

# 1 CREATING A WEB SERVICE

This project has chosen to use the **ARK** projects for creating a web service which can serve the data from any **ARK** project as **XML**. For creating this web service the **SOAP** protocol will be used to send and receive data through **HTTP**.

## 1.1 MS4W SERVER

**SOAP** needs an **Apache** server to run from and since this project will also need **Mapserver** to create a **WFS** server and the **ARK** interface, the project has chosen the server package **ms4w** to run everything on. The **ms4w** package is created for windows and contains an **Apache** server with **PHP5** and **Mapserver** together with several other tools which are useful for serving spatial data online.

Download the **ms4w** package and unzip the file into the root folder. If the C drive is chosen this will create a series of subfolders under C:\ms4w\.

To install the server it is necessary to run \ms4w\apache-install.bat (at the command line or by double-clicking on it). A DOS window should pop up with the following message:

```
The Apache MS4W Web Server service is starting.
```

```
The Apache MS4W Web Server service was started successfully.
```

If this happens **Apache** is successfully up and running and will automatically start each time the machine is restarted. To test that **Apache** is running properly open a web browser and type in:

```
http://localhost/ or http://127.0.0.1/
```

If the main **ms4w** page turns up then the package is installed

If the package does not install there is a troubleshooting section in the **ms4w** README file. This project has had problems with the \ms4w\apache-install.bat not executing properly and it was found that this version of **Mapserver** runs on the socket port 80 by default. Unfortunately, this port was already in use and I had to change the default setting to port 8080 instead. This was done in the file /ms4w/Apache/conf/httpd.conf where the following change was made on line 120:

```
line 120: Listen 80
line 120: Listen 8080
```

After this change was made the MS4W main page was running in the web browser on the page:

```
http://localhost:8080/
```

## 1.2 INSTALLING PEAR PACKAGES

Since **ARK** already uses **PHP** with some **PEAR** protocols this web service will also make use of **PHP** and the **PEAR:SOAP** package.

Download **PEAR** and the **PEAR** package manager. Once this is present run the pear.bat program on the command line:

```
C:\ms4w\Apache\php\pear.bat download-all
```

Downloads all available packages to the server – go to C:\ms4w\Apache\php to see all the package names.

```
C:\ms4w\Apache\php\pear.bat install -f <package>
```

This force (-f) installs the package (<package>) that you have chosen for example:

```
C:\ms4w\Apache\php\pear.bat install -f XML_XPath
```

To get other packages go to the website for **PEAR**.

## 1.3 SOAP SERVER

The **SOAP** server (ark\_webservice/ark\_server.php) is where the web service is initialised and this script receives the **SOAP** message and executes the functions called in the following order:

1. Require the server.php functions from **PEAR:SOAP** package

```
require_once 'C:/ms4w/Apache/php/PEAR/SOAP/Server.php';
```

2. Include the class and functions file for the service - is in the same folder

```
include("ark_service_class.php");
```

3. Runs the arkServer function

```
$arkServer = new arkServer;
```

4. Creates a new **SOAP** server

```
$server = new SOAP_server;
```

5. Adds the class to the object map together with a namespace which must be the same as the namespace in the client

```
$server->addObjectMap($arkServer, "http://localhost:8080/ark_webservice");
```

6. Handle **SOAP** requests coming in as POST data

```
if (isset($_SERVER['REQUEST_METHOD']) &&
    $_SERVER['REQUEST_METHOD']=='POST') {
    $server->service($_HTTP_RAW_POST_DATA);
} else {
```

7. The DISCO server outputs instructions if the server web service is called with no **SOAP** message

```
require_once 'C:/ms4w/Apache/php/PEAR/SOAP/Disco.php';
```

8. Create the Disco server

```
$disco = new SOAP_DISCO_Server($server, 'arkServer');
```

9. Output as **XML**

```
header("Content-type: text/xml");
```

10. If server.php?wsdl is called return the **WSDL** message as xml

```
if (isset($_SERVER['QUERY_STRING']) &&
    strcasecmp($_SERVER['QUERY_STRING'], 'wsdl')==0) {
    echo $disco->getWSDL();
} else {
```

