

inserted into a production table, e.g. wind\_prod. set\_wind\_prod\_show() is run after a complete raw ingestion is finished. The resulting records that are flagged as being eligible to be included in the hourly snapshot table, are collected and inserted into wind\_map.

- Unit conversion (for example, celsius to fahrenheit conversions)
- Frame of reference (for example, does a positive z value correspond to up or down)
- Contact or standards metadata (for example, provider URL, standard XML fields for WMS, WFS or other standards metadata)

## Remote Sensing Particulars

Remotely sensed images are stored in directories on the filesystem based on their layer name. ( e.g. avhrr\_sst, modis\_chl, modis\_rgb\_composite, etc.) The files under these directories have embedded timestamps as part of their filenames as well as a corresponding georeferencing file (\*.wld) that MapServer requires. (e.g. avhrr\_sst\_2004\_04\_01\_03\_06.png, avhrr\_sst\_2004\_04\_01\_03\_06.wld). Distinct tables contain pointers to all remotely sensed images on the filesystem accessed by timestamp. E.g. The [raster\\_oi\\_sst](#) table DDL whose contents may be:

pass_timestamp	local_filename
2004-08-01 17:30:00	oi_sst/oi_sst_2004_08_01_17_30.png
2004-08-02 17:30:00	oi_sst/oi_sst_2004_08_02_17_30.png
2004-08-03 17:30:00	oi_sst/oi_sst_2004_08_03_17_30.png

The timestamp information is used to determine which image should be displayed from input temporal parameters in SEACOOS mapping applications. Database support is also used for raster query functions. Each raster image is converted to a table lookup of RGB values corresponding to actual measured data values. This functionality should be temporary as MapServer should better support raster query in the future.

## Software Installation and configuration

- Installation Process - precursor configurations/libraries ready? See [seacoos\\_install.txt](#).
- Servers and connections
  - [PostgreSQL database](#)
  - [PostGIS extension](#)
- PostgreSQL database schema [\*NEED HELP : show schema\*]

## Application Performance

Note: The following equipment is dated to around 2003. We are still tending towards using commodity equipment from Dell using Red Hat Linux or perhaps Fedora. We don't

have ongoing staff with specialized skills to support the latest software/hardware optimized setups.

By and large the application response for this system is good. We are able to leverage off of

- Dell PowerEdge2650 dual Xeon servers each with 2 gigabytes of RAM (2 of these, switching roles between query database and loading database). Further info on the PowerEdge purchase [here](#)
- a spatially indexed relational database ( PostgreSQL + PostGIS )
- a relatively simple and straightforward table design which either supports in situ measurements or raster images(see below topic about 'Data Structures' on this).

and all software components are open source to boot.

New data is collected periodically from all of the remote distributed sites, but queries against the GIS application are run against the already collected data on the central server database.

Our overall architecture incorporates 3 separate servers - a webserver which alternately points to one of two database servers. The two database servers switch roles between 1)collecting and populating the database(write operations) and 2)servicing queries(read operations). This allows us to keep a steady query response time not impacted by data upload.

## Visualization Process

After SEACOOS data are collected and aggregated, visualization procedures are implemented to represent this data for constituent user groups. These procedures provide immediate feedback and validation of SEACOOS data aggregation efforts, quickly addressing integration issues about data projection and resolution. These procedures use open source software whenever possible. The methods presented below encompass a significant amount of development work that has coalesced into a robust data visualization effort. This effort is an initial step toward leveraging SEACOOS project data into national and international ocean observing efforts.

## Web mapping with MapServer

Visualization of SEACOOS data over the web utilizes an open source mapping platform known as MapServer. MapServer is well adapted for use with PostgreSQL (via PostGIS) and can serve open web mapping services within a flexible, scriptable environment. Although MapServer can parse ESRI data formats, these are not required and data source customization is encouraged. MapServer utilizes a “mapfile” (\*.map) to setup parameters and control visualization details for each map instance. Basic elements of this mapfile may then be called and manipulated by HTTP requests (URL query strings), giving

MapServer its web mapping flexibility. The instance powering SEACOOS maps is housed at USC, local to the SEACOOS aggregate database.

