Data Exchange with XML, WDDX and PHP

By Vikram Vaswani

Table of Contents

Talking Nice

Let me be completely frank with you.

If you've never had to sit up nights figuring out how to share data between Web servers located on different sides of the planet, each speaking a different languages, or missed a date because you were struggling to get your PHP code to talk to your customer's legacy Perl code, this article isn't going to do much for you. As a matter of fact, you'll probably read it, yawn and wonder if television might not have been a better use for your time.

If, on the other hand, you're the kind of person whose work involves acting as translator between different, often−incompatible systems, conceptualizing new and more efficient ways for back−office systems to integrate with each other, or exchanging data between multiple Web applications in a robust way, you're going to read this article and start weeping buckets, because someone somewhere has not only understood your problem, but come up with a wonderfully elegant solution.

Am I joking? Nope − read on to find out about the wonder that is WDDX. And keep those Kleenex handy.

The Wonderful World Of WDDX

It's quite likely that you've never heard of WDDX before today − so allow me to enlighten you:

WDXX, or Web Distributed Data Exchange, is a mechanism for representing and exchanging data structures (like strings and arrays) between different platforms and applications. It uses XML to create abstract representations of data, application−level components to convert the XML abstractions into native data structures, and standard Internet protocols like HTTP and FTP as the transport mechanism between applications.

Still confused, huh?

All right, let's try English for a change.

WDDX is simply a way of representing data − strings, numbers, arrays, arrays of arrays − in a manner that can be easily read and understood by any application. To illustrate this, consider the following PHP variable:

```
<?
$colour = "tangerine";
?>
```

Here's how WDDX would represent this variable:

```
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='colour'>
<string>tangerine</string>
</var>
</struct>
</data>
</wddxPacket>
```

By creating an abstract representation of data, WDDX makes it possible to easily exchange data between different applications − even applications running on different platforms or written in different languages − so long as they all understand how to decode WDDX−encoded data. Applications which support WDDX will receive application−neutral WDDX data structures and convert them into an application−specific usable format.

The implications of this are tremendous, especially for developers working on different platforms. It means that a PHP associative array could be encoded in WDDX and sent to a Perl script, which could decode it into a hash and use it for further processing, or that a Python list generated on one server could be represented as a WDDX structure and sent to another server, where a Perl, PHP or JSP script could decode it as an array and handle it appropriately. By maintaining the integrity of data structures across different environments, writing platform−independent code becomes much easier.

With applications now able to talk to each other in a common language, a huge number of new business and technological opportunities arise. Data exchange between servers − for example, B2B applications like accounting, inventory management and order fulfillment − becomes more streamlined and intelligent, and data updates − for example, news headlines or stock prices − can be delivered to requesting clients without worries about inter−operability of different operating systems or platforms.

Since WDDX abstractions are represented as text (technically, well−formed XML), they can be moved between applications using any protocol which supports transferring textual data − including HTTP and email. In fact, WDDX was written by Allaire Corporation specifically to solve the problem of data exchange in the anything−but−consistent world of the Web...and, as you'll see over the next few pages, it works beautifully.

Polly Wants A Cracker

Since WDDX data follows the basic rules of XML markup, it can easily be validated against a DTD. And if you look at the DTD, you'll quickly see that all WDDX "packets" are constructed in a standard format.

The root, or document, element for WDDX data is always the <wddxPacket> element, which marks the beginning and end of a WDDX block.

```
<wddxPacket version='1.0'>
```

This is immediately followed by a header containing comments,

```
<header>
<comment>Who da man?</comment>
</header>
```

and a data area containing the XML representation of the data structure.

```
<data>
...
</data>
</wddxPacket>
```

This data area can contain any of WDDX's basic datatypes, which are described in detail in the WDDX DTD − here's the short version, with examples:

Boolean values, represented by the element <boolean> − for example

```
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='validUser'>
<boolean value='true'/>
</var>
</struct>
</data>
</wddxPacket>
```

Numbers, represented by the element <number> − for example

```
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='page'>
<number>76</number>
</var>
</struct>
</data>
</wddxPacket>
```