11. If there is no POST data and the **WSDL** is not called the disco will output instructions for getting the **WSDL**

```
    echo $disco->getDISCO();
}
}
```

12. If the server page is called with no **SOAP** message sent then the DISCO server will output instructions for reaching the **WSDL** message:

```
<disco:discovery>
<sc1:contractRef ref="http://localhost:8080/ark_webservice/ark_server.php?wsdl"/>
</disco:discovery>
```

13. This wsdl message can also be called by viewing the page:

```
ark_server.php?wsdl
```

This will output the **WSDL** message as **XML**:

1. The **WSDL** message is an xml file

```
<?xml version="1.0"?>
<definitions name="arkServer" targetNamespace="urn:arkServer"
xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/" xmlns:tns="urn:arkServer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns="http://schemas.xmlsoap.org/wSDL/">
```

2. If any types where set

```
<types xmlns="http://schemas.xmlsoap.org/wSDL/">
</types>
```

3. List of messages two for each operation

```
<message name="getModuleRequest">
  <part name="ark_name" type="xsd:string" />
</message>
<message name="getModuleResponse">
  <part name="listOfModules" type="xsd:string" />
</message>
<message name="getQueryFieldsRequest">
  <part name="ark_name_and_mod_key" type="xsd:array" />
</message>
<message name="getQueryFieldsResponse">
  <part name="listOfQueryFields" type="xsd:string" />
</message>
<message name="getItemListRequest">
  <part name="ark_name_and_mod_key_and_field_name_and_or_field_value"
type="xsd:array" />
</message>
<message name="getItemListResponse">
  <part name="listOfItems" type="xsd:string" />
</message>
<message name="getItemRequest">
  <part name="ark_name_and_mod_key_and_item_value" type="xsd:array" />
</message>
<message name="getItemResponse">
  <part name="oneItem" type="xsd:string" />
</message>
```

4. The portType is a arkServerPort and contains the information about each operation

```
<portType name="arkServerPort">
<operation name="getModule">
  <input message="tns:getModuleRequest" />
  <output message="tns:getModuleResponse" />
</operation>
<operation name="getQueryFields">
  <input message="tns:getQueryFieldsRequest" />
  <output message="tns:getQueryFieldsResponse" />
</operation>
<operation name="getItemList">
  <input message="tns:getItemListRequest" />
  <output message="tns:getItemListResponse" />
</operation>
<operation name="getItem">
  <input message="tns:getItemRequest" />
  <output message="tns:getItemResponse" />
</operation>
</portType>
```

5. The binding contains information for each operation

```
<binding name="arkServerBinding" type="tns:arkServerPort">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getModule">
```

```

      <soap:operation
soapAction="http://localhost:8080/ark_webservice#arkServer#getModule" />
      <input>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
    <operation name="getQueryFields">
      <soap:operation
soapAction="http://localhost:8080/ark_webservice#arkServer#getQueryFields" />
      <input>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
    <operation name="getItemList">
      <soap:operation
soapAction="http://localhost:8080/ark_webservice#arkServer#getItemList" />
      <input>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
    <operation name="getItem">
      <soap:operation
soapAction="http://localhost:8080/ark_webservice#arkServer#getItem" />
      <input>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded"
namespace="http://localhost:8080/ark_webservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>

```

6. The service is called the arkServerService and here the port is defined.

```

<service name="arkServerService">
  <documentation />
  <port name="arkServerPort" binding="tns:arkServerBinding">
    <soap:address location="http://localhost:8080/ark_webservice/ark_server.php" />
  </port>
</service>
</definitions>

```

This **WSDL** message tells the client that there are four methods/operations in this web service (e.g. getItem) and that each method takes certain parameters. For example the getItem method takes an array as parameters which has the parameters ark\_name, mod\_key and item\_value in it.