## **Mapserv CGI**

MapServer is used in two complimentary configurations to map SEACOOS data. For more basic applications, the Mapserv CGI is used. It handles user input and directs image creation or query requests. The CGI program accepts input via either HTTP "GET" or "POST" methods and can be used in an interactive manner or as an image engine. This can give control over basic elements of the mapfile via a URL query string: toggling map layers on/off, adjusting the map size, and adjusting the map extents. More detailed changes require editing the base mapfile served from USC. For customized web applications, GUIs can be created that generate these methods which pass parameters to the MapServ CGI program. Images that are returned can be displayed in most web browsers. The mapserv CGI also powers SEACOOS OGC web services (WMS and WFS) capable of serving images and feature data into desktop GIS platforms.

## **PHP/MapScript**

A second configuration of MapServer utilizes an additional module called PHP/MapScript to enable scripting access to the MapServer C API using PHP. SEACOOS uses the PHP variant of MapScript because of PHP's templating ability and ease at parsing query string variable pairs. In addition a number of opensource toolsets have been developed utilizing the PHP/Mapscript module. This additional tier of web architecture enables much more flexible control over the base mapfile, enabling the construction of more complex graphical web interfaces. In addition to the abovementioned CGI functionality, PHP/Mapscript can control the size and location of the scalebar, legend placement, and unit conversion for example. These parameters are initially set in the mapfile and then made available, via this module, for adjustment on the fly. This functionality works behind the [Interactive Observations Map](#), the [Interactive Model Display](#), the [Equipment Inventory](#), and the [Remote Sensing Development Map](#), all hosted at USC and served through the SEACOOS website. Components of this functionality have been leveraged to drive some of the internal data exploration efforts mentioned below.

## **Mapfile Details**

MapScript allows for the dynamic setting of the DATA and FILTER parameters in the base .map file which MapServer will use to determine what time and space to display. Most of the time- and space-specific programming occurs in map\_session.php. Here is a snip from map\_session.php that shows [how the wind\\_obs FILTER is set](#). In addition to time and space sensitivities, map\_session.php also handles unit specifications, e.g. wind speeds in knots, miles per hour, and meters per seconds. MapScript adjusts the DATA parameter accordingly. For a given request of a remotely sensed layer at a given time, MapScript code references the control .map file and sets the DATA clause to define the base raster image to fulfill the users space and time query

## Mapfile Example

The code snippet from this [mapfile](#) which supports the OGC WMS/WFS services listed below shows an example of the mapfile query against the underlying database.

```
LAYER
  NAME "wind_obs_hourly_recent"
  STATUS OFF
  DATA "the_geom from (select station_id, time_stamp, z, label_z,
wind_speed, wind_speed_knots,
      normalized_wind_speed, normalized_wind_speed_knots,
wind_from_direction, label_theta,
      %wind_label_char_column% as this_label_char, label_char,
normalized_label_char, lon, lat, title,
      institution, institution_url, institution_dods_url, source,
refs, contact, report_time_stamp, the_geom,
      wind_from_direction_compass, wind_speed_mph,
normalized_wind_speed_mph, label_char_knots,
      label_char_mph, normalized_label_char_knots,
normalized_label_char_mph, value, value_knots,
      value_mph, normalized_value, normalized_value_knots,
normalized_value_mph, seq, can_be_normalized
      from wind_map) as foo USING UNIQUE seq USING SRID=-1"
  FILTER "report_time_stamp = date_trunc('hour',timestamp without
time zone '%formatted_time_stamp%') and
      station_id like '%%station_id%%'"
  DUMP TRUE
  TEMPLATE "dummy"
  TOLERANCE 5
  TOLERANCEUNITS pixels
  TYPE POINT
  CONNECTIONTYPE POSTGIS
  CONNECTION "user=***** dbname=sea_coos_obs
host=neptune.baruch.sc.edu"
  METADATA
    "wms_srs" "EPSG:4269 EPSG:4326"
    "wms_extent" "-180 -90 180 90"
    "wms_title" "In-situ sea wind speed and direction (hourly)"
    "wms_abstract" "**** OGC NOTE :: Do not access this layer
directly. This layer is a subset of a hidden
      layer called 'wind_obs_hourly'. Make requests to
'wind_obs_hourly' instead (allow the server to decide
      whether the data will be pulled from recent obs or archived
obs). Optional parameter is station_id which
      will return records whose station_id contains the parameter
as a substring. Optional parameter is
      wind_speed_units that can be one of mps, mph, knots (mps is
default). *** Using previously existing
      observing systems operating at the sub-regional level,
SEACOOS works towards increasing the quantity and
      quality of environmental information from the coastal ocean
of the Southeast and to facilitate the
      application of that data in a variety of forums and discourse
communities. These observing systems collect
```

