

Implementing an Ad-Hoc Mapping Service With ArcIMS, PHP, and XML

Jeffrey C. Burka

This paper presents the design and use of ArcIMS with PHP and XML to create an ad-hoc dynamic mapping web service in support of TranStats, a one-stop intermodal transportation data warehouse and analysis web site created by the U.S. Department of Transportation's Bureau of Transportation Statistics. The mapping service enables the creation of dynamic (features like pan, zoom, identify, etc.) state and county choropleth maps for any dataset by allowing the client to send the thematic and join data encoded in an XML request which is then processed in a server-side application framework using ArcIMS as the mapping engine.

Introduction

An overly simplified description of ESRI's ArcIMS is that it is an application running on a web server which can accept XML-encoded requests for maps and return XML-encoded pointers to the images of the maps described by the request. The U.S. Department of Transportation's Bureau of Transportation Statistics (BTS) has developed a server application framework that accepts XML-encoded requests for maps and then turns around and uses ArcIMS to create maps which are then used within HTML-based mapping applications sent to the client. This use of multiple XML requests may seem a bit redundant, but this paper intends to show that this methodology is a fast, robust, and flexible way of enhancing ArcIMS while simplifying for third parties the task of providing mapping applications to users.

Background

BTS was created by the 1991 [Intermodal Surface Transportation Efficiency Act \(ISTEA\)](#) with the mission of performing data collection, analysis, and reporting. In support of that mission, BTS has a GIS staff which handles cartographic assignments for the agency, as well as collection and dissemination of geographic data and statistical data with geospatial attribution.

BTS has been interested in a web-based mapping presence for more than three years. Although the initial stages of exploration into web mapping were made with ArcviewIMS, when ArcIMS 3.0 was released in 2000, development moved to that new platform. During the summer of 2000, the different development options available in ArcIMS 3.0 were examined within the constraints of our Solaris-based servers. The two Java-based viewers shipped with the product were ruled out because they would not run in Netscape 4, and support of that browser was a requirement. The Cold Fusion connector was tested, but eventually discarded, due to performance concerns, in favor of the ESRI HTML Viewer. The other connector available at the time, the ActiveX connector, was not an option due to the use of Unix servers.

During the summer of 2000, BTS was working on a web-based data distribution and analysis project referred to as the Intermodal Transportation Database. The mapping applications were viewed as an extension of the ITDB, and would provide two distinct types of functionality. The first set of functionality was for a class of applications intended for specific datasets, applications which would provide users with data-specific methods to query the data, create maps, and interact with those maps. The second use envisioned was to be able to map datasets selected in an ad hoc manner from the warehouse. If a dataset featured a geospatial attribute, such as a state FIPS code, there ought to be a way to have our ArcIMS viewer application display that dataset. To fulfill these requests, several sample applications were built using the HTML Viewer as a base.

The modifications made to the HTML Viewer during this time ranged from simple to complex. On the simple end were changes to various colors and graphics to fit the ITDB look. An online help system was added. But the bulk of the work was focused on creating a method of launching the Viewer in such a way that it not only displayed the layers described in a map service's configuration file, but also including layers whose data were selected, or even generated, by queries made by a user. More radical surgery was required to enable the Viewer to allow the insertion of these new, dynamically defined layers. The technique devised involved replacing the files viewer.htm and MapFrame.htm with perl scripts, mapstart.pl and mapframe.pl respectively, containing most of the contents of the original HTML files, along with references to the dynamically-generated files necessary to get the site running with the custom layers.

ITDB, at the time, was being written in [PHP](#), the PHP Hypertext Preprocessor, and the developer working on that project suggested that the programmer doing this initial ArcIMS work might find PHP useful for building such things as query forms. As a result, the earliest applications were written in a combination of PHP and perl. [List 1](#) contains the various steps and calls back to the server that the original methodology required. [Figure 1](#) provides a flow chart of the process of launch a map with a custom layer added.

1. When the user submits a query form, it is passed to a PHP script.
 1. The PHP script parses the form data and uses it to query an Oracle database and create a temporary table.
 2. The PHP script next creates a pseudo-XML file (no DTD/Schema definition, no expectation that a real XML parser will need to use this file) which lists parameters for the new map and a reference to the new temp table.
 3. Next, the PHP script passes that pseudo-XML to a perl script, mapinterface.pl, on the server. mapinterface.pl parses the tags in the pseudo-XML and uses the values retrieved to create a temporary JavaScript file which will contain the necessary functions and variables needed by the HTML Viewer to create and work with the custom layers, as well as other BTS-designed functions. The perl script returns to the PHP script the name of this new JavaScript file.
 4. Finally, the PHP script passes back to the browser client a web page containing a reference the temporary JavaScript file. The new web page, upon loading, will make a call back to the web server with a URL for a perl script called mapstart.pl, and passes to mapstart.pl the filename of the JavaScript file.
2. mapstart.pl, which takes the place of ESRI's viewer.htm file, sets some JavaScript variables and then creates the frameset for the application. In ESRI's original viewer.htm, one of the subframes of the frameset uses as its source MapFrame.htm. Instead, mapstart.php sets that frame's source to be mapframe.pl, to which it again passes the JavaScript file's name.
3. As the browser renders the frameset, it loads mapframe.pl as the contents of the frame to contain the map image and JavaScript code for interacting with the map.
mapstart.pl simply outputs what would have been output by MapFrame.htm, but includes the contents of the JavaScript file created by mapinterface.pl.