## 1.4 SOAP CLASS AND FUNCTIONS

Each operation/method called through the **SOAP** message has a corresponding function on the server side which when executed returns **XML** which again is sent back to the client. For this web service the functions and the class arkService have been collected in a separate **PHP** file (ark\_webservice/ark\_service\_class.php) which will be explained here together with the output of each method.

### 1. Defining the class arkServer

```
class arkServer {
```

### 2. Sets the variable \_\_dispatch\_map which will help create the **WSDL** view

```
var $__dispatch_map = array()
```

### 3. Sets the variable dbconn which creates the connection to the database

```
var $dbconn;
```

### 4. Function called by ark\_server.php which will initialise the db connection and the **WSDL** view

```
function arkServer() {
```

#### 4.1 Define the signature of the dispatch map for the **WSDL** file - one for each method of the service

```
$this->__dispatch_map['getModule'] =
    array('in' => array('ark_name' => 'string'),
          'out' => array('listOfModules' => 'string'),
    );

$this->__dispatch_map['getQueryFields'] =
    array('in' => array('ark_name_and_mod_key' => 'array'),
          'out' => array('listOfQueryFields' => 'string'),
    );

$this->__dispatch_map['getItemList'] =
    array('in' => array('ark_name_and_mod_key_and_field_name_and_or_field_value'
=> 'array'),
          'out' => array('listOfItems' => 'string'),
    );

$this->__dispatch_map['getItem'] =
    array('in' => array('ark_name_and_mod_key_and_item_value' => 'array'),
          'out' => array('oneItem' => 'string'),
    );
```

#### 4.2. Initiates the database connection

```
$sql_host = "localhost";
$sql_user = "*****";
$sql_pwd = "*****";

$this->dbconn = mysql_connect($sql_host, $sql_user, $sql_pwd);
} // End of arkServer function
```

## 5. Function required by SOAP\_server to create the wsdl output

```
function __dispatch($methodname) {
    if (isset($this->__dispatch_map[$methodname]))
        return $this->__dispatch_map[$methodname];
    return NULL;
}
```

## 6. FUNCTION getModule

```
function getModule($ark_name)
{
```

## 6.1 DATABASE

```
mysql_select_db($ark_name)or die("Could not find database");
```

## 6.2 SQL

```
$sql = "
SELECT cor_tbl_module.description, cor_tbl_module.shortform, cor_tbl_module.itemkey,
cor_tbl_alias.alias
FROM cor_tbl_module
LEFT JOIN cor_tbl_alias
ON cor_tbl_module.id = cor_tbl_alias.itemvalue AND 'cor_tbl_module' =
cor_tbl_alias.itemkey AND cor_tbl_alias.language='en'
";
$result = mysql_query($sql);
```

## 6.3 Outputting the XML header

```
$xml_output = "<?xml version='1.0'>\n";
```

## 6.4 Beginning the XML with the root element ark and the schema

```
$xml_output .= "<ark xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://ay-
dehus.soton.ac.uk:8080/students/hro106/ark_webservice/schema/ark.xsd'>\n";
```