data across a four state region (North Carolina, South Carolina, Georgia, and Florida) using a variety of sensors, which is integrated into the SEACOOS real-time data stream. The observing systems currently cooperating with SEACOOS are: SeaKeepers - an international nonprofit organization that installs sensors on platforms ranging from lighthouses to ocean-going vessels. Explorer of the Seas - The University of Miami and Royal Caribbean Cruise Lines outfitted an 1000 ft. cruise vessel with observation equipment. CaroCOOPS - an array off of the coast of the Carolinas operated by USC, N.C. State, and UNC-Wilmington. Coastal Ocean Monitoring and Prediction System - the University of South Florida operates this system observing the West Florida Shelf. South Atlantic Bight Synoptic Offshore Observational Network - is currently operated by Skidaway Institute of Oceanography in Savannah, Georgia in collaboration with the US Navy, UNC-Chapel Hill and SC Department of Natural Resources."

```

"wms_keywordlist" "wind speed, wind direction, real time"
"wfs_srs" "EPSG:4269 EPSG:4326"
"wfs_extent" "-180 -90 180 90"
"wfs_title" "In-situ sea wind speed and direction (hourly)"
END
LABELITEM "this_label_char"
LABELANGLEITEM "label_theta"
CLASS
  EXPRESSION ([wind_speed] * 1.945 >= 3)
  LABEL
    FORCE TRUE
    TYPE TRUETYPE
    FONT weather
    ANTIALIAS TRUE
    COLOR 0 0 0
    POSITION CC
    OFFSET 2 -9
    BUFFER 0
    PARTIALS TRUE
    SIZE 25
  END
  STYLE END
END
CLASS
  EXPRESSION ([wind_speed] * 1.942 < 3)
  SIZE 5
  SYMBOL 'circle'
  OUTLINECOLOR 0 0 0
  COLOR 255 255 255
END
END

```

When doing a WFS request, a selection on the geometry column from a table will actually return all of the row fields(xml tagged) for the selected rows along with the associated geometry returned in GML tags - what we do to better support this functionality is keep subtables which are populated with just the latest data formatted in

the row layout we want to return to a http query and run WFS queries against the associated subtables.

In addition to `map_session.php`, `seacoos_bathy_contents.php` and `seacoos_bathy.phtml` are top-level control files that contain references to underlying code that defines the functionality of the interactive maps.

## Software Installation and configuration

- Installation Process – precursor configurations/libraries ready? See Appendix I below.

## OGC web services

External presentation of SEACOOS data is enabled through web service standards set by the Open Geospatial Consortium (OGC). OGC web mapping services are XML based and are intended to be platform independent, web-enabled, representations of GIS data. The services can be accessed, controlled, and presented by web browsers as well as other GIS software platforms (i.e. ESRI Interoperability toolbar, GAIA application). SEACOOS OGC compliant services rely on the `mapserv` CGI engine. SEACOOS provides a Web Mapping Service (WMS) and Web Feature Service (WFS). The WMS feed returns a static, georeferenced image to a user's browser or GIS platform, while the WFS feed returns actual feature data, allowing visualization control and spatial analysis on these data externally.

Other ocean observing research projects currently ingest SEACOOS OGC feeds, the [OpenIOOS](#) project and the [GoMOOS](#) project. Several other regional GIS projects also ingest and display SEACOOS data served via WMS: [NCOneMap](#). This method of integration is a critical step in the desired aggregation of ocean observing data across regional, national, and global scales.

## WMS and WFS instances

SEACOOS OGC services rely on a wrapper script around a normal [mapfile](#) that contains additional OGC-specific metadata. Customized Perl wrappers were written to interpret input time parameters. Here is an [example of how SEACOOS remotely sensed WMS requests are handled](#).

- Descriptions of how to access these OGC-ready layers can be found here: [WMS and WFS in-situ layers](#) and [WMS remotely sensed layers](#).
- [Open Geospatial Consortium](#)

## Support Visualization Applications

Several other open source applications are used to graph, animate, and query SEACOOS data. GIFsicle and AnimationS (AniS) are used to create and control data animations over the web. ImageMagick (and PerlMagick) is used for image manipulation and to execute raster data queries. Mouseover query functionality is enabled with the searchmap component of MapServer, which creates an imagemap of the existing map image for queries. Gnuplot is used to generate time series graphs. All of these tools are scriptable and run behind the SEACOOS interactive maps.