List 1: Convoluted Process for Launching Application

Although complex (perhaps overly so!), the method used presented an opening that would allow maps of any data to be created, as long as the data were in the Oracle database and could be joined to the spatial data layers in SDE. Instead of relying on web-based forms to create queries which were then used to generate temporary lookup tables, any other software could cut into the process at step [1-c](#) by providing its own pseudo-XML to the perl script creating the temporary JavaScript file. As 2000 progressed, the perl code was replaced with PHP code performing the same functions. The architecture of the applications did not change, just the implementation

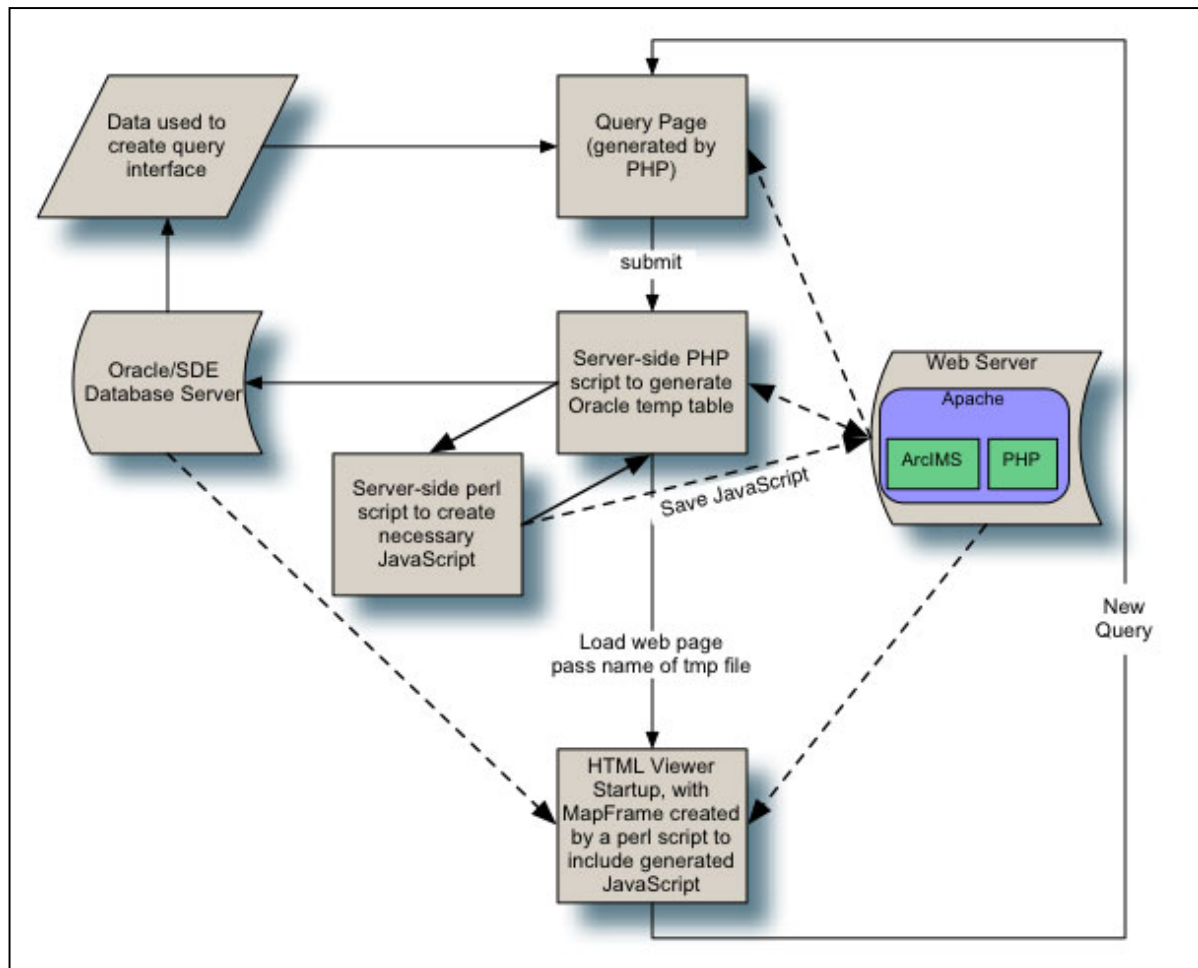


Figure 1: Original Architecture

As the development staff moved into 2001 and started planning for an official site launch, the number of applications expanded, and their abilities grew with additional tools written in PHP. Some of these tools included a new ID window display, high resolution PDF output of map layouts, and a tool for modifying symbology on maps. Perhaps the most significant of these, with respect to what was to come, was a replacement for the ArcIMS legend generation capability. The legend tool was requested to address some bugs in ArcIMS 3.0's legend generation, including the improper resizing of symbols with the "patch" attribute and the overlapping of large raster symbols in a ValueMapRenderer's legend. Along with the PDF layouts, the legend generator was the first step into using PHP for more than database access and forms processing. In particular, the work on the legend marked an early attempt at parsing ArcXML and importance of this would significantly impact future work. The Intermodal Transportation Database web site went live in April of 2001, and the mapping applications followed a few weeks later in May. However, the experiments in allowing ad hoc mapping from the data warehouse, though successful, were never actually implemented as part of the live ITDB site.

PHP Connector

At the end of 2001, as the more modern TranStats web site was being implemented to replace the Intermodal Transportation Database, the GIS staff was once again asked to look into the idea of being able to map ad hoc collections of data. While there was already a working implementation using our heavily modified version of the ESRI HTML Viewer, it was complicated, non-standard, relied on applications beyond the web server, and, most important to the client, the application was large and slow. One of TranStats' main advantages was to be its speed improvements over ITDB.

The issues that lead to those complaints about the HTML Viewer can be divided into three categories: the need to store state on the client, the need to process XML on the client, and the sheer functionality of the client.

State can be described as the status of the application at any given moment. Any variables that have been set, objects which have been created or had properties set or modified – state is what allows the application to be able to react to the user's interaction with it. In web applications, state can be stored on the server or on the client. In an HTML Viewer-based application, the state contains things related to making maps, like the list of layers in a service or the current extent of a map, and the state also contains information related to user interaction, such as whether the legend or the table of contents is displayed, and what tool is active, should the user click on the map. And all of that information is stored on the client.

The fact that state is stored on the client brings us to the next issue. When state is stored on the client, any request to the server must be generated on the client because that's where all of the information used to create the request is stored. And, conversely, the client must be capable of understanding any information sent to it by the server so that it can not only update the display for the user, but also update its own state so that it will be ready to generate the next request. In a web browser, all of this functionality must be written in JavaScript (also known by the name ECMAScript, given by an international standards body when the language was [accepted as a standard](#)), the standard browser scripting language. And parsing XML is simply not one of JavaScript's strong suits. While there are now some JavaScript libraries implementing object-oriented parsers using regular expressions for relatively speedy data extraction, the code in the HTML Viewer responsible for extracting the important pieces of information from an ArcIMS AXL response, such as the URL specifying a newly created image's location, was implemented using simple loops through the AXL strings, searching for specific element names and then searching for attributes, and finally calculating the length of the attribute's value so a substring command could fetch the actual value. This method works, and the speed is acceptable when dealing with small AXL statements, but it can bog down very quickly with larger statements. To make matters worse, as the size of an AXL string grows, the memory usage and stability of the browser can diminish significantly.