String values, represented by the element <string> − for example

```
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='phrase'>
<string>Polly wants a cracker</string>
</var>
</struct>
</data>
</wddxPacket>
```

Arrays and hashes (or associative arrays), represented by the elements <array> and <struct> respectively − for example

```
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='desserts'>
<array length='5'>
<string>apple pie</string>
<string>angel food cake</string>
<string>chocolate fudge</string>
<string>strawberries and cream</string>
<string>tiramisu</string>
</array>
</var>
</struct>
</data>
```

```
</wddxPacket>
```

Tabular data, represented by the element <recordset> − for example

```
<wddxPacket version='1.0'>
<header/>
<data>
<recordset rowCount="2" fieldNames="symbol, price">
<field name="symbol">
<string>HDST</string>
<string>BDGF</string>
</field>
<field name="price">
<number>56.78</number>
<number>373.03</number>
</field>
</recordset>
</data>
</wddxPacket>
```

Timestamps, represented by the element <dateTime> − for example

```
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='today'>
<dateTime>2001−08−08T16:48:23</dateTime>
</var>
</struct>
</data>
</wddxPacket>
```

Humbert Redfinch−Northbottom The Third, I Presume?

The process of converting a data structure into WDDX is referred to as "serialization". Deserialization is, obviously, the reverse process. And your favourite language and mine, PHP, comes with a number of WDDX functions to accomplish both these (although you may need to recompile your PHP build to enable them.)

The first of these is the wddx_serialize_value() function, used to create a WDDX packet containing a single value. The function also accepts an optional second parameter, which is used to generate a comment for the packet. So the following line of code

```
<?
echo wddx_serialize_value("Humbert Redfinch−Northbottom The
Third", "Who da
man?");
?>
```

would generate the WDDX fragment

```
<wddxPacket version='1.0'>
<header>
<comment>Who da man?</comment>
</header>
<data>
<string>Humbert Redfinch−Northbottom The Third</string>
</data>
</wddxPacket>
```

Once a value has been serialized, it can be reconstituted into its original form with the wddx_deserialize() function, used to convert a WDDX packet into a native data structure. Consider the following example,

```
<?
// serialize
$packet = wddx_serialize_value("Humbert Redfinch−Northbottom
The Third");



// deserialize
$original_string = wddx_deserialize($packet);
```

```
// print
echo "My name is $original_string";
?>
```

which generates the output

```
My name is Humbert Redfinch−Northbottom The Third
```

This also works with arrays − the following code

```
<?
$star_wars = array("princess" => "Leia", "teacher" => "Yoda",
"new hope" =>
"Luke", "bad guy" => "Darth", "worse guy" => "The Emperor");



$packet = wddx_serialize_value($star_wars);



$data = wddx_deserialize($packet);



echo $data['princess'];
?>
```

outputs

```
Leia
```

Old Friends And New

You can serialize more than one variable at a time with the wddx_serialize_vars() function, which accepts a list of variable names and creates a single WDDX packet representing the entire set. The code snippet

```
<?
// variables
$friends = array("Rachel", "Phoebe", "Monica", "Chandler",
"Joey", "Ross");



$total = 34238;



$phrase = "The wild blue fox jumped over the indigo
submarine";



$error_flag = false;



// serialize
echo wddx_serialize_vars("friends", "total", "phrase",
"error_flag");
?>
```

would generate the WDDX packet

```
<wddxPacket version='1.0'>
<header/>
<data>

<struct>

<var name='friends'>
<array length='6'>
<string>Rachel</string>
<string>Phoebe</string>
<string>Monica</string>
<string>Chandler</string>
<string>Joey</string>
```

```
<string>Ross</string>
</array>
/var>

<var name='total'>
<number>34238</number>
</var>

<var name='phrase'>
<string>The wild blue fox jumped over the indigo
submarine</string>
</var>

<var name='error_flag'>
<boolean value='false'/>
</var>

</struct>

</data>
</wddxPacket>
```

Note that the line breaks have been added by me for clarity − PHP generates the entire packet as a single string.

