



**From Net.Data to PHP:
A Comparison Study**

*IBM System i
PHP Speed Team*

November 2006

Abstract: This article is the second of two parts written by the IBM PHP Speed Team from the summer of 2006. In addition to the authors, the members include Margaret Myslinska (University of Chicago, Urbana Champaign), Andrew Nere (St. Cloud State University), David Swenson (Minnesota State University, Mankato), and Gloria Velazquez (University of Texas at El Paso). The article comes from their research and experience during the course of their work on the team. The goals for the team included the following:

- **Attain hands on experience with PHP on the IBM System i Platform**
- **Gain insight into PHP through the use of PHP applications**
- **Port PHP applications from MySQL to DB2**
- **Summarize knowledge, research, and experience gained for IBM in two developerWorks articles, including this one**

Introduction

This article is intended to compare the scripting languages of PHP and Net.Data on System i hardware, specifically on i5/OS. It focuses on the origins of the language, differences in syntax, learning curves, database connections, and other uses. Listings 1-6 represent code segments of the same program, an example created in both PHP and Net.Data to populate and display tables. This paper includes a performance comparison for this program as well.

Overview of PHP and Net.Data

Origins of PHP and Net.Data

PHP is one of the most widely used Web programming languages today. PHP is a powerful server-side scripting language that can be used to create dynamic websites and applications. PHP began as “Personal Home Page”, a simple Perl hack written by Rasmus Lerdorf in 1994. Today, PHP has come to mean “PHP: Hypertext Preprocessor”. A big change to PHP came in 1997, when Zeeve Suraski and Andi Gutmans produced PHP 3. Shortly afterwards, they created Zend Technologies, the company that produces the Zend Engine for PHP.

On July 13, 2004, the latest version of PHP, PHP 5, was released. Prior to this, PHP 3 saw the introduction of object-oriented functionality to the language. PHP 4 added the ability to pass and return by reference. The object-oriented functionality was rewritten for PHP 5, allowing for better performance with more functionality and features.

Today, there literally exist thousands of PHP applications, many of which are open source. They can be downloaded, used, and modified freely. Shopping carts, bulletin boards, and wikis are all popular uses of this powerful language.

Net.Data is also a server-side scripting language that extends Web servers by enabling the dynamic generation of Web pages. The origins of Net.Data come from the web macro capability of the DB2 WWW Connection from Santa Teresa Lab. The DB2 WWW Connection was first released on OS/2 and AIX in November of 1995. By its second release in July of 1996, the project had been renamed Net.Data and was available in a beta for both systems. Later that year it was made Generally Available (GA) on the AS/400 with OS/400 V3R7 and also ported back to V3R2.

One of the crowning moments of Net.Data was its use in the creation of the dynamic web pages of the 1998 Winter Olympics at Nagano. There were nearly 650 million hits during the 16-day event, reaching a peak volume of 110,414 hits per minute.

The data for these dynamic web applications comes from relational and non-relational database management systems such as DB2, (DRDA)-enabled databases, and flat file data. Net.Data applications can be rapidly built using a scripting language that is simple yet powerful.

Popularity

PHP has become one of the most widely used Web programming languages today. PHP is a powerful server-side scripting language that can be used to create dynamic websites and applications. With over 4.5 million PHP developers and over 22 million Internet domains using PHP, it is easy to see the popularity of this technology.

While Net.Data is not nearly as common as PHP, it still is very popular among IBM's customers, especially those that use the System i platform and i5/OS. It provides a way for businesses to get their data on the web without a lot of education and resources. No specific numbers have been collected on number of developers, but there are a large number of IBM's customers who still use and love the product.

Platforms

PHP is platform independent, meaning it works with a wide range of operating systems. Net.Data was also designed to be a cross platform language. Net.Data has been available for many years on a variety of platforms, including Windows NT, AIX, HP-UX, Linux, Sun Solaris, OS/390 and i5/OS. Net.Data is currently only supported for i5/OS and OS/390.