Finally, with respect to the function set, it must be said that the HTML Viewer is surprisingly similar in functionality to ArcView 1.0 in what the user is able to do to explore and query data. And in order to support that level of functionality in a client-only application, a tremendous amount of JavaScript was necessary – close to 400 kilobytes, which needed to be transferred from the service to the client's browser just to start the application.

It was at this time that the GIS programming staff floated the idea of abandoning the existing architecture and application code altogether, and moving from the thick HTML Viewer client to a thin client and servlet. [Numerous improvements](#) to the application could be had by shifting to a server-side application, in which state was determined and maintained off of the client, the JavaScript code could be pruned to just what was necessary for the client to display and allow users to interact with maps – and depending on the type of application, the amount of JavaScript could be reduced to nil. BTS was still running their web servers and ArcIMS on a Sun Solaris platform. By this time, staffing had changed and the programmers on the BTS GIS staff had no Cold Fusion experience. With ArcIMS 3.1 ESRI included the AppServerLink, but not a full-blown Java connector. And with all of the work the programmers had done to enhance the HTML Viewer using PHP, the logical choice seemed to be leverage that PHP experience.

- The client application is much smaller (approximately 0% - 5% of the size of the HTML Viewer)
- The client application is more stable

- The state is stored on the server, so less data is transferred from the client to the server during an update, and vice versa
- The XML parsing takes place on the server, using optimized parsing libraries
- The web server, mapping application, and ArcIMS were all on the same server, speeding up requests and response

List 2: Expected Benefits of Server-Side Redesign

In December of 2001, work began on this new server-based application. There were a few simple ideas guiding the design and implementation of the servlet. First, it would use the Servlet Connector to communicate with ArcIMS. Second, it would be object-oriented. Third, due to time constraints, the initial version of the servlet and the client web application would be limited to the bare necessities for the proof-of-concept – an ArcIMS connector written in PHP, with a fast, lightweight client which would be capable of creating maps based on XML files passed to the application.

Looking to the ActiveX connector object model for its convenient poster, the first three weeks of December, 2001 saw the most basic implementations of [objects](#) necessary to prove the concept put together, along with a simple application using the new connector. The design and implementation of the connector was kept very simple. The Connector object is in charge of communication with the ArcIMS server, and handles this by opening a socket and posting a request directly to ArcIMS using the Servlet Connector. The Connector does not do anything with the requests it receives or the responses it returns; it merely acts as a conduit to enable applications to talk to ArcIMS. The Map object, when created, is given a pointer to a Connector object and the name of a service on the ArcIMS server to which Connector is pointing. The Map then tracks its own variables, such as the height and width of an image to draw, the extent of the map display, the layers in the map and their order and visibility.

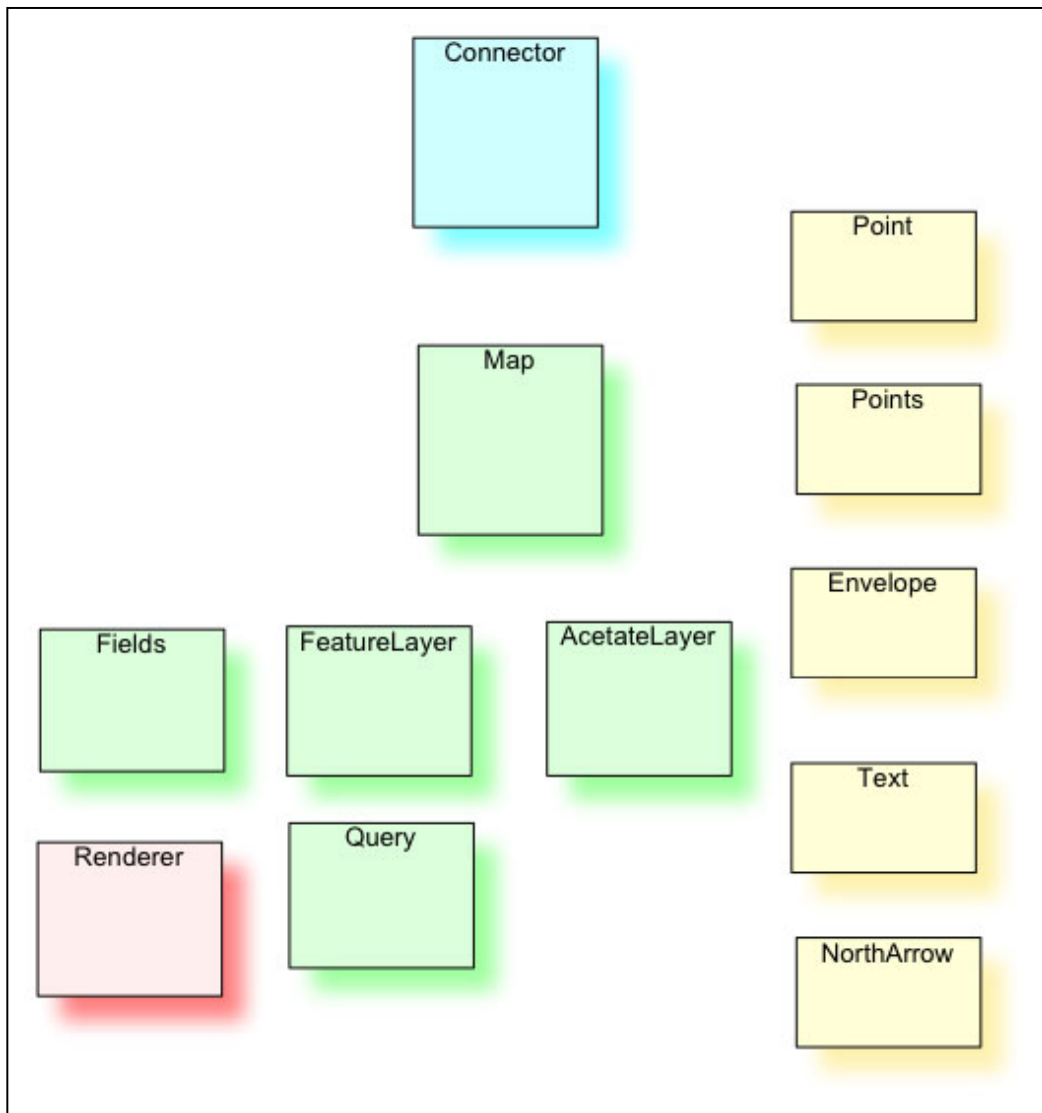


Figure 2: Objects Implemented for Proof of Concept

The application which went along with the PHP objects was a simple [frameset](#), showing the continental United States, along with insets for