## 6.5 First element child the arkName element

```
$xml_output .= "\t<arkName>" . $ark_name . "</arkName>\n";
```

## 6.6 Get rows from sql

```
$nrows = mysql_num_rows($result);
```

## 6.7 If there are any rows

```
if(!$nrows==0){
    $xml_output .= "\t<modules>\n";
```

## 6.8 For each row in the sql get xml

```
for($i=0;$i<$nrows;$i++) {
    $row = mysql_fetch_assoc($result);
    $xml_output .= "\t\t<module>\n";
    $xml_output .= "\t\t\t<modAlias>" . $row['alias'] . "</modAlias>\n";
    $xml_output .= "\t\t\t<modDescription>" . $row['description'] .
        "</modDescription>\n";
    $xml_output .= "\t\t\t<modShortForm>" . $row['shortform'] .
        "</modShortForm>\n";
    $xml_output .= "\t\t\t<modKey>" . $row['itemkey'] . "</modKey>\n";
    $xml_output .= "\t\t</module>\n";
}
$xml_output .= "\t</modules>";
}
$xml_output .= "</ark>";
return $xml_output;
```

```

} // FUNCTION getModule ENDS

- <ark xsi:noNamespaceSchemaLocation="http://ay-dehus.soton.ac.uk:8080/students/hro106/ark_webservice/schema/ark.xsd">
  <arkName>sintana_ark</arkName>
  - <modules>
    - <module>
      <modAlias>Sintana</modAlias>
      <modDescription>The Sintana module</modDescription>
      <modShortForm>sin</modShortForm>
      <modKey>sin_cd</modKey>
    </module>
    - <module>
      <modAlias>Bibliography</modAlias>
      <modDescription>Bibliographic module</modDescription>
      <modShortForm>bib</modShortForm>
      <modKey>bib_cd</modKey>
    </module>
  </modules>
</ark>

```

*This is how the output of the getModules method would look if called with the parameter: ark\_name=sintana\_ark.*

## 7. FUNCTION getQueryFields

```

function    getQueryFields($param)
{

```

### 7.1 PARAMETERS

```

$ark_name = $param->ark_name;
$mod_key = $param->mod_key;

```

### 7.2 DATABASE

```

mysql_select_db($ark_name)or die("Could not find database");

```

7.3 Types available in this ark service (standard are: txt, number, data, attribute and action) If more are added to the array a method must be specified.

```

$type = array ('txt', 'number', 'date', 'action');

```

### 7.4 Number of instances in the array called type

```

$type_count = count($type);

```

### 7.5 SQL for getting the module stuff from the module key

```

$sql_module = "
SELECT cor_tbl_module.shortform, cor_tbl_module.itemkey, cor_tbl_alias.alias
FROM cor_tbl_module
LEFT JOIN cor_tbl_alias
ON cor_tbl_module.id = cor_tbl_alias.itemvalue AND 'cor_tbl_module' =
   cor_tbl_alias.itemkey
WHERE cor_tbl_module.itemkey = '$mod_key'

```

```

";
$result_module = mysql_query($sql_module)or die("data not found");

```

## 7.6 Outputting the XML header

```
$xml_output = "<?xml version=\"1.0\"?>\n";
```

## 7.7 Beginning the XML with the root element ark and the schema

```

$xml_output .= "<ark xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://ay-
dehus.soton.ac.uk:8080/students/hro106/ark_webservice/schema/ark.xsd'>\n";

```

## 7.8 First element child the arkName element

```
$xml_output .= "\t<arkName>" . $ark_name . "</arkName>\n";
```

## 7.9 Get the number of rows in the module sql

```
$nrows_module = mysql_num_rows($result_module);
```

## 7.10 If there is a module with this name - if not returns an error

```

if(!$nrows_module==0) {
$xml_output .= "\t<modules>\n";

```

## 7.11 For each module - there is only one though

```

for($i=0;$i<$nrows_module;$i++) {
$row_module = mysql_fetch_assoc($result_module);
$xml_output .= "\t\t<module>\n";
$xml_output .= "\t\t\t<modKey>" . $row_module['itemkey'] .
"</modKey>\n";
$xml_output .= "\t\t\t<modAlias>" . $row_module['alias'] .
"</modAlias>\n";
$xml_output .= "\t\t\t<fields>\n";

```

## 7.12 Looping through the field types in the array above

```
for($i=0;$i<$ftype_count;$i++){
```

## 7.13 Adding each type to the field\_type

```
$field_type = $ftype[$i];
```

## 7.14 Adding the right type to the sql bits

```

$cor_lut_fieldtype = "cor_lut_" . $field_type . "type";
$cor_tbl_field = "cor_tbl_" . $field_type;
$cor_fieldtype = $field_type . "type";