Available Development Environments

PHP can be developed in several different environments. It can be written in general editing tools such as Notepad and other Notepad editors, many of which recognize PHP syntax. Zend Studio is an Integrated Development Environment (IDE) specifically created for PHP development and is free for System i customers on Zend's website, http://www.zend.com/products/zend_core/zend_core_for_ibm. No specific development environments have been created for Net.Data. Net.Data is developed in Notepad editors.

Syntax of the Languages

Format of code

All PHP code begins and ends with a PHP tag. Each line in PHP must end with a semicolon, similar to C. This distinguishes one set of instructions from the next. White space is ignored between PHP statements.

The Net.Data macro language has only a few rules about programming format. This simplicity provides programmers with freedom and flexibility. A single instruction can span many lines, or multiple instructions can be entered on a single line. Instructions can begin in any column. Spaces or entire lines can be skipped. Comments can be used anywhere.

Use of Variables

In PHP, all variables except defined constants must begin with a '\$'. After the '\$', all variables names must begin with either an underscore or letter. All variables are case sensitive.

Variables in PHP are weakly typed, that is, a variable can be assigned as an integer and later as a character. Variables are always assigned by value, which means that when one variable is set equal to another variable, they are then independent of each other. In order to assign by reference, a '&' must be used. Math and logic operators are handled in PHP the same way as they are in C language. PHP also has automatic type conversion for arithmetic operations. If any of the operands is a float, then all operands are evaluated as floats, and the result will be a float. Otherwise, the operands will be interpreted as integers, and the result will also be an integer. The type of a variable is usually not set by the programmer; rather, it is decided at runtime by PHP depending on the context in which that variable is used. Supported variables include integers, floating point numbers, strings, arrays, and objects

In Net.Data, variables also must start with an underscore or letter and are case sensitive. Net.Data regards all data as character strings. Content of the variables can change like PHP. However, this is not because Net.Data decides at runtime what each character type is. Since all character data is a string, Net.Data uses built-in functions to perform arithmetic operations on a string that represents a valid number, including those in exponential formats.

Net.Data has three types of variables. The first type are the variables specifically defined by the user using the %DEFINE statement (define in PHP). Variables that make up the second type are pre-defined and made available by Net.Data. The third type of variables constitutes those that are a part of a %FUNCTION block and they can only to be referenced within that block. Also, there are five types of scope that these three types of variables can have, depending on how and where they were declared. These scopes are: global macro file, function block, report block and row block.

Learning Curve

PHP is very simple to learn. The syntax of PHP is very similar to that of Java, C, and Perl, so programmers who have ever seen these languages will be very comfortable with PHP. Math and logic operators are handled in PHP the same way as they are in the C language.

The Net.Data scripting language is also easy to learn and allows the use of existing skills to rapidly develop web applications. Net.Data is a favorite of many IBM customers because of its easy learning curve.

Databases

Connection to Database

Net.Data can connect to relational and non-relational database management systems such as DB2 and DRDA-enabled databases. DRDA (Distributed Relational Database Architecture) is an industry standard from The Open Group for accessing database information across IBM platforms that follows SQL standards. It is also possible for Net.Data to connect to the DB2 database using the ODBC connection.

PHP can also connect to multiple database systems, including PostgreSQL, MySQL, Access, and DB2. Connection to IBM's DB2 database can use either the ODBC driver or the IBM_DB2 connection driver. The ODBC driver is a bit older than IBM DB2 driver, but since ODBC isn't necessarily used specifically for the DB2 database, it is more often used by open source applications.

DB2 Database Connection

There are many differences between PHP and Net.Data when connecting to DB2. PHP has two main drivers available, ODBC and IBM_DB2 where as Net.Data has one interface to DB2 for i5/OS. In the PHP example created for this article, the IBM_DB2 driver is used. The IBM_DB2 drivers are specific to the DB2 database, and therefore may relieve some of the connectivity issues that have arisen in the past with ODBC. IBM_DB2 Connect implements the DRDA architecture to reduce the complexity and cost of accessing data stored in the DB2 database.