Hawaii and Alaska, as well as a legend. To make the application faster, separate image map services were created for each frame, and the data displayed in those frames was projected to an appropriate coordinate system for the area to be portrayed. Navigation was performed using small buttons at the bottom of the browser window. If only a single map had been used in the application, navigation would be handled by including a navigation instruction as part of the "href" attribute of the link containing the button image (e.g. href="http://websas.bts.gov/website/transtatsService/mapViewer.php?nav=zoomout") so the PHP application could be told how to adjust the Map object to affect the change to the extent. Instead, because of the multiple frames, a small JavaScript function is called so that the PHP application can be told which of its Map objects needs to be updated, and to ensure that the resulting image is placed into the correct frame. Additionally, in the sample application, a small amount of JavaScript was also used to calculate the sizes of the frames so that the Map objects could be adjusted automatically to provide images to properly fit the window size.

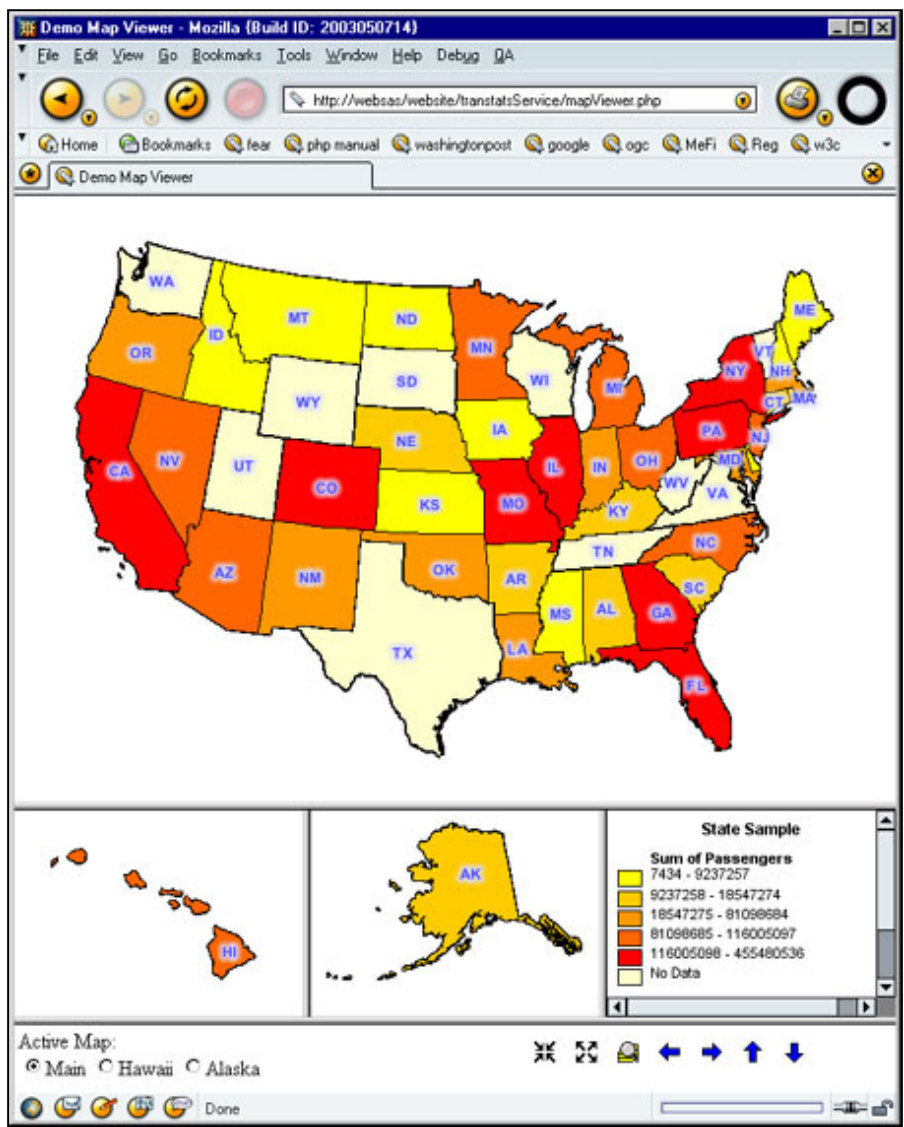


Figure 3: Sample Application with Three Maps and Legend

BTSXML

To pass requests to the BTS server software, an XML grammar was needed which would allow clients to specify the details of what was to be mapped. Not knowing who might want to make use of a service such as BTS was trying to put together, it was felt that a more formal specification should be provided, rather than the pseudo-XML used in earlier iterations of the software. Further, having a defined schema would allow validation of the XML passed to the servlet to ensure that all necessary data were included and correctly formatted and arranged. To that end, the [BTSXML DTD](#), or Document Type Definition, was created. The specification would contain elements for all of the different options one might want to specify in a BTS application, such as:

- map name
- legend title
- join table

- join field
- geographic area (e.g. state, county)
- layer info
- layer data
- breaks/colors

Not long after demonstrating a fully functional prototype application, the programming staff was given a twist to work around. It turned out that the database which would be serving as the new TranStats warehouse would be running in Sybase, and not sharing an Oracle database as had been expected. This left the programmers needing to find a way to get the attribute data to be mapped from the Sybase database into Oracle so that they could be joined to the spatial layers in SDE and mapped. Fortunately, since the external applications (e.g. TranStats) which would be making requests to the PHP servlet were already composing and sending XML strings, it was a fairly simple matter to add a data element to the BTSXML schema and modify the servlet to be able to parse out attribute data and write it to an Oracle temporary table which could then be joined to the existing SDE layers in the database. This made the composition of the XML string more difficult for the third party applications calling the BTS mapping service, but the tradeoff for that complexity would be the ability to map any data one might wish to send the server, as long as the BTS Oracle/SDE database had a matching spatial layer to join against. The flow of data in this new version can be seen in [Figure 4](#). The biggest drawback to having to process attribute data encoded in XML and then loaded into an Oracle table was not one of complexity, but rather one of speed. After experimenting with different methods of parsing and loading the data and running benchmarks against the different methods, the technique settled upon was to write the data into a text file and then use Oracle's batch loading application, sqlldr, to move the data in bulk into the database. Using this method, approximately one second was added to the time necessary to create a map from the moment the request was received by the server to create a lookup table for US states. For US counties, with over 3,000 records, the added time was approximately five seconds. While slower than would be ideal, the speed hit was deemed small enough. A [BTSXML sample](#) including attribute data to be loaded into the database can be found in Appendix B.