```

## 7.15 Running the sql with the right type

```

$sql_field = "
SELECT DISTINCT $cor_lut_fieldtype.$cor_fieldtype AS fieldname,
cor_tbl_alias.alias AS fieldalias,
$cor_tbl_field.$cor_fieldtype AS fieldid
FROM $cor_tbl_field
LEFT JOIN $cor_lut_fieldtype ON $cor_tbl_field.$cor_fieldtype =
$cor_lut_fieldtype.id

```



8.1 PARAMETERS: From the mod\_key the module shortform has been extracted. The variables field\_name and field\_value are set up so that they are empty if they are not there.

```
$ark_name = $param->ark_name;
$mod_key = $param->mod_key;
$mod_explode = explode('_', $mod_key);
$mod_short = $mod_explode[0];
if($param->field_name){
    $field_name = $param->field_name;
}else{
    $field_name = FALSE;
}
if($param->field_value){
    $field_value = $param->field_value;
}else{
    $field_value = FALSE;
}
```

8.2 DATABASE

```
mysql_select_db($ark_name)or die("Could not find database");
```

8.3 TYPE STUFF: Types available in this ark service (standard are: txt, number, data, attribute and action) If more are added to the array a method must be specified.

```
$ftype = array ('txt', 'number', 'date', 'action');
```

8.4 Number of instances in the array called type

```
$ftype_count = count($ftype);
```

8.5 SQL for getting the module stuff from the module key

```
$sql_module = "
SELECT cor_tbl_module.shortform, cor_tbl_module.itemkey, cor_tbl_alias.alias
FROM cor_tbl_module
LEFT JOIN cor_tbl_alias
ON cor_tbl_module.id = cor_tbl_alias.itemvalue AND 'cor_tbl_module' =
cor_tbl_alias.itemkey
WHERE cor_tbl_module.itemkey = '$mod_key' AND cor_tbl_alias.language='en'
";
$result_module = mysql_query($sql_module)or die("data not found");
```

8.6 SQL for getting the item values for a specific module

```
$mod_table = $mod_short . "_tbl_" . $mod_short;
$item_val = $mod_short . "_cd" ;
$sql_item = "
SELECT $mod_table.$item_val as itemvalue
FROM $mod_table
";
$result_item = mysql_query($sql_item)or die("data not found");
```

8.7 XML OUTPUT

```
$xml_output = "<?xml version='1.0'>\n";
$xml_output .= "<ark xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://ay-
dehus.soton.ac.uk:8080/students/hro106/ark_webservice/schema/ark.xsd'>\n";
$xml_output .= "\t<arkName> " . $ark_name . "</arkName>\n";
```



8.13 Use the field\_type variable to create stuff for the sql for each type

```

$cor_lut_fieldtype = "cor_lut_" . $field_type . "type";
$cor_tbl_field = "cor_tbl_" . $field_type;
$cor_fieldtype = $field_type . "type";

```

8.14 Switch for what to do for each field type

```

switch($field_type)
{
case action:

```

8.15 Special search of the fields for action type

```

$mysql_itemlist = "
SELECT DISTINCT $cor_lut_fieldtype.$cor_fieldtype AS fieldname,
                cor_tbl_alias.alias AS fieldalias,
$cor_tbl_field.$cor_fieldtype AS fieldid,
                $cor_tbl_field.itemvalue AS itemvalue, cor_tbl_txt.txt as
fieldvalue
FROM $cor_tbl_field
LEFT JOIN $cor_lut_fieldtype ON $cor_tbl_field.$cor_fieldtype =
                $cor_lut_fieldtype.id
LEFT JOIN cor_tbl_alias ON $cor_tbl_field.$cor_fieldtype =
                cor_tbl_alias.itemvalue AND '$cor_lut_fieldtype' =
cor_tbl_alias.itemkey
LEFT JOIN cor_tbl_txt ON $cor_tbl_field.actor_itemvalue =
                cor_tbl_txt.itemvalue AND 'abk_cd' = cor_tbl_txt.itemkey
WHERE $cor_tbl_field.itemkey = '$mod_key' AND
cor_tbl_alias.language='en' AND $cor_lut_fieldtype.$cor_fieldtype =
'$field_name'
";
$result_itemlist = mysql_query($mysql_itemlist)or die("data not
found");
break;

case txt:
case number:
case date:
if($field_type=="txt" && $field_value!=FALSE){