With PHP, to connect to DB2, the `db2_connect()` function is used with the parameters as listed in Listing 1. Once the link is successfully connected, it returns a connection handle resource that remains active until it is closed using the `db2_close()` function. With Net.Data the connection is done differently. A DATABASE variable is defined initially within the macro that tells it which database to connect to. Once the macro is interpreted the connection is active and ready to execute SQL statements. See Listing 2. To show SQL statements in PHP, the `echo()` function must be used for each statement. Displaying a SQL statement in Net.Data is as simple as defining `SHOWSQL = "YES"`.

```
$i5user = strtoupper($i5user);
i5host = strtoupper($i5database);
$i5link = db2_connect($i5host, $i5user, $i5password, array("i5_commit" => B2_I5_TXN_READ_COMMITTED,
    'autocommit' => DB2_AUTOCOMMIT_OFF));
if (!$i5link) die('Failed connect: '. db2_conn_error(). " : ". db2_conn_errormsg());
```

Listing 1: PHP Connection to DB2 using IBM_DB2

```
%DEFINE {
    DATABASE    = "D1092546"
    TABLE     = "SHAZAM.NETTABLE"
% }
```

Listing 2: Net.Data Connection to DB2

Execution of SQL statements:

In PHP, SQL statements are run using either the `db2_exec()` or `db2_execute()`. The two commands are similar in the fact that they both will execute a SQL query; however results that are returned differ greatly. The `db2_execute()` statement will return true or false once completed and the result set of the query must be bound to a variable before the SQL statement is run. With `db2_exec()`, the result of a successful query is a statement resource that contains all of the requested data and an unsuccessful query is false. Listing 3 shows how PHP implements the `db2_exec()` command. The connection handle resource that is given when a database connection is created is a necessary parameter along with the query that needs to be run. Listing 3 shows the `db2_exec()` command inserting into a table using a nested for-loop. In comparison, Net.Data requires a function to be created using the DTW_SQL language environment which converts the

character string contained within or passed into the function to a correct SQL statement. Once converted the SQL statement is executed, the results are returned from the function. The %MACRO_FUNCTION of Listing 4 shows six insert statements executed 40 times via a while-loop. It is important to note that all snippets of code are portions of the same program used to create and populate a table in both PHP and Net.Data. Therefore a table has to be created before running this code. Here is the create statement:

```
CREATE TABLE SCHEMA.TABLENAME
(CAT INT NOT NULL GENERATED ALWAYS AS IDENTITY(START WITH 1
  INCREMENT BY 1),
SUBCAT varchar(15) NOT NULL,
PRDNAM varchar(25) NOT NULL);

ALTER TABLE SCHEMA.TABLENAME ADD CONSTRAINT
  SCHEMA.CONSTRAINTNAME PRIMARY KEY (CAT);
```

```
//Insert Data
$data[0] = "INSERT INTO $schema.$table (SUBCAT, PRDNAM) VALUES ('HJ', 'WINTER')";
$data[1] = "INSERT INTO $schema.$table (SUBCAT, PRDNAM) VALUES ('GT', 'HAHAHA')";
$data[2] = "INSERT INTO $schema.$table (SUBCAT, PRDNAM) VALUES ('QW', 'SUMMER')";
$data[3] = "INSERT INTO $schema.$table (SUBCAT, PRDNAM) VALUES ('AB', 'GOOD')";
$data[4] = "INSERT INTO $schema.$table (SUBCAT, PRDNAM) VALUES ('HF', 'SPRING')";
$data[5] = "INSERT INTO $schema.$table (SUBCAT, PRDNAM) VALUES ('SK', 'FALL')";

for($j=0; $j<40; $j++) {
  for($i=0; $i<count($data); $i++){
    @db2_exec($i5link, $data[$i]) or die("Failed insert: ' . db2_stmt_error()." .db2_stmt_errormsg());
  }
}

//Committing insert
db2_commit($i5link);

// Execute Query
$query = "SELECT cat, subcat, prdnam
        FROM $schema.$table
        order by cat, subcat, prdnam
        for fetch only";
$result = db2_exec($i5link, $query);
```