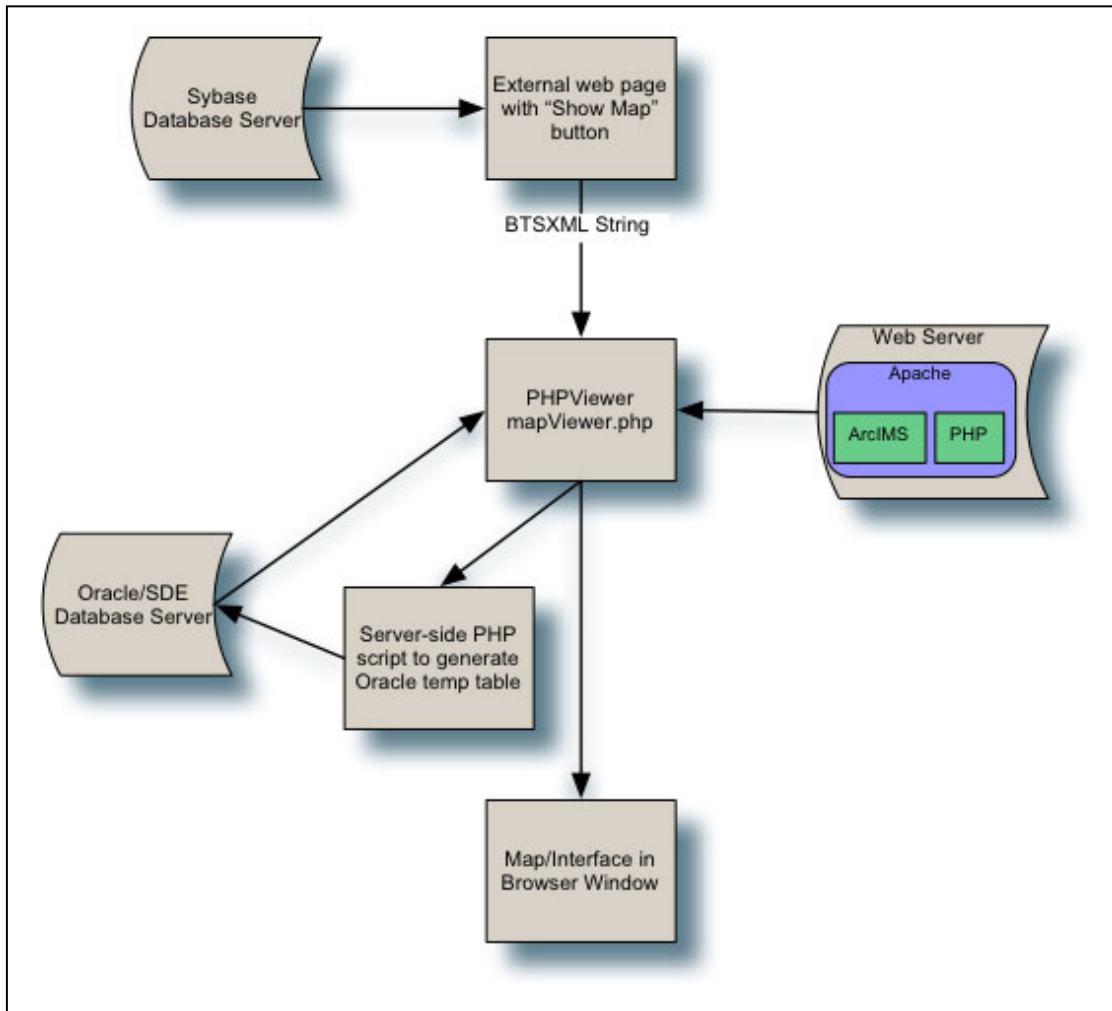


Figure 4: Updated Architecture with Passed-In Data

During the ensuing months in early 2002, the TranStats mapping project was put on hold. But having seen the performance gains possible, the programming staff was given a green light to enhance the PHP Connector and implement the objects necessary to support the functions already

present in the ESRI HTML Viewer, and to develop a PHP-based version of the Viewer with which to replace the existing applications. The resulting Connector was much more fully realized, as can be seen in the images in [Appendix C](#). The resulting PHP-based applications, which went live on the BTS web site starting in the spring of 2002, were approximately two times faster than the original versions, significantly smaller, and more stable. All subsequent BTS web mapping applications have been developed using the PHP Connector and the accompanying PHP Viewer.

In May of 2002, TranStats had reached a point in its development where it was feasible to once again look at the dynamic mapping service, the prototype of which had been demonstrated in the previous January. The BTSXML DTD was provided to the TranStats team, along with suggestions as to how they might wish to embed the mapping application in their own application. TranStats had been designed to allow users to drill down through the various databases in the warehouse, and see tables and perform simple analyses on data from those tables. [Figure 5](#) contains an image of TranStats with such a table displayed. On the top right of the table is a series of drop-down boxes which allow the user to change the data displayed in the table, by field, category, time period, or calculated value. In addition to the tabular display of the data, a list of options to the left of the table provides alternate ways to visualize the data.

The screenshot shows a web browser window titled "Analysis" with the URL http://www.transtats.bts.gov/Oneway.asp?Field_Desc=Day%20. The page header includes the TranStats logo, "Bureau of Transportation Statistics", and the date "Tuesday, June 17, 2003". Navigation links for "home", "glossary", "databases", "contact us", "about", and "help" are present. A search bar on the left contains the text "Search the Results:" and "Go". The main content area is titled "On-Time : On-Time Performance" and displays a table with the following data:

Code	Description	Summary
1	Monday	315,767
2	Tuesday	309,871
3	Wednesday	327,325
4	Thursday	316,270
5	Friday	316,982
6	Saturday	263,814
7	Sunday	294,972
All Rows	All Rows (including those not displayed)	2,145,001

Additional interface elements include filter controls for "DayOfWeek", "Flights", "Sum", and "2003", and a list of "Analysis Type" options on the left: Table, Chart, Map, Crosstabs, Time Series, and Terms & Definitions. An "Analysis Summary" section offers options like "Top N", "Bottom N", "Value", "Percent of Total", "Sort Descending", and "Sort Ascending".