- Implementation details
  - ImageMagick details for image manipulation and raster queries. Here is [an example of extracting a sea surface temperature from an image](#).
  - Animation Routine – user details. Here is [an example of how to produce an animated gif of map images](#).

## GMT (Generic Mapping Tools)

The code below lists examples of how GMT is used to create contour or color ramped images for subsequent image query by mapserver. This is a preferred method where there might be an unnecessarily large number of mapfile layer classes needed to simulate a color ramp or where a mapping technique does not exist as with contours.

trailed from machine trident, user scmodel crontab:

nautilus:/usr2/home/scmodel/scripts/[mk\\_gmt\\_grids.pl](#)

runs an automated regular(hourly) psql query which returns a text file (lon, lat, value)

```
$psql_command = $psql_prefix
.' -c "select lon, lat, prmsl'.$units.' from
met_sea_level_pressure_unc_stage'
.' where prmsl'.$units.' > -1 and time_stamp = timestamp without
time zone'
.'" '$time_stamp';\''";
$cmd = $psql_command
.' | '$psql_suffix
.' >
.'"$txt_dest_dir/met_sea_level_pressure"'.$units.'"_unc_prod_$formatted_t
ime_stamp.txt";
```

This text value is passed to GMT 'surface' function which will take a text file and return a grid (.grd) file

```
$surface_cmd = 'surface'
.'"
$txt_dest_dir/met_sea_level_pressure"'.$units.'"_unc_prod_$formatted_time
_stamp.txt"
.'" -R$grd_surface_bounds"
.'" -I$grd_surface_sample_incr"
```

```
." -G  
$txt_dest_dir/met_sea_level_pressure".$units."_unc_prod_$formatted_time  
_stamp.grd";
```

see GMT surface command <http://gmt.soest.hawaii.edu/gmt/doc/html/surface.html>  
and GMT color palette tables (.cpt files)  
[http://gmt.soest.hawaii.edu/gmt/doc/html/GMT\\_Docs/node51.html](http://gmt.soest.hawaii.edu/gmt/doc/html/GMT_Docs/node51.html)

these grid files are later used by  
nautilus:usr2/maps/seacoos/ui/portal\_model/wrapper/[map\\_session.php](#) which pulls just  
the grid area of interest using the appropriate contour scale via  
nautilus:/usr2/maps/seacoos/util/[mk\\_contour.php](#)

## Data exploration applications

Further data exploration and visualization has been enabled to allow researchers quick access to the SEACOOS database. These tools are automated web pages that rely on PHP to interact with the PostgreSQL database and MapServer, presenting database content ranges and simple maps. The following pages are in use by SEACOOS researchers and are updated in near real time:

A [data overview page](#) displays a list of min and max timestamps for all SEACOOS interactive map data (model input data and observations data). It also provides links to individual pages for each data layer displaying the specific time slices available for each individual layer. Access is also available to map images of each layer and each time and date stamp, across 3 selectable regions. While these pages are not intended for the general public, they provide on-demand access and visualization to the entire SEACOOS database for our distributed research community.

The [data animation page](#) takes URL query string parameters and creates animations of data ingested by SEACOOS. The animation routine combines maps and graphs for most SEACOOS data. Users have control over the GIS layers, scale, platforms to graph, and time step. These animations are then served via another PHP generated page with full animation movement controls. These animations are created, stored, and served at USC until the user asks for them to be removed. Example [here](#).

The flexibility of PHP/MapScript allows other SEACOOS institutions to utilize the USC database for further visualization development. A [cached observation page](#) serves static images each hour for a variety of SEACOOS data layers and sub regions. This page is supplied by a script that sends modified URL query strings to the USC MapServer and caches the map images that are returned.