Listing 3: PHP SQL Execution

```
%MESSAGE {
  default : { <p align="center"><font
color="red"><b>$(DTW_DEFAULT_MESSAGE)...</b></font></p>
  } : continue
%}

%FUNCTION(DTW_SQL) execSql(sqlstmt) {
  $(sqlstmt)
%}

%MACRO FUNCTION crtTables() {
  @DTW ASSIGN(COUNT, "1")
  @DTW ASSIGN(END, "40")
  %WHILE ( COUNT <= END ) {
    @execSql("INSERT INTO $(schema).$(table) (SUBCAT, PRDNAM) VALUES ('HJ', 'WINTER')")
    @execSql("INSERT INTO $(schema).$(table) (SUBCAT, PRDNAM) VALUES ('GT', 'HAHAHA')")
    @execSql("INSERT INTO $(schema).$(table) (SUBCAT, PRDNAM) VALUES ('QW', 'SUMMER')")
    @execSql("INSERT INTO $(schema).$(table) (SUBCAT, PRDNAM) VALUES ('AB', 'GOOD')")
    @execSql("INSERT INTO $(schema).$(table) (SUBCAT, PRDNAM) VALUES ('HF', 'SPRING')")
    @execSql("INSERT INTO $(schema).$(table) (SUBCAT, PRDNAM) VALUES ('SK', 'FALL')")
    @DTW_ADD(COUNT, "1", COUNT)
  }
%}

%FUNCTION(DTW SQL) getinv() {
  SELECT cat, subcat, prdnam
  FROM $(schema).$(table)
  order by cat, subcat, prdnam
  for fetch only
%}
```

Listing 4: Net.Data Execution of SQL Statements

Output Results

Outputting the result set using PHP is slightly more complicated than using Net.Data. PHP requires the use of two or more IBM_DB2 functions to extract the rows and fields out of the results set. In our example in Listing 5, `db2_fetch_row()` and `db2_result()` are used to extract the data from the statement resource. The `db2_fetch_row()` is used to iterate through the statement resource returning true till there are no more rows in the resource. On each iteration, the `db2_result()` is used to get the values for each field within the row. Then formatting the output of the data is a matter of either using the `printf()` function which is similar to C/C++ or proper HTML table tags. In Net.Data, the data is either outputted directly from the query results or are assigned to a table using the `@DTW_ASSIGN(DTW_HTML_TABLE,"YES")`. See Listing 6. There is no need to use HTML or other ways to format the output because Net.Data will handle all the basic formatting. Another way of formatting in Net.Data is to generate a SQL report.

```
// Output Query
if ( $_POST["submit"] == "NO TABLES" ) {
    global $result, $query;

    echo '
        <FORM action="DBExampleNetComp.php" method="POST">
            <input type=hidden name=method value=createTable>
            <input type="submit" name="submit" value="TABLES" />
        </FORM><PRE>';
    printf("%9s", "-----\n");
    printf("|%s|%s|%s|\n", "CAT", "SUBCAT", "PRDNAM");
    printf("%9s", "-----\n");
    while (db2_fetch_row($result)) {
        printf("|%3s|%5s|%7s|\n", db2_result($result, 0), db2_result($result, 1), db2_result($result, 2) );
        printf("%9s", "-----\n");
    }
    echo '<PRE>';
    list($totalSeconds, $extraMilliseconds) = timeAndMilliseconds();
    fwrite($resultFile, "[" . date('Y-m-d H:i:s') . "] - No Tables: Start Time: ;" . $startTime .
        ";; - End Time: ;" . date("H:i:s", $totalSeconds) . ".$extraMilliseconds;\n");
}
else {
    global $result, $query;

    echo '
        <FORM action="DBExampleNetComp.php" method="POST">
            <input type=hidden name=method value=createTable>
            <input type="submit" name="submit" value="NO TABLES" />
        </FORM><TABLE border=1><TR>
            <TD>CAT</TD> <TD>SUBCAT</TD> <TD>PRDNAM</TD> </TR>
    ';
    while (db2_fetch_row($result)) {
        echo '<TR>';
        echo '<TD>' . db2_result($result, 0) . '</TD>';
        echo '<TD>' . db2_result($result, 1) . '</TD>';
        echo '<TD>' . db2_result($result, 2) . '</TD>';
        echo '</TR>';
    }
    echo "</TABLE>";
}
}
```