You can also serialize an associative array

```
<?
// set up array
$star_wars = array('princess' => 'Leia', 'teacher' => 'Yoda',
'new hope' =>
'Luke', 'bad guy' => 'Darth', 'worse guy' => 'The Emperor');

// serialize
echo wddx_serialize_vars("star_wars");
?>
```

into this WDDX representation:

```
<wddxPacket version='1.0'>
<header/>
<data>
<struct>
<var name='star_wars'>
```

```
<struct>
<var name='princess'>
<string>Leia</string>
</var>

<var name='teacher'>
<string>Yoda</string>
</var>

<var name='new hope'>
<string>Luke</string>
</var>

<var name='bad guy'>
<string>Darth</string>
</var>

<var name='worse guy'>
<string>The Emperor</string>
</var>
</struct>

</var>
</struct>
</data>
</wddxPacket>
```

Wanna really cause some heartburn? Try serializing an array of arrays.

```
<?
// array of arrays
$all_mixed_up = array(
array("red", "green", "blue"),
array("laurel", "hardy"),
array("macaroni", "spaghetti", "lasagne", "fettucine"),
array("Spiderman", "Superman", "Human Torch", "Batman"),
array("princess" => "Leia", "teacher" => "Yoda", "new hope" =>
"Luke",
"bad guy" => "Darth", "worse guy" => "The Emperor")
);

// serialize
echo wddx_serialize_vars("all_mixed_up");

?>
```

Here's the WDDX representation:

```
<wddxPacket version='1.0'>
<header/>
<data>

<struct>
<var name='all_mixed_up'>
<array length='5'>
<array length='3'>
<string>red</string>
<string>green</string>
<string>blue</string>
</array>

<array length='2'>
<string>laurel</string>
<string>hardy</string>
</array>

<array length='4'>
<string>macaroni</string>
<string>spaghetti</string>
<string>lasagne</string>
<string>fettucine</string>
</array>

<array length='4'>
<string>Spiderman</string>
<string>Superman</string>
<string>Human Torch</string>
<string>Batman</string>
</array>

<struct>
<var name='princess'>
<string>Leia</string>
</var>
<var name='teacher'>
<string>Yoda</string>
</var>
<var name='new hope'>
<string>Luke</string>
</var>
<var name='bad guy'>
<string>Darth</string>
</var>
<var name='worse guy'>
```

```
<string>The Emperor</string>
</var>
</struct>
</array>

</var>
</struct>
</data>
</wddxPacket>
```

Note that when this structure is deserialized, it will result in an associative array containing the single key "star_wars", which points to an array of the original values. The following example demonstrates this:

```
<?
// array of arrays
$all_mixed_up = array(
array("red", "green", "blue"),
array("laurel", "hardy"),
array("macaroni", "spaghetti", "lasagne", "fettucine"),
array("Spiderman", "Superman", "Human Torch", "Batman"),
array("princess" => "Leia", "teacher" => "Yoda", "new hope" =>
"Luke",
"bad guy" => "Darth", "worse guy" => "The Emperor")
);

// serialize
$packet = wddx_serialize_vars("all_mixed_up");

// and deserialize (as associative array)
$structure = wddx_deserialize($packet);

// returns Array
echo $structure;

// retrieve keys
$keys = array_keys($structure);
// returns "all_mixed_up"
echo $keys[0];

// returns Array (first element of original $all_mixed_up)
echo $structure['all_mixed_up'][0];

// returns "red"
echo $structure['all_mixed_up'][0][0];

// returns "Yoda"
echo $structure['all_mixed_up'][4]['teacher'];
```

```
?>
```

Hip To Be Square

The wddx_packet_start(), wddx_add_vars() and wddx_packet_end() functions are used in tandem, to create a WDDX packet and add variables to it one after another. This comes in handy if you're adding values in a loop, or retrieving them from a database. Consider the following example, which creates a WDDX structure containing the squares of numbers between 1 and 10.

```
<?
// variable prefix
$prefix="square_of_";

// create packet
$wp = wddx_packet_start();

// add variables to it
for ($x=1; $x<=10; $x++)
{
// dynamically generate variable name
$varname = $prefix . $x;
$$varname = pow($x, 2);
// add to packet
wddx_add_vars($wp, "$prefix$x");
}

// end and print
echo wddx_packet_end($wp);
?>
```

In this case, I'm first starting a new WDDX packet with the wddx_packet_start() function, which returns a WDDX resource handle for further use; this handle is used for all subsequent operations.