Figure 5: Transtats Table Display With Filters

To leverage this existing framework, a simple "Map" option was added to the possible analysis types. When a user clicks on the Map option, the TranStats application generates a BTSXML stream and sends a request to the BTS map server. The target of this request is an IFrame. The various filters remain in place above the map, as they were above the table, and clicking the "Recalculate" button causes a new BTSXML string to be generated and the mapping application to be reloaded with its new parameters.

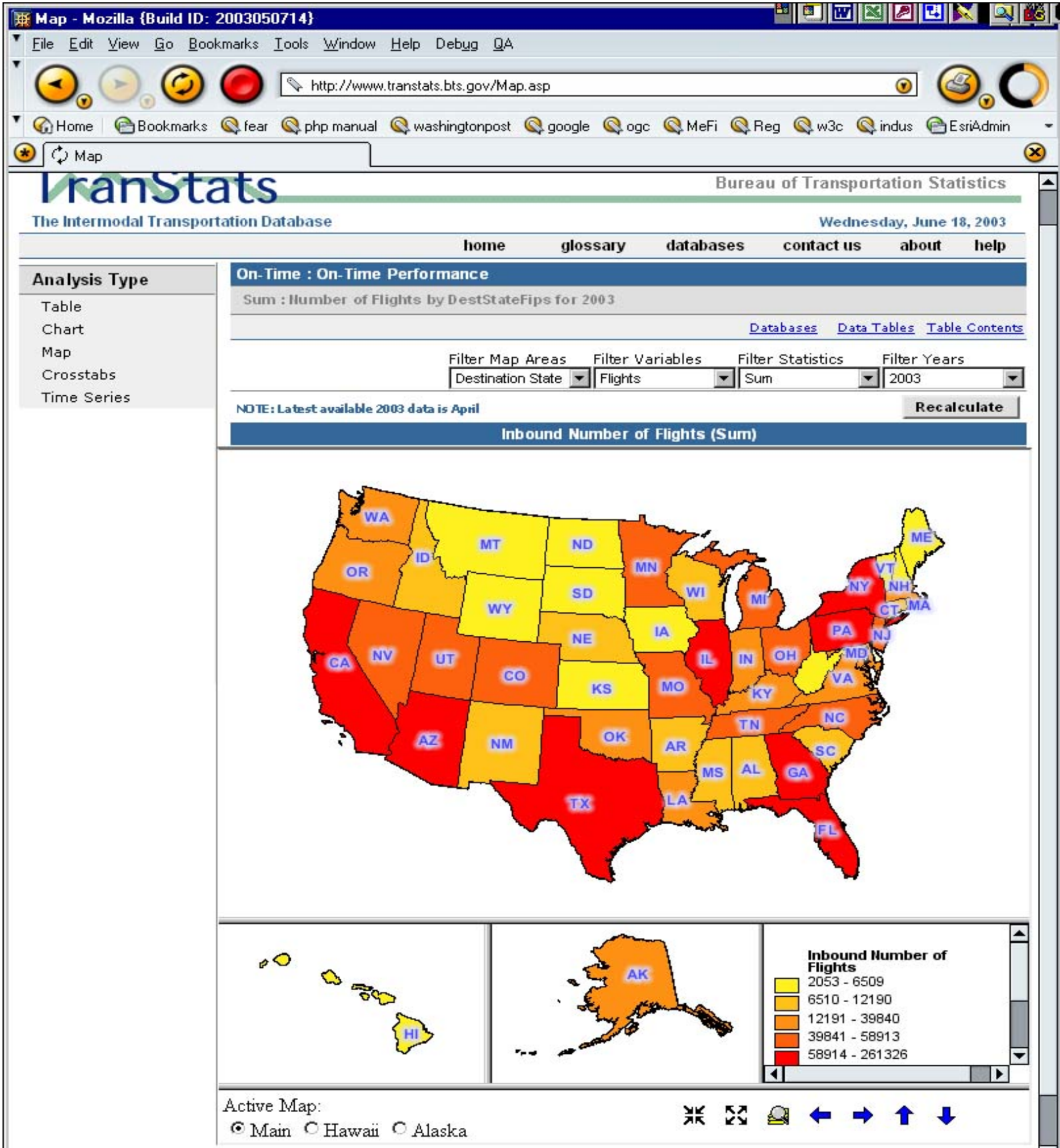




Figure 6: Transtats Map Display With Filters

In many senses, the BTS ad hoc mapping service is not really a web service. Although it accepts data in a manner much like a traditional web service, what happens after the request is processed is much more like a plain old web application – the web server returns a web page and by clicking on links on that page, the user can request updated web pages. From a programmatic standpoint, there is no reason that a more traditional web service could not be created from the building blocks already in place. In fact, in May of 2002, the GIS staff offered to create a service using SOAP, but since what the TranStats team really needed was an actual mapping application to show up within their application, the decision was made to simply return an application!

Conclusion

So is the system created by the Bureau of Transportation Statistics redundant? The resulting application framework, from the PHP Connector to the viewers based on it, BTS has achieved improved flexibility, speed of development, and stability. BTS has recreated existing applications and created applications that third parties can use to generate maps or applications based on data supplied by that third party. The creation of the BTSXML document type definition standardized all of the BTS mapping applications and provided an extensible way of organizing and even validating the parameters and data necessary to create a map.