Listing 5: PHP SQL results output

From Net.Data to PHP: A Comparison Study

```
%HTML(main) {
  @DTW_ASSIGN(filename, $(fname))
  @DTWF_OPEN(filename, "a", "850")
  @DTW_TIME("X", STARTTIME)
  <HTML>
  <HEAD>
  <TITLE>Basic SQL Example</TITLE>
  </HEAD>

  <BODY bgcolor="white">
  <CENTER>
  <H1>Basic SQL Example</H1>
  <P><A HREF="/netdata/basic_sql.mac/table">Switch to Table format</A></P>

  @crtTables()
  @getinv()

  </CENTER>
  </BODY>
  </HTML>
  @DTW_TIME("X", ENDTIME)

  @DTWF_WRITEFILE(filename, "Start Time: ;$(STARTTIME); -- End Time: ;$" .
    "(ENDTIME); | ", "Y")
  @clrTables()
  @DTWF_CLOSE(filename)
%}

%HTML(table) {
  @DTW_ASSIGN(filename, $(fname))
  @DTWF_OPEN(filename, "a", "850")
  @DTW_TIME("X", STARTTIME)
  <HTML>
  <HEAD>
  <TITLE>Basic SQL Example</TITLE>
  </HEAD>
  <BODY bgcolor="white">
  <CENTER>
  <H1>Basic SQL Example</H1>
  <P><A HREF="/netdata/basic_sql.mac/main">Switch to Text format</A></P>

  @DTW_ASSIGN(DTW_HTML_TABLE,"YES")
  @crtTables()
  @getinv()

  </CENTER>
  </BODY>
  </HTML>
  @DTW_TIME("X", ENDTIME)
  @DTWF_WRITEFILE(filename, "Start Time: ;$(STARTTIME); -- End Time: ;$" .
    "(ENDTIME); | ", "Y")
  @clrTables()
  @DTWF_CLOSE(filename)
%}
```

Listing 6: Net.Data SQL results output

Output of PHP and Net.Data

Listings 7 and 8 show the output of the PHP code with and without tables while Listings 9 and 10 show the same output created using Net.Data. Note again that these outputs are the result of the code snippets in each of the previous figures.

```

Basic SQL Example

    TABLES

-----
| CAT | SUBCAT | PRDNAM |
-----
|  1 |   HJ   | WINTER |
-----
| 10 |   GT   | HAHAHA |
-----
|100 |   QW   | SUMMER |
-----
    
```

Listing 7: PHP Output without tables

```

Basic SQL Example

    NO TABLES



| CAT | SUBCAT | PRDNAM |
|-----|--------|--------|
| 1   | HJ     | WINTER |
| 10  | GT     | HAHAHA |
| 100 | QW     | SUMMER |


```

Listing 8: PHP Output with tables

```

Basic SQL Example

    Switch to Table format

    CAT | SUBCAT | PRDNAM |
-----
    1  | HJ     | WINTER |
-----
   10 | GT     | HAHAHA |
-----
  100 | QW     | SUMMER |
-----
    
```

Listing 9: Net.Data Output without table

```

Basic SQL Example

    Switch to Text format

| CAT | SUBCAT | PRDNAM |
|-----|--------|--------|
| 1   | HJ     | WINTER |
| 10  | GT     | HAHAHA |
| 100 | QW     | SUMMER |


```

Listing 10: Net.Data Output with tables

Database Performance Comparison

To compare the performance of DB access from PHP and Net.Data scripts, a workload was created using the previously described example. The SilkPerformer tool was used to drive 10 concurrent users, each executing the same workload using unique tables for each user. From this test, the transaction times and throughput (transactions per second - TPS) were recorded. Then, with the TPS data the capacity of PHP relative to Net.Data was computed by normalizing the TPS values of each run (TPS/TPS_{net.data}). System resources were monitored to ensure that

there were no system bottlenecks, such as disk or memory constraints, and that the CPU was driven to 100%.