## Appendix I - Installation Tips

General

- Apache (latest version)
- postgresql 7.4.1
- PostGIS 0.8.1 (follow installation instructions [here](#))
- Perl 5.8.0 (get latest version from perl.com)
  - netcdf-perl-1.2.2 (for data scout)
  - udunits-1.12.1 (for data scout)
- ImageMagick 5.5.7 (build the PHP .so)

```
./configure --with-perl=/usr/local/perl/bin/perl
```

#### Application server

- PHP 4.3.2 (both the cgi and cli binaries)

```
./configure \
--enable-shared \
--with-regex=system \
--with-gd \
--with-ttf=/usr \
--enable-gd-native-ttf \
--with-jpeg-dir=/usr \
--with-png-dir=/usr \
--with-zlib \
--enable-force-cgi-redirect \
--enable-dbase \
--with-config-file-path=/etc/httpd/conf \
--with-freetype-dir=/usr/include/freetype \
--with-pgsql \
--with-dbase1
```

- MapServer-related installs
  - proj-4.4.7
  - gd-2.0.28
  - gdal-1.2.4
  - libwww-5.3.2-20011211
- MapServer 4.0.1 (latest version is 4.4.x, but let's stick w/ 4.0.1 for now)

```
./configure --with-proj \
--with-wmsclient \
--with-libwww \
--with-gdal \
--enable-runpath \
--with-php=../php-4.3.2 \
--without-tiff \
--with-gd=/usr/local \
```

```
--enable-force-freetype1 \  
--enable-internal-ld-detect \  
--with-postgis \  
--with-wfs \  
--with-ogr
```

- GMT 3.4.3

## Appendix II - Troubleshooting (Gotcha's and Gremlins)

A section for notes on common bugs or problems and their fixes, especially those that are difficult to diagnose and address.

### PostgreSQL

[Postgresql documentation](#)

Indexes <http://www.postgresql.org/docs/8.1/static/indexes.html>

[Performance tips](#)

[Maintenance tips](#)

### Additional Links

<http://www.varlena.com/varlena/GeneralBits/Tidbits/perf.html>

[http://www.varlena.com/varlena/GeneralBits/Tidbits/annotated\\_conf\\_e.html](http://www.varlena.com/varlena/GeneralBits/Tidbits/annotated_conf_e.html)

Good Overview <http://www.budget-ha.com/postgres>

[http://database.sarang.net/database/postgres/optimizing\\_postgresql.html](http://database.sarang.net/database/postgres/optimizing_postgresql.html)

Kernel shared memory

[http://database.sarang.net/database/postgres/optimizing\\_postgresql.html](http://database.sarang.net/database/postgres/optimizing_postgresql.html)

When trying to optimize against operating system parameters, it's best to try changing one thing gradually at a time on a non-critical system to get an idea of how this effects things(run some experiments where you can afford to learn from mistakes). An easy mistake to make is to set some resource parameter(like shared memory) too high or low and a database or other external process becomes hung as a result.

### PostgreSQL vs MySQL

I've talked to several people who ask what PostgreSQL offers that MySQL doesn't. My list on this is the following:

- spatial indexes - compliance and awareness of OGC specs ( <http://opengeospatial.org> ) for geometric datatypes and support of bounding box and other functionality via the PostGIS modules ( <http://www.postgis.org> ).
- PostGIS also supports PostgreSQL to ESRI shapefile, ESRI shapefile to PostgreSQL conversion utilities.
- more closely follows conventional SQL standards and database functionality from proprietary enterprise databases like Oracle and SQL Server.

A few other links echoing similarly from a Slashdot September 2005

<http://ask.slashdot.org/comments.pl?sid=161173&cid=13484662>

<http://ask.slashdot.org/comments.pl?sid=161173&threshold=1&commentsort=0&mode=thead&cid=13487926>

March 2006

<http://developers.slashdot.org/article.pl?sid=06/03/14/2112210>

April 2006 - Postgresql vs. Oracle

<http://developers.slashdot.org/developers/06/04/06/1828206.shtml>

### **PostgreSQL GUI tool**

Here's the Postgres GUI interface tool we use if needed. I don't use the GUI tools that much since I can get most of what I need done via command line(live by the GUI die by the GUI), but it is helpful for getting a quick view of table structures or data(shared development environment) or learning the functionality of the database(basically the command line arguments are wrapped in the GUI).