```

8.16 Search of the fields for txt if a field value has been given

```

$mysql_itemlist = "
SELECT DISTINCT $cor_lut_fieldtype.$cor_fieldtype AS
fieldname,
                cor_tbl_alias.alias AS fieldalias,
$cor_tbl_field.$cor_fieldtype AS
                fieldid,
$cor_tbl_field.itemvalue AS itemvalue,
                $cor_tbl_field.$field_type as fieldvalue
FROM $cor_tbl_field
LEFT JOIN $cor_lut_fieldtype ON
$cor_tbl_field.$cor_fieldtype =
                $cor_lut_fieldtype.id
LEFT JOIN cor_tbl_alias ON $cor_tbl_field.$cor_fieldtype =
                cor_tbl_alias.itemvalue AND '$cor_lut_fieldtype' =
                cor_tbl_alias.itemkey
WHERE $cor_tbl_field.itemkey = '$mod_key' AND
                cor_tbl_alias.language='en' AND
$cor_lut_fieldtype.$cor_fieldtype =
                '$field_name' AND
$cor_tbl_field.$field_type LIKE '%$field_value%'
";
$result_itemlist = mysql_query($mysql_itemlist)or die("data
not found");
}else{

```

8.17 Search of the fields for txt, number and dates if no field value has been given

```

$mysql_itemlist = "

```



```

- <ark xsi:noNamespaceSchemaLocation="http://ay-dehus.soton.ac.uk:8080/students/hro106/ark_web/service/schema/ark.xsd">
  <arkName>sintana_ark</arkName>
  - <modules>
    - <module>
      <modKey>sin_cd</modKey>
      <modAlias>Sintana</modAlias>
      - <items>
        - <item>
          <itemValue>SIN_2</itemValue>
          - <fields>
            - <field>
              <fieldType>txt</fieldType>
              <fieldId>74</fieldId>
              <fieldName>placename</fieldName>
              <fieldAlias>Place name</fieldAlias>
              <fieldValue>Alexandru Odobescu </fieldValue>
            </field>
          </fields>
        </item>
        + <item></item>
        + <item></item>
        + <item></item>
        + <item></item>
      </items>
    </module>
  </modules>
</ark>

```

*This is how the xml output of the getItemList method would look if called with the parameters: ark\_name=sintana\_ark and mod\_key=sin\_cd and field\_name=placename.*

## 9. FUNCTION getItem

```

function getItem($param)
{

```

### 9.1 PARAMETERS

```

$ark_name = $param->ark_name;
$mod_key = $param->mod_key;
$item_value = $param->item_value;

```

### 9.2 DATABASE CONNECTION

```

mysql_select_db($ark_name)or die("Could not find database");

```

9.3 TYPE STUFF: Types available in this ark service (standard are: txt, number, data, attribute and action) If more are added to the array a method must be specified.

```

$type = array ('txt', 'number', 'date', 'action');

```

### 9.4 Number of instances in the array called type

```

$type_count = count($type);

```

### 9.5 Sql for getting module specific stuff

```

$sql_module = "
SELECT cor_tbl_module.shortform, cor_tbl_module.itemkey, cor_tbl_alias.alias
FROM cor_tbl_module
LEFT JOIN cor_tbl_alias

```

```

ON cor_tbl_module.id = cor_tbl_alias.itemvalue AND 'cor_tbl_module' =
cor_tbl_alias.itemkey
WHERE cor_tbl_module.itemkey = '$mod_key' AND cor_tbl_alias.language='en'
";
$result_module = mysql_query($sql_module)or die("data not found");