Net.Data was run first to establish a baseline. It is important to note that with Net.Data, SQL statements are run in-line in the Net.Data process that establishes the DB connection. In contrast, with PHP you can specify if the SQL requests are run in-line in the PHP process (ZENDCOREAP jobs), or if the SQL requests are submitted to run in server mode, i.e., processed by the QSQRVR jobs. Another difference between Net.Data and PHP is that PHP has the capability to cache the prepared statements so that the prepare is done once, not on every execute. Net.data on the other hand only supports execute immediate, where the prepare is done each time a statement is executed. To see how to change the source code in order to use prepared statements refer to Listing 11. Listing 12 shows how to define the connection parameters so that the SQL statements run in-line. In order to compare directly to Net.Data, the results in Listing 13 for the PHP environment utilize in-line SQL Processing. As stated above, PHP also supports server mode SQL processing where the QSQRVR host server jobs process the SQL requests. From a performance perspective, in-line SQL processing is recommended; however, if you need the SQL requests to be processed under unique user-ids with specific authorities, you will need to use server mode processing.

```
// Insert Data
$data = array(
    array('HJ', 'WINTER'),
    array('GT', 'HAHAHA'),
    array('QW', 'SUMMER'),
    array('AB', 'GOOD'),
    array('HF', 'SPRING'), array('SK', 'FALL') );

$insert = "INSERT INTO $schema.$table (SUBCAT, PRDNAM) VALUES (?, ?)";
// Preparing insert statement
$stmt = db2_prepare($i5link, $insert);

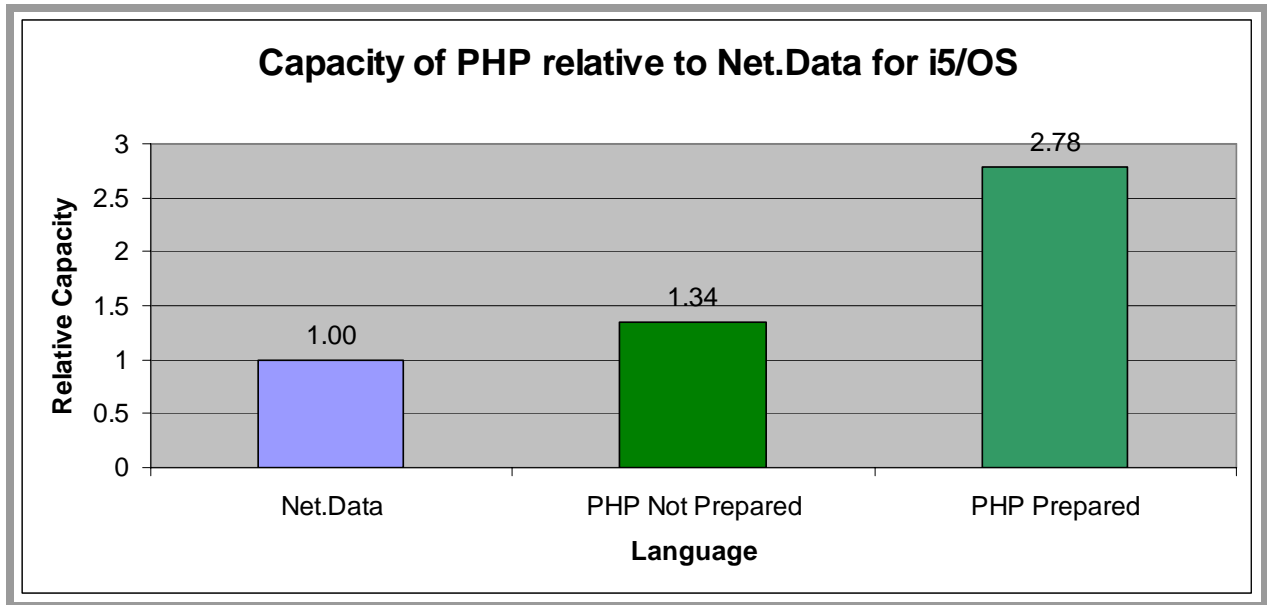
//Implements for Large Data group
for($j=0; $j<40; $j++) {
    for($i=0; $i<count($data); $i++){
        @db2_execute($stmt, $data[$i]) or die('Failed insert: ' . db2_stmt_error()." : ".db2_stmt_errormsg());
    }
}

db2_commit($i5link);
```