Appendix A

Additional Information

BTS Websites

BTS Website: <http://www.bts.gov>

BTS GIS Website: <http://www.bts.gov/gis>

TranStats: <http://transtats.bts.gov>

PHP Connector Download

The Bureau of Transportation Statistics GIS Staff freely distributes the software it writes, though of course without any sorts of guarantees about stability or support. The latest version of the PHP Connector / PHP Viewer distribution can be downloaded at:

http://gisweb1.bts.gov/website/distribution/BTS_INDUS_PHP_Connector_v1.1.2.zip

In addition, any updates to this paper, including corrections, will be available at:

<http://gisweb1.bts.gov/website/distribution/p0555.htm>

Contact Information

Steve Lewis

[Bureau of Transportation Statistics](#)

Mr. Lewis is in charge of applications development by the GIS staff. Questions about distributing this software may be directed to Mr. Lewis. steve.lewis@bts.gov

Jeffrey Burka

GIS Programmer

[INDUS Corporation](#)

The PHP Connector and applications based on it were designed and implemented by Jeff Burka. Technical questions should be directed to Jeff.

jeff.burka@bts.gov or

jburka@induscorp.com

More information about PHP can be found at:

<http://www.php.net>

and

<http://www.zend.com>

Appendix B

BTSXML Document Type Definition

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!-- edited with XMLSPY v5 U (http://www.xmlspy.com) by jeffrey burka (private) -->
```

```
<!--
```

```
btsxml.dtd
```

```
version 1.0
```

```
used to pass parameters to mapping software
```

```
jburka 12/5/01
```

```
jburka 1/11/02 added the records/record tags and attributes
```

```
jburka 2/27/02 added layer and symbology
```

```
jburka 5/31/02 rearranged a few things to make the larger tags (e.g. layer) make more sense
```

```
jburka 6/4/02 added version, allowed use of title/layer or data under <btsxml>
```

```
jburka 11/8/02
```

```
cleaned up a lot of stuff to make it validate (removed () around EMPTY, etc.)
```

```
cleaned up the btsxml contents to make the dtd more logical and to properly group
```

```
items like layer and data and to allow extents to be applied to the entire map
```

```
-->
```

```
<!ELEMENT btsxml (title, extent?, (layer, data?)*, geocode?)>
```

```
<!ELEMENT title EMPTY>
```

```
<!ELEMENT layer (title, extent?, tableinfo, data?, color?, classificationtype, classnumberofbreaks)>
```

```
<!ELEMENT tableinfo (tablename, tableowner, tabledescriptivename?, variablefieldname, variablelabel, geography, geofieldname)>
```

```
<!ELEMENT data (record+)>
```

```
<!--
```

```
jburka 11/7-8/02
```

```
geocoding elements
```

```
-->
```

```
<!ELEMENT geocode (geocodeItem+)>
```

```
<!ELEMENT geocodeItem (address, score, point)>
```

```
<!ELEMENT address EMPTY>
```

```
<!ELEMENT score EMPTY>
```

```
<!ELEMENT point EMPTY>
```

```
<!--
```

```
extent to display when map is first shown
```

```
-->
```

```
<!ELEMENT extent EMPTY>
```

```
<!ELEMENT tablename EMPTY>
```

```
<!ELEMENT tableowner EMPTY>
```

```
<!ELEMENT variablefieldname EMPTY>
```

```
<!ELEMENT variablelabel EMPTY>
```

```
<!-- join field in primary -->
```

```
<!ELEMENT geography EMPTY>
```

```
<!-- join field in linked table -->
<!ELEMENT geofieldname EMPTY>
<!ELEMENT classificationtype EMPTY>
<!ELEMENT classnumberofbreaks (class*)>
<!ELEMENT class (symbol*)>
<!ELEMENT record EMPTY>
<!ELEMENT symbol EMPTY>
<!ELEMENT color EMPTY>
<!ELEMENT tabledescriptivename EMPTY>
<!--version of the btsxml spec -->
<!ATTLIST btsxml
version CDATA #REQUIRED
>
<!--a name contains the title for the map -->
<!ATTLIST title
name CDATA #REQUIRED
>
<!-- name contains the internal name of the layer; the layer name is not drawn on the map.
fromlayer specifies the feature layer in the application which will be used and is not required (nor really expected)-->
<!ATTLIST layer
name CDATA #REQUIRED
fromlayer CDATA #IMPLIED
>
<!-- geographic extent of the layer; if included, the map will be displayed showing the extent, otherwise the default extent of the application will
be used -->
<!ATTLIST extent
xmin CDATA #REQUIRED
ymin CDATA #REQUIRED
xmax CDATA #REQUIRED
ymax CDATA #REQUIRED
>
<!-- name is a required attribute for both TABLENAME and TABLEOWNER. however,
if the DATA tag is used in a layer and records are embedded in the XML stream, the service will create an oracle temp table containing the
data and then replace the TABLEOWNER and TABLENAME name attribute with the generated values. as such, if you are passing in DATA,
use an empty string ("") for the name attribute of these two tags. -->
<!ATTLIST tablename
name CDATA #REQUIRED
>
<!-- name of oracle table containing attribute data to be joined to the feature table -->
<!ATTLIST tableowner
name CDATA #REQUIRED
>
<!-- owner of oracle table containing attribute data -->
<!ATTLIST tabledescriptivename
name CDATA #IMPLIED
>
<!-- optional description of data in table -->
<!ATTLIST variablefieldname
name CDATA #REQUIRED
>
<!-- attribute field in tablename to be mapped -->
<!ATTLIST variablelabel
name CDATA #REQUIRED
>
<!-- label to use for attribute field in the map's legend -->
<!ATTLIST geography
level (state | county | cdist | dot) "state"
>
<!-- type of feature to be mapped: state, county, congressional district, or DOT region -->
<!ATTLIST geofieldname
```

```

name CDATA #REQUIRED
>
<!-- the name of the field in tablename used to join tablename to the feature table -->
<!ATTLIST classificationtype
name (custom | quantile | equalinterval) "quantile"
>
<!-- type of classification scheme -->
<!ATTLIST classnumberofbreaks
count CDATA "5"
>
<!-- number of breaks to use when generating quantile or equal interval breaks -->
<!--color specifies the beginning and ending colors of the symbols' color ramp. it is optional and if not included, a default ramp of white to red
will be used. -->
<!ATTLIST color
start CDATA #REQUIRED
end CDATA #REQUIRED
>
<!-- lower and upper values for a custom class. range is used to define the equality tests as follows:
all: lower <= value <= upper
lower: lower <= value < upper
upper: lower < value <= upper
-->
<!ATTLIST class
lower CDATA #REQUIRED
upper CDATA #REQUIRED
equality (lower | upper | all) "lower"
>
<!-- a custom symbol for a class; if this is not specified you probably want to make use of the COLOR tag to specify start and end colors for
the system to ramp-->
<!ATTLIST symbol
color CDATA #REQUIRED
style CDATA #REQUIRED
size CDATA #REQUIRED
>
<!-- a record for embedded data. fips is the join attribute (even if you're not using fips as the foreign key!); value is feature value -->
<!ATTLIST record
fips CDATA #REQUIRED
value CDATA #REQUIRED
>
<!--
Address that's been geocoded
-->
<!ATTLIST address
value CDATA #REQUIRED
>
<!--
Score of the geocode
-->
<!ATTLIST score
value CDATA #REQUIRED
>
<!--
Point location for address found through geocoding
-->
<!ATTLIST point
x CDATA #REQUIRED
y CDATA #REQUIRED
>

```

Sample BTSXML With Embedded Data