```

## 9.6 2 pieces of sql for links - one from the left and one from the right of the xmi

```

$sql_xmyleft = "
SELECT cor_tbl_xmi.xmi_itemkey AS xmikey, cor_tbl_xmi.xmi_itemvalue AS xmivalue,
cor_tbl_alias.alias AS modalias
FROM cor_tbl_xmi
LEFT JOIN cor_tbl_module ON cor_tbl_xmi.xmi_itemkey = cor_tbl_module.itemkey
LEFT JOIN cor_tbl_alias
ON cor_tbl_module.id = cor_tbl_alias.itemvalue AND 'cor_tbl_module' =
cor_tbl_alias.itemkey
WHERE cor_tbl_xmi.itemkey = '$mod_key' AND cor_tbl_xmi.itemvalue = '$item_value' AND
cor_tbl_alias.language='en'
";
$result_xmyleft = mysql_query($sql_xmyleft)or die("data not found");

$sql_xmiright = "
SELECT cor_tbl_xmi.itemkey AS xmikey, cor_tbl_xmi.itemvalue AS xmivalue,
cor_tbl_alias.alias AS modalias
FROM cor_tbl_xmi
LEFT JOIN cor_tbl_module ON cor_tbl_xmi.itemkey = cor_tbl_module.itemkey
LEFT JOIN cor_tbl_alias
ON cor_tbl_module.id = cor_tbl_alias.itemvalue AND 'cor_tbl_module' =
cor_tbl_alias.itemkey
WHERE cor_tbl_xmi.xmi_itemkey = '$mod_key' AND cor_tbl_xmi.xmi_itemvalue =
'$item_value' AND cor_tbl_alias.language='en'
";
$result_xmiright = mysql_query($sql_xmiright)or die("data not found");

```

## 9.7 XML OUTPUT

```

$xml_output = "<?xml version=\"1.0\" encoding='UTF-8'?>\n";
$xml_output .= "<ark xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='http://ay-
dehus.soton.ac.uk:8080/students/hro106/ark_webservice/schema/ark.xsd'>\n";
$xml_output .= "\t<arkName>\" . $ark_name . "</arkName>\n";

```

## 9.8 MODULE STUFF: Check for modules in sql and loop through the rows to get output

```

$rows_module = mysql_num_rows($result_module);
if(!$rows_module==0) {
    $xml_output .= "\t<modules>\n";
    for($j=0;$j<$rows_module;$j++) {
        $row_module = mysql_fetch_assoc($result_module);
        $xml_output .= "\t\t<module>\n";
        $xml_output .= "\t\t\t<modKey>\" . $row_module['itemkey'] .
        "</modKey>\n";
        $xml_output .= "\t\t\t<modAlias>\" . $row_module['alias'] .
        "</modAlias>\n";
        $xml_output .= "\t\t\t<items>\n";
        $xml_output .= "\t\t\t\t<item>\n";
        $xml_output .= "\t\t\t\t\t<itemValue>\" . $item_value .
        "</itemValue>\n";
    }
}