Listing 11: Preparing the insert statements

```
$i5link = db2_connect("", "", array("i5_commit" => DB2_I5_TXN_READ_COMMITTED,
    'autocommit' => DB2_AUTOCOMMIT_OFF));
if (!$i5link) die('Failed connect: ' . db2_conn_error()." : ".db2_conn_errormsg());
```

Listing 12: Connecting to DB2 with default parameters (running in-line)



Listing 13: Performance Results

Please note that this is a DB intensive workload. In this case, significantly more PHP users can be supported versus Net.Data, particularly if you utilize the best practice of using prepared statements, ie, preparing a statement once, not on each execute.

Keep in mind that this data was obtained with the latest available versions of both Net.Data and PHP and is subject to change as both scripting languages evolve. Also, Results listed here do not represent any particular customer environment. Actual performance may vary significantly from what is provided here.

Other Uses and Functions

Use with HTML

Because both Net.Data and PHP are often used to create websites and online applications, it is important that they are able to easily produce HTML, both static and dynamic. PHP is especially suited for web development using HTML and can be embedded into it using simple PHP tags that let the user go into and out of the PHP mode quickly and easily. An example is shown by Listing 14 bellow.

```
<html>
  <head>
    <title>Hello World Example</title>
  </head>
  <body>
    <?php
      echo "Hello World!";
    ?>
  </body>
</html>
```

Listing 14: PHP Embedded in HTML

Embedded Net.Data in HTML is also fairly simple, but requires the use of blocks. An example of that is shown by Listing 15 below.

```
%HTML>HelloWorld) {
  <html>
  <head><title>Hello world Example</title></head>

  <body>
  <P>Hello World! From HelloWorld
  </body>
  </html>
  % }
```

Listing 15: Net.Data using HTML

Including Other Files

Both PHP and Net.Data allow developers to include source files in an application. This allows applications to be created without writing all HTML and PHP or Net.Data in a single file. Net.Data uses the function %INCLUDE followed by the file path enclosed in quotation marks. PHP uses the include function followed by the file path as well.

Calling i5/OS Commands and Programs

Besides database connectivity and use with HTML, a scripting language may need to access other system resources, such as programs and commands. Both Net.Data and PHP provide methods of calling external programs written in high-level programming languages including COBOL, RPG, and C/C++. One method is to register the external program as a stored procedure and then use the SQL Call statement to execute the program with a call to the stored procedure. The other method consists of proprietary functions for running the external programs. To call a program, Net.Data provides the function call %FUNCTION (DTW_DIRECTCALL). See Listing 16. PHP uses a set of three function calls. See Listing 17. First, the i5_connect() function must be used to connect to i5/OS, followed by the i5_program_prepare() function to set up the program and parameters. Finally, the program is called with the i5_program_call() function.

```
%function(DTW_DIRECTCALL dc1() {  
%EXEC { /QSYS.LIB/NETDATA.LIB/MYPGM.PGM % }  
% }
```

Listing 16: Net.Data Program Call

```
//First, i5 connection must be established  
$conn = i5_connect("localhost". "userid". "password");  
if(!$conn) die("Failed connection");  
  
//If connected, prepare the program  
$prog = i5_program_prepare("PHP/PROGRAM". $other_params);  
  
//Finally, we make the program call  
i5_program_call($prog, $params);
```

Listing 17: PHP Program Call

To enable calling i5/OS system commands, Net.Data provides the DTW_SYSTEM. See Listing 18. PHP uses a set of two function calls to accomplish the same, as seen in Listing 19. First, the `i5_connect()` function must be used to connect to the i5/OS and then the `i5_command()` function can be used to make system calls to commands and APIs.

```
%FUNCTION(DTW_SYSTEM) sys1() {  
%EXEC { /QSYS.LIB/ADDLIBLE.CMD LIB(MYLIBRARY) % }  
% }
```

Listing 18: Net.Data System Call

```
//First, i5 connection must be established  
$conn = i5_connect("localhost". "userid". "password");  
if(!$conn) die("Failed connection");  
  
//If connected, make command call  
i5_command("COMMAND");
```

Listing 19: PHP System Call

Net.Data and PHP Use Each Other

It is possible for PHP and Net.Data to call each other. This is done by passing parameters through the URL. Both PHP and Net.Data allow this. Both languages are able to pass and receive parameters through the URL string.