PostgreSQL Manager(free 30 day evaluation download, single user license approximately \$200) quoting from their website (<http://ems-hitech.com/pgmanager/>)

*EMS PostgreSQL Manager is a powerful graphical tool for PostgreSQL? administration and development. It makes creating and editing PostgreSQL? database objects easy and fast, and allows you to run SQL scripts, manage users and their privileges, build SQL queries visually, extract, print and search metadata, export data to 14 available formats and import them from most popular formats, view and edit BLOB fields, and many more...*

30 day trial version <http://nautilus.baruch.sc.edu/resources/misc/pgmanager.zip>

### **Postgres operations (update, fetch, etc) seem to hang**

With the database running fine for several months, one day postgres processes (ps -eaf | grep postgres) seem to be piling up and preventing other processes from completing. Turns out that this database instance was not being vacuumed at all. After running

vacuum/analyze on the database with the below command, processes stopped stacking up and things ran smoothly again.

```
/usr/local/pgsql/bin/vacuumdb -U postgres -d sea_coos_obs -h neptune.baruch.sc.edu -f -v --analyze
```

If you're running the 'cluster' command tables contained within the above database, you'll want to run the cluster command before the vacuum/analyze command (see <http://archives.postgresql.org/pgsql-admin/2004-05/msg00217.php>)

## PostGIS

[PostGIS documentation](#)

[WKT examples](#)

### WFS queries against a table have disappeared

After a table which has been spatially enabled using PostGIS indexes has been running for several months, the WFS queries against the table stop returning data. The problem here is that if you are using the default postgresql 'oid' sequence for your sequence reference, this is good till about 2 million references and then ceases to be a usable index(see <http://mapserver.gis.umn.edu/data2/wilma/mapserver-users/0405/msg00013.html>). The solution is to always define your own unique sequence id per table and make sure the maximum value is well high enough or it's ok to cycle the sequence.

Adding a sequence column to an existing table 'my\_table'

```
create sequence my_table_seq increment 1 minvalue 1 maxvalue 9223372036854775807 start 1 cache 1;
```

```
alter table my_table add column seq integer not null default nextval('public.latest_obs_by_station_id_seq'::text);
```

Reference within mapfile

```
LAYER
  NAME "my_layer"
  STATUS OFF
  DATA "the_geom from my_table as foo USING UNIQUE seq USING SRID=-1"
  TYPE POINT
```

**mapfile FILTER query doesn't return map - get sql error**

If the FILTER condition fields are **outside (like a joined table)** the same table reference as 'the\_geom' or geometry column the query is selected from, then **make sure the field references are included in the DATA section** of the mapfile and the DATA section is also a valid SQL selection.

### can't get COPY to run

Very hard to find this documented anywhere, but **naming the columns you are copying to\*(as in the example below) seems to be more effective than leaving them unnamed.** Also note that the **\*file system reference is local to the postgresql database instance system which the copy statement is being run against** and **not** the local filesystem where the copy statement is run from. So if the below statement is run from machine A against a postgresql database on machine B, the file system reference is for machine B..

```
#file md_0.cpy
```

```
1,089,1,0,0.000656,POINT(-79.988844 32.700249)
1,089,1,0,0.000984,POINT(-79.988011 32.700249)
1,089,1,0,0.00164,POINT(-79.987179 32.700249)
```

```
#copy statement
```

```
copy surge_products
(catalog_id,track_id,product_id,time_frame,measurement_value,the_geom)
from '/home/jsmith/cpy_files/md_0.cpy' using delimiters ',';
```

### creating shapefiles

The following script loops through a control file to determine the scenario track\_id, time step (step\_id) and uses this data to produce a database query against a PostGIS table (table 'hurricane') which returns a shapefile using PostGIS pgsq2shp function.

pgsql2shp documentation

<http://postgis.refrains.net/docs/ch04.html#id2854856>

```
#!/perl
```

```
#the following file just provides a control file which changes the
query and output filename according to the track_id, step_id
open(RANK_FILE, "surge_list_seq.csv");
my $line;
```

```
while ($line = <RANK_FILE>) {

    #ignore lines which start with hash
    if ($line =~ /^#/) {
        next;
    }
}
```

```
    }

    #process line

    my ($track_id, $step_id) = split(/,/ , $line);

print "processing sql track_id: $track_id ";
print `date`;

$filename = 'scenario_'. $track_id;

#get rid of previous run
`rm track*.dbf; rm track*.shp; rm track*.shx`;

#generate shapefiles
`/usr/local/pgsql/bin/pgsql2shp -u postgres -h neptune.baruch.sc.edu -f
$filename hurricane "select measurement_value,the_geom from
surge_products where catalog_id = 1 and track_id = $track_id and
product_id = 1 and time_frame = $step_id"`;

#generate zip
`find . -name "track*" -print | zip shapefile_$filename -@`;
}

close(RANK_FILE);

exit 0;
```