Next, I generate the squares of all numbers between 1 and 10 using a loop, dynamically generate a variable name to hold the value, and add this variable to the packet via the wddx_add_vars() function. This function works exactly like wddx_serialize_vars() − you can specify a list of variables to be added if you like − and requires you to specify the WDDX resource handle generated in the first step.

Once the packet has been generated, the wddx_packet_end() function is used to add the closing tags to the generated WDDX packet.

Here's the output:

```
<wddxPacket version='1.0'><header/>
<data>
<struct>
<var name='square_of_1'><number>1</number></var>
```

```
<var name='square_of_2'><number>4</number></var>
<var name='square_of_3'><number>9</number></var>
<var name='square_of_4'><number>16</number></var>
<var name='square_of_5'><number>25</number></var>
<var name='square_of_6'><number>36</number></var>
<var name='square_of_7'><number>49</number></var>
<var name='square_of_8'><number>64</number></var>
<var name='square_of_9'><number>81</number></var>
<var name='square_of_10'><number>100</number></var>
</struct>
</data>
</wddxPacket>
```

The Truth Is Out There

Now that you've understood the fundamentals, let's look at a simple application of WDDX.

One of the most popular uses of this type of technology involves using it to get and display syndicated content from a content provider. Since WDDX is designed expressly for transferring data in a standard format, it excels at this type of task − in fact, the entire process can be accomplished via two very simple scripts, one running on the news server and the other running on the client.

Let's look at the server component first. We'll begin with the assumption that the content to be distributed (news headlines, in this case) are stored in a single database table, which is updated on a regular basis from an external data source. Every headline has a timestamp, which helps to identify the most recent. Consequently, we can postulate a table which looks something like this:

```
mysql> select * from news;
+-------------------------------------------------------+-----------------
--+
| slug | timestamp
|
+-------------------------------------------------------+-----------------
--+
| Alien life found on Mars | 2001−08−09
10:46:21 |
| Stem−cell controversy stirs up fresh protests | 2001−08−06
06:27:12 |
| "Planet Of The Apes" opens nationwide | 2001−08−10
13:41:18 |
| Schumacher one shy of all−time F1 record | 2001−08−04
18:23:33 |
| Lucas reveals title of upcoming Star Wars installment |
2001−08−07
10:25:30 |
| Computing power increases threefold | 2001−08−07
15:31:18 |
+-------------------------------------------------------+-----------------
--+
6 rows in set (0.00 sec)
```

Now, we need to write a script which will reside on the server, connect to this table, retrieve the four most recent headlines, and encode them as a WDDX packet

```
<?
// server.php − output WDDX packet containing four news
headlines
```

```
// database parameters
$hostname = "content_server";
$user = "wddx_agent";
$pass = "823h3459";
$database = "db6483";

// open connection to database
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

// get four newest headlines
$query = "SELECT slug FROM news ORDER BY timestamp DESC LIMIT
0,4";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// add headlines to array $serverSlugArray
if (mysql_num_rows($result) > 0)
{
while($row = mysql_fetch_row($result))
{
$serverSlugArray[] = $row[0];
}
}
mysql_close($connection);

// create WDDX packet
echo wddx_serialize_value($serverSlugArray);
?>
```

This is an extremely simple script − it performs a query to get the four most recent headlines, adds them to an array named $serverSlugArray, and then serializes and prints this array as a WDDX packet.

The requesting client now needs only to send a HTTP request to this script, read and deserialize the generated WDDX packet, and use it in whatever manner it sees fit. For this example, I'm using the data to create a scrollable tickertape containing the news items.

```
<?
// client.php − connect to server, retrieve and decode WDDX
packet

// url of Web page
$url = "http://content_server/server.php";

// read WDDX packet into string
```

```
$package = join ('', file($url));

// deserialize
$clientSlugArray = wddx_deserialize($package);

?>
<html>
<head>
<basefont face="Verdana">
</head>
<body>
<!-- this works only in IE -->
<marquee bgcolor="Black" loop="INFINITE">
<font size=-1 color=white><b>
<?
for($x=0; $x<sizeof($clientSlugArray); $x++)
{
echo $clientSlugArray[$x] . "       ";
}
?>
</b></font>
</marquee>
</body>
</html>
```

The first three lines are the most important − they take care of reading the URL into a single string (using standard file functions) and deserializing the WDDX packet into a native PHP array called $clientSlugArray. The remainder of the script simply iterates through this array and prints the elements (news headlines) within a <marquee> tag.