Net.Data -> `http://system:8090/nd/macro/test?parm1=somedata&parm2=somedata`

PHP -> `http://system:89/phpapp/test?parm1=somedata&parm2=somedata`

This means that it is possible to have interaction between PHP and Net.Data applications.

Support

Both Net.Data and PHP have product support. Net.Data is supported by IBM but only for i5/OS and OS/390. Until September 30, 2004, Net.Data had been supported on HP-UX, Linux, and Windows NT. Zend provides standard Internet support for three years for both the Zend Core and Zend Studio Professional for i5/OS. Gold and Platinum phone support can be purchased for a fee.

There also exists unofficial development support online for both languages. PHP has a formal development manual available online at <http://us2.php.net/>. This manual is updated and maintained by the open source community. PHP users can help each other using forums and websites. There are many tutorials and beginner sites available for PHP, including <http://www.php.net> and <http://www.zend.com>. The websites <http://dtwdude.com> and www.ignite400.org are places for Net.Data users to find support.

Conclusion

Net.Data has been a favorite of many System i customers in the past. It has helped them create dynamic web applications to use for their business. With the availability of Zend's PHP products on i5/OS, PHP provides another scripting option to System i customers with thousands of readily available open source applications. Both languages are easy to use and develop on the System i platform, with readily available support too. System i customers now have two great choices in dynamic web application development.

Resources

Nadir Amra. IBM Employee and Developer/Support for Net.Data

“Net.Data Administration and Programming Guide for OS/400.” IBM Manuals. October 2001.
15 July 2006.

<<http://www-03.ibm.com/servers/eserver/series/software/netdata/dtwa2mst.pdf>>

“Net.Data Reference.” IBM Manuals. 15 July 2006.

<<http://www-03.ibm.com/servers/eserver/series/software/netdata/db2rn.pdf>>

“Net.Data Language Environment Interface Reference.” IBM Manuals. 15 July 2006.

<<http://www-03.ibm.com/servers/eserver/series/software/netdata/db2ln.pdf>>

“Net.Data Return Codes.” IBM Manuals. 15 July 2006.

<<http://www-03.ibm.com/servers/eserver/series/software/netdata/db2mn.pdf>>

“Net.Data Brochure.” IBM White Papers. 15 July 2006.

<<http://www-03.ibm.com/servers/eserver/series/software/netdata/news/broch2.htm>>

“Zend Solutions for Businesses Evaluating PHP.” Zend the PHP Company. 15 June 2006.

<http://www.zend.com/company/solutions/for_businesses_evaluating_php>

Stogov, Dmitry. “PHP SOAP Extension.” March 16, 2004. 13 June 2006.

<<http://www.zend.com/php5/articles/php5-SOAP.php>>

Miller, Chris Alan. “iSeries News: Native PHP Ready to Burst onto the i5 Scene.”

4 April 2006

<<http://www.iseriesnetwork.com>>

Morgan, Timothy P. “PHP is Almost Certainly Coming to the iSeries.”

ITJungle 14 (2005). 12 June 2006

<<http://www.itjungle.com>>

Morgan, Timothy P. “Business Review: PHP Will Soon Be Native on the System i5.”

3 April 2006

<<http://www.cbronline.com>>

From Net.Data to PHP: A Comparison Study

Disclaimer: This article is intended to educate the audience on PHP and Net.Data scripting languages. The information included in this article is a product of research and development of programs in both Net.Data and PHP. It is meant to compare the different aspects of these languages, such as syntax, database connection, and performance.

Copyright IBM Corporation 1994-2006. All rights reserved. References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM System i5 System i i5/OS Net.Data

Intel, Intel Logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM.

Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.