```

## 9.9 FIELD STUFF

```

$xml_output .= "\t\t\t\t\t<fields>\n";

```

## 9.10 Looping through the field types in the array above

```

for($i=0;$i<$ftype_count;$i++){

```

## 9.11 Adding each type to the field\_types

```

    $field_type = $ftype[$i];

```





```

        $xml_output .= "\t\t\t\t\t</links>";
    }
    $xml_output .= "\t\t\t\t\t</item>\n";
    $xml_output .= "\t\t\t\t\t</items>\n";
    $xml_output .= "\t\t\t\t\t</module>\n";
    }
    $xml_output .= "\t</modules>";
}
$xml_output .= "</ark>";

return $xml_output;
} // END of getItem Function
<ark xsi:noNamespaceSchemaLocation="http://ay-dehus.soton.ac.uk:8080/students/hro106/ark_webservice/schema/ark.xsd">
  <arkName>sintana_ark</arkName>
  - <modules>
    - <module>
      <modKey>sin_cd</modKey>
      <modAlias>Sintana</modAlias>
      - <items>
        - <item>
          <itemValue>SIN_2</itemValue>
          - <fields>
            - <field>
              <fieldType>txt</fieldType>
              <fieldId>74</fieldId>
              <fieldName>placename</fieldName>
              <fieldAlias>Place name</fieldAlias>
              <fieldValue>Alexandru Odobescu </fieldValue>
            </field>
            + <field></field>
            + <field></field>
          </fields>
          - <links>
            - <modules>
              - <module>
                <modKey>bib_cd</modKey>
                <modAlias>Bibliography</modAlias>
                - <items>
                  - <item>
                    <itemValue>SIN_2</itemValue>
                  </item>
                </items>
              </module>
            </modules>
          </links>
        </item>
      </items>
    </module>
  </modules>
</ark>

```

*This is how the xml output of the getItem method would look if called with the parameters: ark\_name=sintana\_ark and item\_value=SIN\_2.*

## 1.5 SOAP CLIENT

In theory the end-user client could create their own **SOAP** client which calls this web service based on the operations and methods shown in the **WSDL** file. However, to make the web service more accessible a **SOAP** client (ark\_webservice/ark\_client.php) has been created

which can be made to call the web service once the right variables are sent. This client sets up the parameters sent to the web service with the method and prints the response as **XML**.

## 1. Getting variables

### 1.1 Send a method name

```
if ($_GET["method"]){
    $method = $_GET["method"];
}else{
    $method = 'getModule';
}
```

### 1.2 Send a ark name

```
if ($_GET["ark_name"]){
    $ark_name = $_GET["ark_name"];
}else{
    $ark_name = 'portus_ark';
}
```

### 1.3 Send a module name key

```
$mod_key = $_GET["mod_key"];
```

### 1.4 Send a field name

```
$field_name = $_GET["field_name"];
```

### 1.5 Send a field value is any

```
$field_value = $_GET["field_value"];
```

### 1.6 Send an item value

```
$item_value = $_GET["item_value"];
```

2. Namespace must be the same as the namespace specified in the server.php under addObjectMap

```
$namespace = "http://localhost:8080/ark_webservice";
```

3. Requires the Client.php file

```
require_once 'C:/ms4w/Apache/php/PEAR/SOAP/Client.php';
```

4. initialises the soap client where the param is the server.php

```
$info = new SOAP_client("http://localhost:8080/ark_webservice/ark_server.php");
```

5. params is the stuff sent to the server (input) - must be an array

```
switch ($method)
{
    case getModule:
        $params = array(
            "ark_name" => $ark_name
        );
        break;
```

```

case getQueryFields:
    $params = array(
        'params' => array(
            'ark_name' => $ark_name,
            'mod_key' => $mod_key
        )
    );
    break;

case getItemList:
    $params = array(
        'params' => array(
            'ark_name' => $ark_name,
            'mod_key' => $mod_key,
            'field_name' => $field_name,
            'field_value' => $field_value,
        )
    );
    break;

case getItem:
    $params = array(
        'params' => array(
            'ark_name' => $ark_name,
            'mod_key' => $mod_key,
            'item_value' => $item_value
        )
    );
    break;
default:
    $params = array("ark_name" => "portus_ark");
}

```

6. Getting the response from the server through the call function with three parameters:

- 1) name of the method to call
- 2) the parameters in an array
- 3) the namespace - same as in the server.php file

```
$response = $info->call($method,$params,$namespace);
```

7. Header specifies the output in **XML**

```
header ("content-type: text/xml");
```

8. print the response

```
printf($response);
exit;
```

By calling this **SOAP** client it is possible to receive data from the web service for any **ARK** project as **XML**. This is basically the goal of the web service and further down I will look at how to use this **XML** formatted data.