As you can see, using WDDX as the transport mechanism between servers offers a number of advantages − it's simple, flexible and far more efficient than coding your own solution to the problem. In fact, for certain business applications − content syndication is a prime example − it can save you a great deal of development and testing time.

Money Talks

Let's try another example, this one using WDDX to transmit five−day currency rate data to a requesting client. The client then uses this data to calculate an average rate for the past five days.

Let's assume that the data is stored in the following database table:

```
mysql> select * from currency;
+--------+-------+-------+-------+-------+-------+
| symbol | mark1 | mark2 | mark3 | mark4 | mark5 |
+--------+-------+-------+-------+-------+-------+
| GBP | 0.72 | 0.69 | 0.73 | 0.75 | 0.69 |
| INR | 0.023 | 0.045 | 0.012 | 0.019 | 0.025 |
| DZD | 0.01 | 0.009 | 0.015 | 0.011 | 0.01 |
| CAD | 0.66 | 0.65 | 0.68 | 0.7 | 0.64 |
| DEM | 0.43 | 0.44 | 0.43 | 0.42 | 0.42 |
+--------+-------+-------+-------+-------+-------+
5 rows in set (0.00 sec)
```

The server script needs to get all five values and encode them as a WDDX packet.

```
<?
// server.php − output WDDX packet containing currency data

// database parameters
$hostname = "medusa";
$user = "wddx_agent";
$pass = "jser745mf";
$database = "trends";

// open connection to database
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

// get rates for last five days
$query = "SELECT mark1, mark2, mark3, mark4, mark5 FROM
currency WHERE
symbol = '$symbol'";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// add data to packet
if (mysql_num_rows($result) > 0)
```

```
{
$mark = mysql_fetch_row($result);
}
mysql_close($connection);

// print packet
echo wddx_serialize_value($mark);
?>
```

Again, the array of five values is encoded as a WDDX packet, and sent to the requesting client, which decodes it and prints the data, together with an average of all five values.

```
<?
// client.php − connect to server, retrieve and decode WDDX
packet

// url of Web page
$url = "http://medusa/server.php?symbol=$symbol";

// read WDDX packet into string
$package = join ('', file($url));

// deserialize
$rates = wddx_deserialize($package);

?>
<html>
<head>
<basefont face="Verdana">
</head>
<body>
<h3>Five−day currency trends for currency symbol <? echo
$symbol; ?></h3>

<table border="1" cellspacing="3" cellpadding="3">
<tr>
<td>Data pointer 1</td>
<td>Data pointer 2</td>
<td>Data pointer 3</td>
<td>Data pointer 4</td>
<td>Data pointer 5</td>
<td>Average</td>
</tr>
<tr>
<?
$sum = 0;
```

```
// print rate data
for($x=0; $x<5; $x++)
{
$sum = $sum + $rates[$x];
echo "<td>$rates[$x]</td>";
}

// calculate average
$avg = $sum/5;
echo "<td>$avg</td>";
?>
</tr>
</table>

</body>
</html>
```

Simple, huh?

Closing Time

That's about it from me. If you'd like to learn more about WDDX, I'd recommend that you visit the following links:

The WDDX Web site, at http://www.openwddx.org/

The WDDX DTD, at http://www.allaire.com/documents/objects/whitepapers/wddx_dtd.txt

PHP's WDDX functions, at http://www.php.net/manual/en/ref.wddx.php

The WDDX.pm Perl module, at http://www.scripted.com/wddx/

Moreover.com, which offers free news headlines in WDDX format (non−commercial use only), at http://www.moreover.com/

Until next time...stay healthy!