```

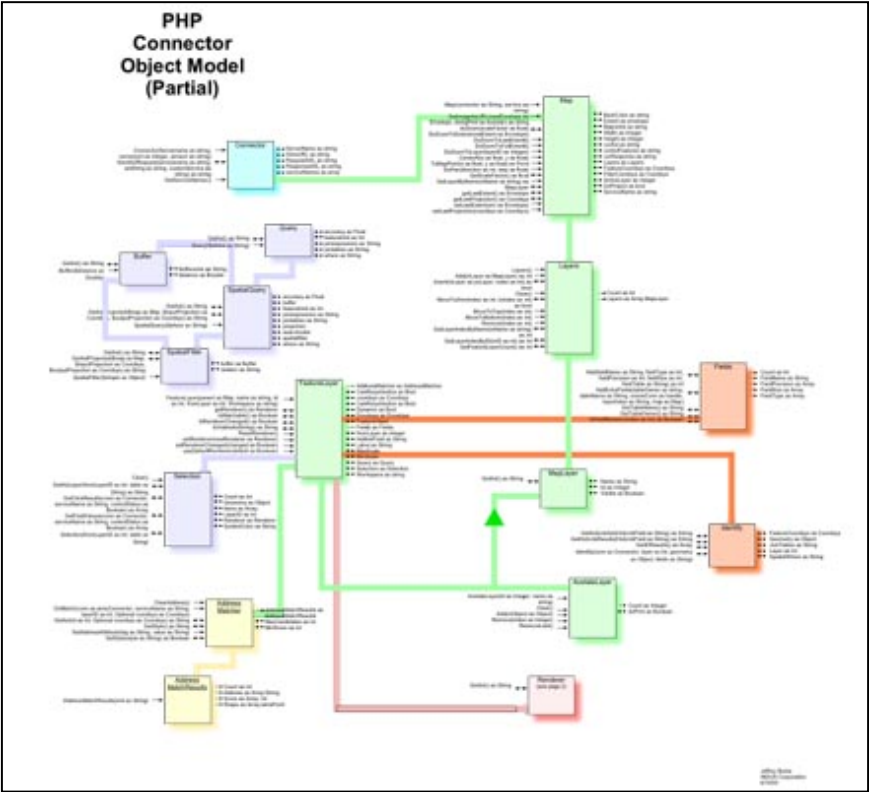
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE btsxml SYSTEM "http://websas.bts.gov/website/xml/btsxml.dtd">
<btsxml version="1.0">
<title name="state sample"/>
<layer name="Sum of Passengers">
<title name="Sum of Passengers" />
<tableinfo>
<!--tableinfo, tablename, and tableowner are blank because the table will be created by a php script-->
<tablename name=""/>
<tableowner name=""/>
<variablefieldname name="value"/>
<variablelabel name="sum of passengers"/>
<geography level="state"/>
<geofieldname name="state_fips"/>
</tableinfo>
<data>
<!-- data is abbreviated -->
<record fips="01" value="15087542"/>
<record fips="02" value="17225371"/>
<record fips="04" value="111922861"/>
<record fips="05" value="9955428"/>
<record fips="06" value="455480536"/>
<record fips="08" value="120709183"/>
<record fips="09" value="18547274"/>
<record fips="10" value="7434"/>
<record fips="11" value="81098684"/>
<record fips="12" value="301946256"/>
<record fips="13" value="226569946"/>
<record fips="15" value="95194665"/>
<record fips="16" value="9237257"/>
<record fips="17" value="242321692"/>
<record fips="18" value="27540770"/>
<record fips="19" value="8370159"/>
<record fips="20" value="3925406"/>
<record fips="21" value="14256828"/>
<record fips="22" value="35561152"/>
<record fips="23" value="3906165"/>
<record fips="24" value="46531653"/>
<record fips="25" value="70790933"/>
<record fips="26" value="112406650"/>
<record fips="27" value="94670812"/>
<record fips="28" value="5293933"/>
</data>
<color start="254,241,31" end="255,0,0"/>
<classificationtype name="quantile"/>
<classnumberofbreaks count="5"/>
</layer>
</btsxml>

```

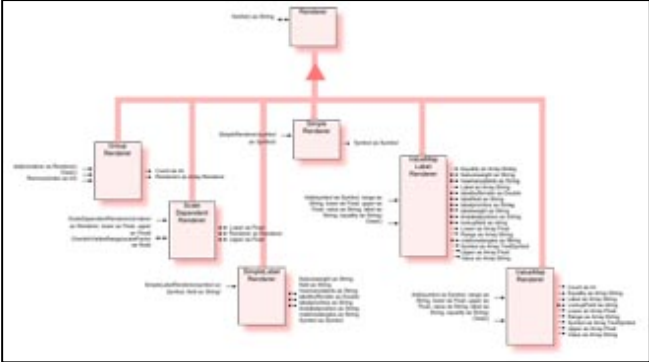
Appendix C

PHP Connector Object Models

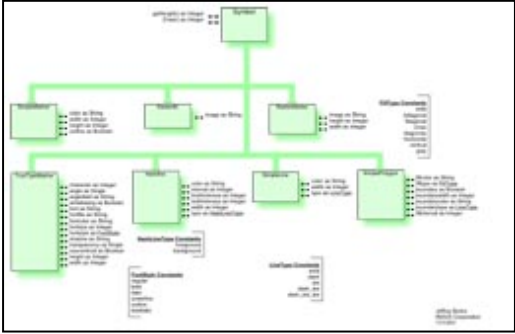
The images of the object model are rather large. Click on an image below to see a full-sized version.



PHP Connector Object Model (main)



PHP Connector Object Model (renderers)



PHP Connector Object Model (symbols)